ABSTRACT INTERPRETATION AND PARTITION REFINEMENT FOR MODEL CHECKING

PROEFSCHRIFT

T U E ,
R M , . . J.H. L ,
C
D
1 1996 16.00

DENNIS RENÉ DAMS

V (ZH)

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.C.M. Baeten
en
prof.dr. W. Damm.

Copromotor: dr. R.T. Gerth.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Dams, Dennis René

Abstract interpretation and partition refinement for model checking / Dennis René Dams. - Eindhoven : Eindhoven University of Technology. - Ill.
Thesis Technische Universiteit Eindhoven. - With index, ref. - With summary in Dutch.
ISBN 90-386-0078-X
Subject headings: formal methods / program verification / model checking

Copyright © 1996 by Dennis Dams
Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven.
wsindd@win.tue.nl

Cover: P.C. Mondriaan, Compositie met rood en blauw (1936).

printed and bound by CopyPrint 2000, Enschede.

Contents

Acknowledgements **vii**

1 Program Analysis and Verification **1**
1.1 Program Verification 2
1.2 Program Analysis 4
1.3 Scope of the Thesis 5
1.4 Related Work 8
1.4.1 Program analysis and Abstract Interpretation 8
1.4.2 Program verification 10

2 Preliminaries **13**
2.1 Relations and Functions 14
2.2 Lattice and Fixpoint Theory 15
2.2.1 Orderings 15
2.2.2 Functions over posets 16
2.2.3 Galois connections 17
2.3 Temporal Logic 18
2.3.1 Nextless fragments 20
2.4 Transition Systems 21
2.4.1 Interpretation of CTL* 22
2.4.2 Behavioural preorders and equivalences 25
2.4.3 Model checking 27

3 Abstraction and Preservation **29**
3.1 Introduction 30
3.1.1 Abstract Interpretation 30
3.1.2 Overview of the chapter 34
3.2 Preservation Results 34
3.2.1 Frameworks for weak preservation 35

3.2.2 Strong preservation 49
3.3 Abstract Semantics 50
3.3.1 Approximation of fixpoints 51
3.4 Related Work 54
3.5 Concluding Remarks 55

4 Abstract Interpretation of Nondeterministic Systems 57
4.1 Introduction 58
4.1.1 Overview of the chapter 58
4.2 Abstract Kripke Structures 59
4.2.1 Valuation of literals 60
4.2.2 Abstract initial states 61
4.2.3 Abstract transition relations 62
4.3 Abstract Interpretation of Programs 67
4.3.1 Example: dining mathematicians 69
4.4 Approximations 75
4.4.1 Abstract interpretation gives approximations 79
4.4.2 Dining mathematicians continued (I) 82
4.5 Optimal Abstract Interpretations 83
4.5.1 Conditions on the abstract domain 83
4.5.2 Adapting the abstract interpretation 87
4.6 Computing Approximations 87
4.7 Practical Application 89
4.8 Refinement of Abstractions 92
4.8.1 Abstraction families 93
4.9 Related Work 100
4.9.1 Comparing the simulation-based and Galois-insertion approach 103
4.9.2 [KDG95]: An application 107
4.10 Concluding Remarks 112

5 Logical Partition Refinement 115
5.1 Introduction 116
5.1.1 Overview of the chapter 119
5.2 Preliminaries 120
5.3 Companions 122
5.4 A Generic Splitting Algorithm 126
5.5 Splitting for ∀CTL 129
5.5.1 Termination 130

5.6 Splitting for $\varphi \in \forall$CTL 136
5.6.1 Example 138
5.7 Related Work 141
5.8 Concluding Remarks 142

6 Logics, Equivalences and Behavioural Partition Refinement 145
6.1 Introduction 146
6.1.1 Logics and equivalences 147
6.1.2 Equivalences and partition refinement algorithms 148
6.1.3 Overview of the chapter 148
6.2 CTL*, CTL, and Bisimulation 149
6.3 Nextless CTL* and CTL 152
6.3.1 The Next operator 152
6.3.2 CTL*(U), CTL(U) and stuttering equivalence 155
6.4 Partition Refinement Algorithms 163
6.5 Flat CTL and CTL* 167
6.5.1 flat$^-$CTL(U) and flat equivalence 170
6.5.2 flatCTL*(U) and flat star equivalence 173
6.6 Partition Refinement for Flat Logics 178
6.7 Related Work 179
6.8 Concluding Remarks 180

7 In Conclusion 183
7.1 Research Goal & Approach 184
7.2 Conclusions 185
7.2.1 Weak preservation 186
7.2.2 Strong preservation 187
7.3 Looking Ahead 189

Bibliography 191

Samenvatting (Summary in Dutch) 211

Index 215

Curriculum Vitae 219

Acknowledgements

According to the Dutch regulations for doctorates, the dissertation may contain modest words of thanks. But how can I remain modest on this point? There are so many people who have contributed to this thesis in one way or another, and many of them were so very important to me.

First of all, there is Rob Gerth, my supervisor, teacher, and co-author, who is always available to answer my questions. His broad perspective of computer science, and far beyond, has helped me many times to make the right choices — but he has never forced anything upon me. Rob: I hope we can continue our cooperation in the future.

I would like to thank Jos Baeten for being my first promotor, keeping an eye on my progress, and thinking about the future when I was too busy with the present. Most likely he is going to be my boss for the time to come too.

It is an honour to have Werner Damm as my second promotor. In December '93, he invited Rob and me to his department, where I learned the inspiring e ect of a well organised "bull session". I joined the sessions half a year later again — while I wasn't even invited —, and I hope there will be more occasions to travel to Oldenburg.

Loe Feijs was the perfect reader. He must have invested vast amounts of time in correcting spelling and style, pointing out inclarities, and uncovering mistakes in the darkest corners of proofs. During the past weeks he has generously let me use his room — which I did, up to the last square inch.

Over the years, Susanne Graf has never failed to read my articles and to provide extensive feedback. Often she knew what I was doing, or what I was trying to do, before I did so myself. And when I look through all those messages that have piled up in my e-mail folder over the past months, discussing my thesis, I am convinced that Susanne is the most patient person I know.

Like nobody else, Orna Grumberg knows how to make hard work feel like sheer fun. Thanks to Ed Clarke, I had the opportunity to work with her while visiting

CMU in the summer of '92. Those two weeks were to be the inspiration for the largest part of this thesis. I hope we can work, eat cakes, and dance many more times together.

Mike Codish is not directly involved in the research of this dissertation. Yet, he is responsible for much of the knowledge and many of the skills that were so important in its writing. But also has he never failed to be a great advisor in more personal a airs.

There are a number of people who have contributed to this thesis by providing useful feedback, pointing to related research, or just answering my many e-mailed questions. Mentioning Rob van Glabbeek, Anthony McIsaac, Purush Iyer, Peter Kelb, Doron Peled, Amir Pnueli, Joseph Sifakis, Howard Wong-Toi, Sergio Yovine, I only realise that there are many more. Thank you all.

During all those years, my collegues have provided the climate without which I would not have enjoyed being an "AIO": Alda Bouten, Arie van Deursen, Bart Knaack, Chris Verhoef, Hans Mulder, Herman Geuvers, Jozef Hooman, Kees Huizing, Michel Reniers, Peter Peters, Pleun van der Steen (who probably doesn't remember his role in my decision to become a scientist), Rob Nederpelt, Roel Bloo, Roland Bol, Ruurd Kuiper, Sjouke Mauw, Twan Basten, Twan Laan, and Wojtek Penczek. In particular I'm grateful to my roommates over the years, Jan Joris Vereijken, Javier Blanco, Paula Severi, Ping Zhou, and Tijn Borghuis, for the good company, and the discussions about work or completely di erent matters.

Starting March '95, I have spent one year in the Applied Logic group at Utrecht University. Alex, Albert, Bas, Henri, Jaco, Jan, Jan, Kees, Kees, Marc, Marco, Peter, Wan: thanks to all of you, every day has been worth the trip to Utrecht. Special thanks go to Jan Friso, for pushing me to finish, for which he unexpectly turned out to have another good reason. To Karst, for keeping the hardware and software going. And to Freek, for being a tireless source of information about fonts. Together with Niene, whom I met during the last weeks in Utrecht, I wanted to try all fonts and style packages that we kew, while Freek kept installing them.

My family and friends have always been there, to listen to me or endure me, understanding or making me understand.

To my wife and best friend, Lieve: bedankt voor al je geduld en steun. Het ei is eindelijk gelegd — en het is lente.

Finally, there are two people to whom I owe most of what I know and what I am. Pa & ma: bedankt!

Eindhoven, May 1996

Voor mijn ouders

Chapter 1

Program Analysis and Verification

Verification is the process of deciding whether a program satisfies a given specification or property, and should yield a "yes" or "no" answer. On the other hand, analysis seeks to infer any information about a program that may be useful for a certain purpose, but does not impose an exact minimum on the amount of information to be obtained. This thesis is an attempt to tackle the state-explosion problem, that occurs when the behaviour of complex programs has to be investigated, by considering an abstraction of this behaviour. We argue that in the context of program analysis, weak preservation of the abstraction's properties is su cient, while for program verification, properties need to be strongly preserved. We propose to use Abstract Interpretation for the construction of weakly preserving abstractions, and Partition Refinement for the construction of strongly preserving ones.

1.1 Program Verification

Computer programs should be correct. In industry, *testing* has traditionally been the main debugging technique. For example, "beta-releases" of a program are sent out to a group of people who are willing to use it and report on errors encountered. As soon as the rate of such error reports has decreased to an acceptable level, the program is put on the market. Other programs, like microprocessor code implemented in hardware[1], are often automatically tested by feeding them sequences of inputs and comparing the corresponding outputs to the desired ones. Because such components have to be highly dependable, e orts are made to design such *simulations* in such a way that as many of the (combinations of) functions of the processor as possible are drawn on. However, *exhaustive* testing is usually infeasible as the number of possible execution sequences is too large (or even infinite). Also, the specification of what are the desired output patterns may be very di cult. Such specifications tend to result in bulky documents in which ambiguity, inconsistency and incompleteness are hard to pinpoint.

Recent applications increasingly require programs that maintain a continuous interaction with their environment. Such systems are often *embedded*, meaning that they are an integral part of an environment that consists of some physical process. An example is a controller of a chemical plant, which monitors and influences processes taking place in reactors through sensors and actuators. Typically, these programs consist of many separate processes that communicate with each other by message passing or memory sharing. Such programs are called *reactive*, in contrast to the "old-fashioned" view of a program as something that takes some input, computes for a while, and then produces a result and terminates. Specifications of reactive systems also di er radically from those of traditional programs. Instead of implementing a certain relation between input and output, such systems are required to satisfy properties like safety (certain situations may not occur), liveness (situations have to occur), and constraints involving time and probabilities. The high complexity of reactive systems together with the often disastrous consequences of malfunctioning turns their specification, design and verification into an intricate undertaking.

In order to overcome the problems mentioned above, the scientific community has proposed the use of *formal methods*. This term covers all approaches to specification and verification based on mathematical formalisms. Their aim is to establish program correctness with mathematical rigour. Every formal approach to program correctness has three basic ingredients:

[1] In this thesis, the word *program* has a very general meaning and may denote, e.g., an algorithm written in PASCAL, but also a robot controller implemented on a chip, or a distributed airline reservation system.

- A mathematical model of the program.
- A formal language for expressing the specification.
- A methodology to establish whether the model of the program satisfies the specification.

Most approaches to formally establishing the correctness of programs can be classified according to a number of categories which are explained below.

A-priori/a-posteriori Verification of a given, complete program is called *a-posteriori* verification. Alternatively, a program may be designed hand-in-hand with a proof of its correctness — which we refer to as *a-priori* verification. In practice, this dichotomy is too simple. A program may be built up hierarchically while the correctness of the building blocks is established before combining them. A-posteriori verification of the parts then is part of a-priori verification of the whole. Also, the advent of high-level programming languages that encourage *prototyping* of programs, and the development of *executable specification languages* render a clear separation more di cult.

What is important is that the correctness of programs be established at an early stage in the development phase, so that errors can be corrected without too much e ort.

Proof-based/model-based In *proof-based* or *syntactic* methods, the notation of a program in some programming language is taken to be its model, while the specification is expressed in some powerful formal language. The meaning of elementary programming-language constructs is expressed by axioms, and that of larger constructs by inference rules in some proof system. The designer proves correctness by constructing a proof within this system. In the notation of logic, denoting the program by P and the specification by φ, this would be written as $P \vdash \varphi$.

In *model-based* or *semantic* methods, the model of a program consists of a description of all its possible behaviours in a mathematical structure like a transition system. The correctness criterion is a formula in a logic that is interpreted over such structures (e.g. temporal logic). Proving correctness then boils down to showing that the formula is satisfied by the model, i.e. $P \models \varphi$ is demonstrated. As the size of the model is potentially of the same order of magnitude as the total number of states that a program has, model-based verification easily runs into the same "*state-explosion*" problem as exhaustive testing.

Manual/automatic Here, the question is whether the designer carries through the proof by hand, or a computer does this automatically. Approaches that lie somewhere in between these extremes are often called *computer-assisted*. Typically, proof-based methods are more manual, while model-based methods are intended to be automatic.

Full verification/property verification Even if a specification can be written that captures all desired aspects of a program (not only functional ones but also those concerning e ciency, appearance, etc.), it probably is too much work to check it all. Rather, one is often more interested in checking certain aspects of the behaviour, like deadlock freedom, responsiveness, etc.

1.2 Program Analysis

Data-flow analysis (also *static* or *compile-time* analysis) is the name given to a collection of diverse methods aimed at the investigation of specific aspects of a program's behaviour. The location of errors is only one of the purposes that such analyses may serve. Another aim is the optimisation of programs through, e.g., dead-code detection or strictness and binding analysis (in functional languages). Also, information may be extracted that is useful in the compilation of programs, e.g. the identification of independent fragments of code that may be executed in parallel. Although their goals may be more diverse, data-flow analyses can be given a place in the taxonomy presented above. They are a-posteriori, model-based, automatic and property-aimed.

In [CC77], a unified formal framework for data-flow analyses is proposed, called *Abstract Interpretation*. According to this theory, every analysis can be viewed as a "non-standard execution" of the program, where concrete data values are replaced by symbolic *descriptions* of data and the operators of the programming language are given corresponding non-standard interpretations. Many applications of Abstract Interpretation have focussed on the analysis of *safety* (or *invariance*) properties, that hold in all states along all possible executions of a program.

When comparing data-flow analyses to program verification methods as discussed in the previous section, we note a di erence in "attitude". In verification, the properties to be checked are stated in advance. They form the specification that has to be validated against the program, resulting in one of two answers: "yes" or "no". On the other hand, while in a program analysis the domain of properties to be investigated is known beforehand (e.g. "possible deadlocks", "definite aliasing of variables", etc.), which *specific* information is inferred from a program is of less importance. One might say that verification seeks to answer specific questions, while analysis seeks to find any answers to less specific questions. The reason for this is the di erent use

of the results of each of the methods. In verification, the answer should eventually be "yes" and certify the correctness of the program. In analysis, any of the answers found can be employed to improve the program (e.g. by removing errors, by optimising the code, by parallellising its execution, etc.), but many questions may remain unanswered.

1.3 Scope of the Thesis

This thesis deals with both the topics introduced above, analysis and verification, applied to the correctness problem of reactive programs. One assumption is that *model checking* is used to establish the validity of properties over programs; this is an a-posteriori, model-based, automatic method for property verification and was introduced by Queille and Sifakis in [QS82] and independently by Clarke and Emerson in [CE81]. An alternative approach, often called model checking as well, based on inclusion between automata over infinite words, was developed by Vardi and Wolper ([VW86]) and by Kurshan ([Kur90], also see [Kur94]). A closely related method is the tableau-based approach, see [LP85, PZ86]. Model checking has been quite successful in the verification of finite-state programs like protocols and controllers. Some reports may be found in the "Applications" sessions of the CAV conferences ([Sif89], [CK90], [LS91], [vBP92], [Cou93], [Dil94], [Wol95]). Model checking a program (in the narrow sense) involves two distinct phases, depicted in Figure 1.1a. First, a given notation[2] P of the program has to be "unfolded" into a model $\mathcal{C}$ — this is called *(model) construction*. This is formalised by a mapping $\mathcal{I}$ called *interpretation*. Second, the property φ of interest has to be checked over this model: $\mathcal{C} \models \varphi$.

It is no surprise that in applying this method, the abovementioned state-explosion problem, visualised in Figure 1.1b, forms the limiting factor. Being oriented on verification of reactive properties, the sort of programs that model checking is applied to often consist of several concurrent processes whose execution speeds are loosely related at most. As a result, the set of possible behaviours of the program as a whole contains a sequence for every possible interleaving of actions of the individual components. It is not hard to see that the number of such sequences may grow exponentially in the number of concurrent processes. In a similar fashion the presence of *data values* contributes to the problem. Every extra bit of memory that a program may access and every extra bit of width on a data channel potentially doubles the size of the state space. However, for many properties that one is interested in, the di erences

[2] We use the term *program notation* or *program text* (*program* for short) for a representation (of an algorithm) written in some programming language. The word *specification* is reserved for correctness criteria, while *model* and *denotation* will, from now on, be used for a mathematical structure representing the behaviour of a program.

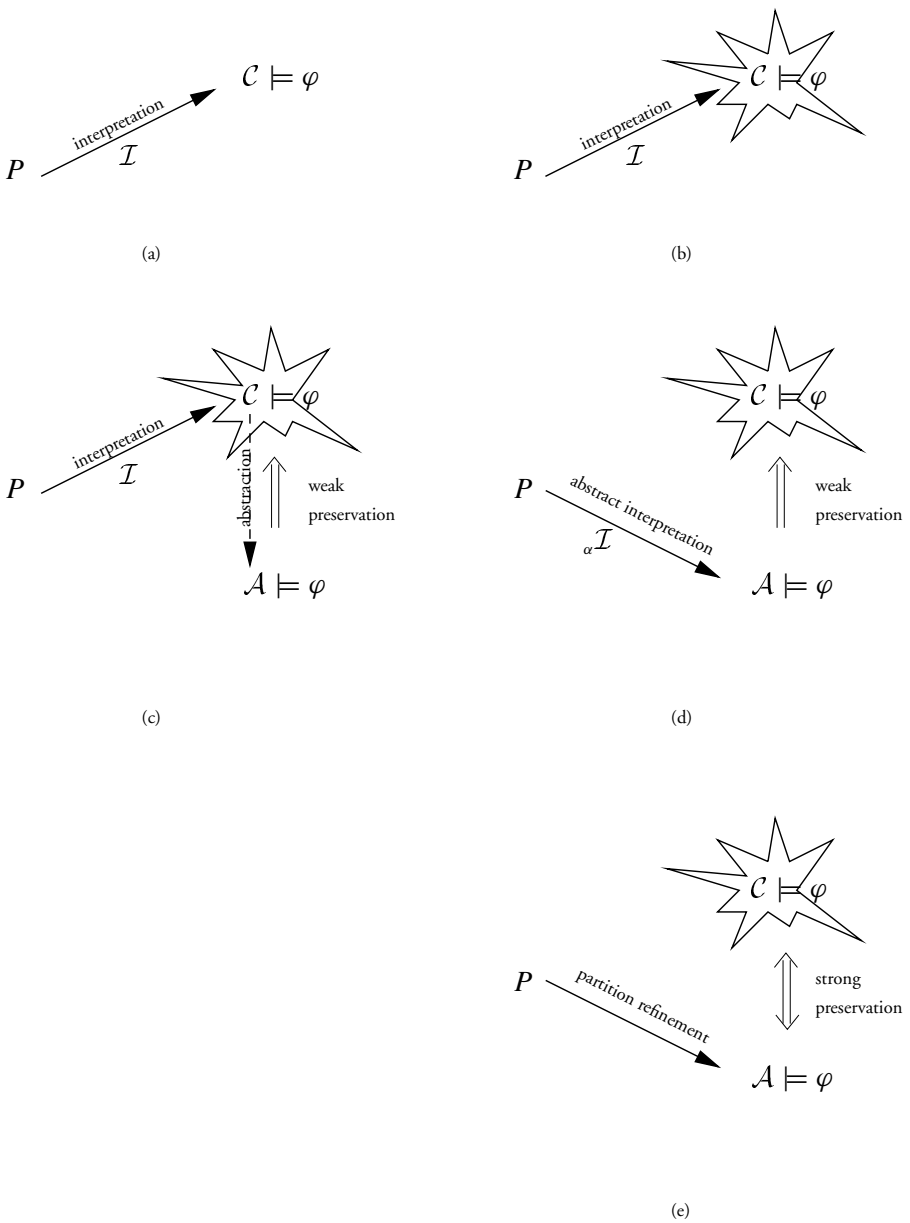


Figure 1.1: Model checking and abstraction.

between various interleavings or between the precise values sent over a channel are immaterial. Such aspects may then be abstracted in such a way that the size of the model is drastically reduced, but the property is still safely verified. This is illustrated in Figure 1.1c. Ideally, the model should be reduced to an abstraction $\mathcal{A}$ on which an e cient check can be performed, but under the condition that satisfaction of φ over $\mathcal{A}$ implies satisfaction over $\mathcal{C}$. Note that we do not require that negative results carry over as well: if φ is not satisfied over $\mathcal{A}$, then this does not imply that this is also the case for $\mathcal{C}$. It may equally well be that too much information was abstracted away from the concrete model. We call this implication in one direction *weak preservation* (of property φ under the abstraction).

Preferably, the full model $\mathcal{C}$ is not constructed at all during this — such an intermediate phase would still be the bottleneck. Rather, we would like to have a mapping from programs directly to the abstract model $\mathcal{A}$: an "*abstract interpretation function*" ${}_{\alpha}\mathcal{I}$, see Figure 1.1d.

Precisely these two aspects, the weak preservation of properties and the construction of abstract models directly from the program notation, are central in the theory of Abstract Interpretation. The first part of this thesis, to wit, (most of) Chapter 3 and Chapter 4, deals with these aspects. Chapter 3 presents a general theory of abstraction and preservation. In particular, it o ers a systematic overview of the theory of Abstract Interpretation (Sections 3.2 and 3.3.1). In Chapter 4, Abstract Interpretation is extended to the analysis of reactive properties. In the first part of that chapter, a notion of an abstract model for reactive programs is introduced that preserves any property expressed in the specification logic CTL*. Then, it is shown how such abstract models can be constructed by abstract interpretation of the operations in a simple programming language, which is introduced for that purpose.

Weak preservation is compatible with program *analysis*: any property that can be inferred from the abstract model is guaranteed to hold in the concrete model as well. However, the absence of a property does not give information. If our goal is the *verification* of properties, then we would like the converse direction of the implication in Figure 1.1c to hold as well. This is called *strong preservation* (of property φ under the abstraction), see Figure 1.1e. Strong preservation forms the main theme of the second part of this thesis. Section 3.2.2 indicates the changes to our starting points if we move from weak to strong preservation. A di erent paradigm, *partition refinement*, is then introduced in Chapters 5 and 6. Figure 1.2 gives a pictorial roadmap of the chapter structure.

Origin of the chapters Although part of the material contained in this thesis has been published in the form of articles, it has been restructured and extended. Chap-

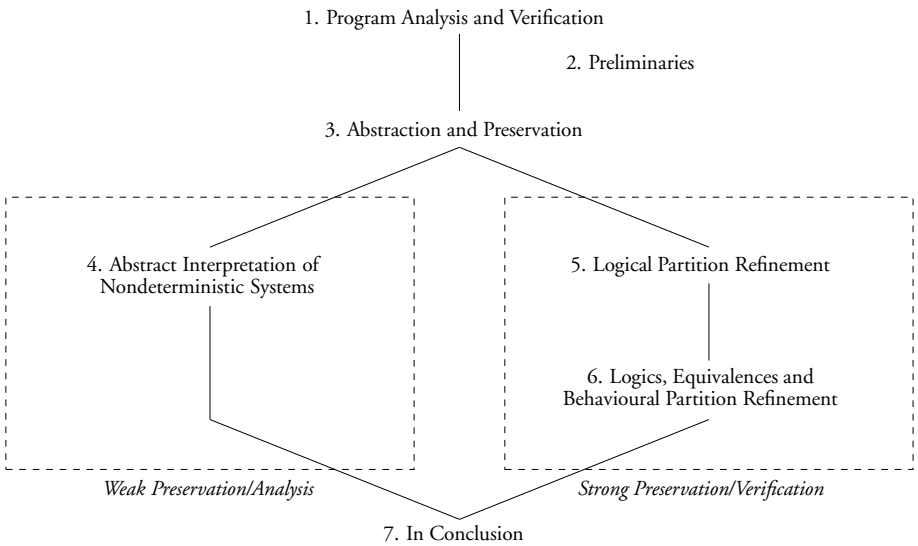


Figure 1.2: Structure of the chapters.

ter 4, with the exception of Section 4.8, is based on [DGG94]. Chapter 5 is a revision of [DGG93a] and contains some material from [DGD$^+$94] as well, while another part of [DGD$^+$94] is worked out in Section 4.8. Chapter 6 covers more recent research that has not yet been published. Chapter 3 summarises a series of unfinished notes that attempt to set up a systematic introduction to Abstract Interpretation.

All general preliminary material has been collected in Chapter 2 to which explicit references are provided throughout.

1.4 Related Work

References will be given at the end of each chapter. Here, we indicate a few main lines of research relating to program analysis and verification, and point out some work that is closest to the subject of this thesis.

1.4.1 Program analysis and Abstract Interpretation

Data-flow analysis has traditionally been aimed at the optimisation of programs. Typical applications include dead-code detection, type checking and inference, performance analysis, and partial evaluation. Furthermore, in the field of program correctness it has been used, for example, to prove termination and discover inductive

invariants. Literature on these subjects can be found through the bibliographies provided in [CC77] and [AH87]. [CC77] is the seminal paper that introduced Abstract Interpretation as a unifying framework for data-flow analyses. Since then, it has become particularly popular in the fields of functional and logic programming, the main reasons being (a) the uniform structure and mathematically simple semantics of programming languages in these areas, and (b) the urgent need to optimise compilers due to the relatively low performance of implementations of such languages. A main issue in functional programming seems to be *strictness analysis*, while in logic programming properties of logical variables, like groundness, sharing and freeness are intensively studied. Besides that, also various applications relating to program correctness, like debugging and type checking, have been formulated as abstract interpretations. See [AH87], [CC92a], and [JN95] for references.

Recently, ideas from Abstract Interpretation have been applied to correctness of reactive programs in a number of studies. [CGL92][3] shows how to construct abstract models that weakly preserve universal[4] safety and liveness[5] properties expressed in (a fragment of) the temporal logic CTL*. Preservation of full CTL* is shown in the context of strong preservation only. The preservation results are based on the properties of homomorphic functions, which have a long history in property preservation; see also Section 4.9. [BBLS92] generalises some of these results by presenting preservation results in the setting of simulations, which may be considered a generalisation of homomorphisms, and the μ-calculus, which exceeds CTL* in expressivity. Weak-preservation results are presented for the universal and existential fragments of the μ-calculus, while for the full μ-calculus only a strong-preservation result is given. The topic of constructing abstract models, which is briefly illustrated in that paper, is worked out further in the journal version, [LGS$^+$95], where it is also shown how the abstraction of a parallel system can be constructed compositionally from the abstractions of the individual components. In [Loi94], this theory is not only worked out in full detail, the implementation of a tool based on it is described and analysed too. The theory about preservation between, and the construction of, transition systems presented in Chapter 4 of this thesis is based on [DGG94], presenting material that was developed independently from [BBLS92, Loi94]. That paper focusses on the definition of a notion of abstraction of transition systems that preserves properties from full CTL*. The relation between a concrete and abstract systems is defined in terms of a *Galois insertion*, in a way that is less general than the approach based on simulation relations, but more general than homomorphic functions. The ad-

[3] A journal version appears as [CGL94].

[4] A property is called *universal* i it expresses that something holds along all possible executions of a program.

[5] A liveness property expresses that something happens eventually.

vantage of this Galois-insertion approach is that notions of precision and optimality can be captured, allowing for a better distinction between the quality of abstractions and of construction methods. Another advantage is that weak preservation of both universal and existential properties can be combined within a single notion of abstraction, without necessarily having strong preservation, which would limit the maximal reduction possible. [DGG93a] and [DGD$^+$94] investigate approaches to the construction of strongly preserving abstractions for fragments of CTL*, the former from a more theoretical and the latter from a more practical point of view. Chapter 5 of this thesis is based on [DGG93a], while part of the material in [DGD$^+$94] is worked out at the end of Chapter 4.

[Kel95] develops preservation results for the μ-calculus in the context of symbolic model checking (see Section 1.4.2 below and 2.4.3). The interpretation of a μ-calculus formula, which is a set of states, is approximated from below and above. By combining these dual approximate interpretations, using one for the $\Box$-operator and the other for the $\Diamond$, weak preservation of arbitrary μ-calculus formulae is obtained. This technique is similar to the mixed abstractions of [DGG94] that will be presented in Chapter 4. Kelb presents an application of these techniques to the verification of StateChart programs. Part of that work was carried out jointly; see [KDG95]. An extensive discussion of the results will be given in Section 4.9.2. Another interesting point in [Kel95] is the generalisation of preservation results to stutter-insensitive specifications, by forbidding an explicit next-step operator.

[CIY95] also develops preservation results for CTL* which are similar to ours. However, the focus of that paper is on the optimality of abstractions. As the notion of optimality defined there is a refined version of the one used in this thesis, the obtained results improve on ours. Another di erence is that the construction of abstract models is not considered in [CIY95].

Application of Abstract Interpretation to verify properties of CCS is described in [DFFGI95]. A recent paper, [KDG95], reports on progress in the practical application of Abstract Interpretation techniques to compositionally construct abstractions preserving the full μ-calculus. A more extensive overview and comparison can be found in Section 4.9.

1.4.2 Program verification

For proof-based methods, introductions as well as bibliographies may be found in [AO91], [Fra92] or [MP92, MP95] for example. The latter books are also recommended for an overview of temporal logic. Model checking was introduced independently in [QS82] and [CE81]. It has led to a large stream of both practically and theoretically oriented articles. Recently, much research has concentrated on tackling the state-explosion problem, see the two subsections below. In connection

with the automata-theoretic approach to model checking ([VW86, Kur90, Kur94]), we should mention the recent developments of [BG93, BVW94], which extend the method to branching-time temporal logics.

In *process algebraic theories* ([Mil80, Hoa85, BW90]), the specification of a program is usually expressed by another (simpler) program that it has to be "behaviourally equivalent" to. Thus, this is closer to the *full* verification of programs.

State abstraction in model checking

The distinction between *data* and *control* suggests a natural abstraction: in order to verify control properties, the actual values of data may sometimes be ignored. Clearly, this is only possible if the program is *data-independent*, i.e. the data values do not a ect the course of the computation. An example is a simple protocol whose only task is to transport messages from sender to receiver, without performing any error checking etc. [Wol86] describes such abstractions. Our approach treats data and control in a uniform way and hence has a more general applicability. Other general frameworks for abstraction in the context of model checking, besides those already mentioned above, are presented in [Sif82, Sif83] (containing much pioneering research), [Kur90] ("homomorphic reductions"), [Bur92] (in the context of trace theory, mainly in a real-time setting) and [Lon93] (combined with compositional techniques).

In Chapters 5 and 6 of this thesis, we study partition refinement algorithms, which reduce the size of a transition system by constructing the quotient under some equivalence relation. An overview of other research in this field is given at the end of those chapters.

Other state-space reduction techniques

Many possible solutions to the state-explosion problem have been suggested, resulting in a number of research directions. Roughly, the approaches can be divided into those that are aimed at e cient *representations* of the full state space, and those which reduce the state space by abstracting from certain aspects.

A very popular approach in the first category is *symbolic model checking*, in which the transition graph is represented by a reduced, ordered binary decision diagram (see [Bry86]). This is a tree-like structure that is used to, often compactly, represent a set of bitstrings; BDDs have gained popularity in the field of verification lately — see e.g. [DJS95, McM93]. That this approach may lead to spectacular results is evidenced by papers like [BCM$^+$92, MS92b, CGH$^+$93]. Although symbolic model checking has actually found its way into industry on an extensive scale, it is not a panacea. At best, it pushes forward the block that is formed by the state explosion — which indeed

holds for all reduction/abstraction methods. For an overview see [CGL93].

On-the-fly (or: *on-line*) techniques minimise the memory demands of model checkers by only expanding those parts of the program that are needed to verify the given property. Being exposed in e.g. [JJ89, BFH90, FM91], these techniques have indeed been used before in many verifiers developed by G. Holzmann; see e.g. [Hol84, Hol91].

An approach from the second category is formed by the *partial-order* techniques, which abstract from di erent interleavings of actions originating from concurrent components of a program. Originally developed for verification of linear-time temporal logic properties ([PL90, Val90, GW91, JK90, KP92, God96]), this technique has recently been extended for branching-time properties in [GKPP95].

Chapter 2

Preliminaries

This chapter introduces the mathematical machinery and reviews some specific notions from computing science that will be used in the rest of this thesis. The reader may choose to skip (parts of) it and only return here if necessary. In order to facilitate this, we provide references to this chapter in the chapters to come.

2.1 Relations and Functions

Let A and B be sets. A relation R over $A \times B$ is defined in the usual way as a set of pairs; A is called the domain and B the range of R. If $A = B$, then R is sometimes called a relation *on* A. In the rest of this section, $R \subseteq A \times B$. We often write $R(a, b)$ for $(a, b) \in R$, or also sometimes aRb, e.g. in $a \to b$ when $\to$ is a transition relation. When R is functional and $R(a, b)$, then $R(a)$ denotes b. R^{-1} is the inverse of R; for relations like $\leq$, $\subseteq$, etc. the inverses will be written as $\geq$, $\supseteq$, etc. The composition[1] (product) of relations $R \subseteq A \times B$ and $S \subseteq B \times C$ is denoted RS and defined as $\{(a, c) \mid \exists_{b \in B}\ R(a, b) \wedge S(b, c)\}$. If R and S are functions, then we sometimes write $S \circ R$ for RS. Totality on the domain and totality on the range are defined as usual; (plain) totality means totality on the domain (also known as *seriality*). Thus, surjectivity of a function means totality on its range. $A \to B$ is the set of all total functions from A to B. Functions are extended to sets pointwise, unless explicitly stated otherwise.

2.1.0.1 D *The pre-image function* $pre_R : \mathcal{P}(B) \to \mathcal{P}(A)$ *and post-image function* $post_R : \mathcal{P}(A) \to \mathcal{P}(B)$ *are defined as follows.*

$$pre_R(Y) = \{x \in A \mid \exists_{y \in Y}\ R(x, y)\}$$

$$post_R(X) = \{y \in B \mid \exists_{x \in X}\ R(x, y)\}$$

The dual functions $\widetilde{pre}_R : \mathcal{P}(B) \to \mathcal{P}(A)$ *and* $\widetilde{post}_R : \mathcal{P}(A) \to \mathcal{P}(B)$ *are defined by* $\widetilde{pre}_R(Y) = A \setminus pre_R(B \setminus Y)$ *and* $\widetilde{post}_R(X) = B \setminus post_R(A \setminus X)$. *The functions*[2] $pre^\bullet_R : B \to \mathcal{P}(A)$ *and* $post^\bullet_R : A \to \mathcal{P}(B)$ *are defined by* $pre^\bullet_R(y) = pre_R(\{y\})$ *and* $post^\bullet_R(x) = post_R(\{x\})$.
The relations $R^{\exists\exists}$, $R^{\forall\exists} \subseteq \mathcal{P}(A) \times \mathcal{P}(B)$ *are defined as follows.*

$$R^{\exists\exists} = \{(X, Y) \mid \exists_{x \in X}\ \exists_{y \in Y}\ R(x, y)\}$$

$$R^{\forall\exists} = \{(X, Y) \mid \forall_{x \in X}\ \exists_{y \in Y}\ R(x, y)\}$$

2.1.0.2 D *R is a* difunctional *i* $RR^{-1}R \subseteq R$.

Since we always have $R \subseteq RR^{-1}R$, R is a difunctional i $RR^{-1}R = R$.

2.1.0.3 D *R is* image-finite *i for every* $a \in A$, $post^\bullet_R(a)$ *has finite cardinality.*

[1] Other authors sometimes denote this by $R; S$.
[2] We need these functions in the next chapter.

2.2 Lattice and Fixpoint Theory

2.2.1 Orderings

Let S be a set and $\sqsubseteq$ a relation over $S \times S$. $\sqsubseteq$ is a *pre-order(ing)* i it is reflexive and transitive, and a *partial order(ing)* i it is anti-symmetric in addition. For a partial ordering $\sqsubseteq$, the pair $(S, \sqsubseteq)$ is called a *partially ordered set*, or *poset* for short; it is sometimes denoted S when the ordering is clear. Let $(S, \sqsubseteq)$ be a poset, $s, s' \in S$ and $T \subseteq S$. When $s \sqsubseteq s'$ we say that[3] s is *below* s' and that s' is *above* s. $s \sqsubset s'$ abbreviates $s \sqsubseteq s' \wedge s \neq s'$. s and s' are *comparable* i $s \sqsubseteq s'$ or $s' \sqsubseteq s$. T is a *chain* i any two elements of T are comparable. s (not necessarily in T) is a *lower bound* for T i s is below all elements of T; it is an *upper bound* for T i it is above all elements of T. A lower bound for T is the *greatest lower bound* (glb) for T i it is above any lower bound for T. An upper bound for T is the *least upper bound* (lub) for T i it is below any upper bound for T; it is denoted $\bigsqcup T$ if it exists. $t \in T$ is a *minimal* element of T i no other element of T is below t; it is a *maximal* element of T i no other element of T is above t. $\min(T)$ and $\max(T)$ denote the sets of minima and maxima of T respectively. $t \in T$ is the *least* (or *bottom*) element of T i it is below all elements in T; it is the *greatest* (or *top*) element of T i it is above all elements in T. $\bot_T$ and $\top_T$ denote the bottom and top of T respectively. Note that if T has a least element, $\min(T)$ is a singleton $\{t\}$. We write $\min(T) = t$ in such cases. Similarly for the greatest element.

T is *downwards-closed* i for any $t \in T$, whenever some $s \in S$ is below t then $s \in T$. T is *upwards-closed* i for any $t \in T$, whenever some $s \in S$ is above t then $s \in T$. T is a *principal ideal* i it is downwards-closed and the greatest element of T exists. T is a *principal filter* i it is upwards-closed and the least element of T exists. T is *downward-chain-limited* i for every chain $U \subseteq T$, T contains a lower bound for U. T is *upward-chain-limited* i for every chain $U \subseteq T$, T contains an upper bound for U.

2.2.1.1 L *If $R \subseteq A \times B$ is total[4] and image-finite, then for every $X \subseteq A$ there exists a $\subseteq$-minimal $Y \subseteq B$ such that $R^{\forall\exists}(X, Y)$.*

P Let $X \subseteq A$. Take C to be the collection of sets Z such that, for every $x \in X$, $post^\bullet_R(x)$ is not contained in Z. For a chain $Z_0 \subseteq Z_1 \subseteq Z_2 \cdots$ of sets in C, let Z^* be the union of the Z_i. Z^* is an upper bound of the chain of sets. We prove that Z^* is in C. If Z^* is not in C, then there is some $x \in X$ such that $post^\bullet_R(x)$ is contained in Z^*. Since $post^\bullet_R(x)$ is finite, $post^\bullet_R(x)$ must be contained in some Z_i. Contradiction.

[3] In order to avoid confusion we sometimes say s is R-below s' and s' is R-above s to denote sRs'. Similarly: R-minimal, R-maximal, R-downward, $\max_R$, etc.

[4] I.e. for every $x \in A$ there exists $y \in B$ such that $R(x, y)$.

We can now apply Zorn's Lemma[5] to conclude that C has a maximal element. Its complement is a minimal set Y as required. □

A poset $(S, \sqsubseteq)$ is a *complete partial order (cpo)* i every chain has a lub in S. Because $\emptyset$ is a chain, it follows that every cpo has a bottom element. It is a *complete lattice* i every subset has a lub in S. Note that in the latter case also every subset $T \subseteq S$ has a glb in S, namely the lub of the lower bounds of T. The lub and glb of $\emptyset$ constitute the bottom and top elements resp. of a lattice.

2.2.2 Functions over posets

Let $(P, \sqsubseteq)$ and $(Q, \preceq)$ be posets. $R \subseteq P \times Q$ is *monotonic* i $p \sqsubseteq p'$, $R(p, q)$ and $R(p', q')$ imply $q \preceq q'$; it is *pre-monotonic* i $p \sqsubseteq p'$, $R(p, q)$ and $R(p', q')$ imply $q \not\succ q'$. If R is functional, it is an *embedding* i $p \sqsubseteq p' \Leftrightarrow R(p) \preceq R(p')$. $R \subseteq P \times P$ is *reductive* i $R(p, p')$ implies $p' \sqsubseteq p$ and *pre-reductive* i $R(p, p')$ implies $p' \not\sqsupset p$. It is *extensive* i $R(p, p')$ implies $p' \sqsupseteq p$ and *pre-extensive* i $R(p, p')$ implies $p' \not\sqsubset p$.

Assume that $(P, \sqsubseteq)$ and $(Q, \preceq)$ are cpos and let $\bigsqcup$ and $\bigvee$ denote the respective lubs. $f : P \to Q$ is *continuous* i it is monotonic and for every non-empty chain $P' \subseteq P$, $f(\bigsqcup P') = \bigvee f(P')$. Note that if Q is a complete lattice, the requirement of monotonicity in this definition is redundant. A more general notion is distributivity: f *distributes over finite lubs* i for every finite $P' \subseteq P$ such that $\bigsqcup P'$ exists, $\bigvee f(P')$ exists and equals $f(\bigsqcup P')$. f *distributes over (arbitrary) lubs* i these conditions hold for every $P' \subseteq P$. Similar definitions can be given for distributivity over glbs. Distributivity over arbitrary lubs is also called *additivity*.

For a poset $(P, \sqsubseteq)$ and a function $f : P \to P$, $x \in P$ is a *fixed point (fixpoint)* of f i $f(x) = x$. Every monotonic function f on a cpo has a *least* fixpoint $\mathrm{lfp}(f)$, satisfying the following properties.

$$f(\mathrm{lfp}(f)) = \mathrm{lfp}(f) \tag{2.1}$$

$$\text{for every } y \in P : f(y) \sqsubseteq y \;\Rightarrow\; \mathrm{lfp}(f) \sqsubseteq y \tag{2.2}$$

Property 2.1 just says that $\mathrm{lfp}(f)$ is a fixpoint of f. Property 2.2 can be proven using the "Fundamental Lemma of Bourbaki" ([Bou50]): see e.g. Exercise 14 in [DP90]. Note that 2.2 implies that $\mathrm{lfp}(f)$ is below any fixpoint of f. Hence 2.1 and 2.2 together imply that $\mathrm{lfp}(f)$ is the least fixpoint of f and thus can be taken as an alternative definition.

[5] Zorn's Lemma, when specialised to this case, states that for any non-empty collection C of sets, if every increasing chain of sets in C has an upper bound in C, then C has a maximal element.

The following well-known[6] result provides a characterisation of $\mathrm{lfp}(f)$ for continuous f that is often more useful in practice than the above definitions. It states that least fixpoints can be computed by repeated function application. f^i denotes the i-fold application of f.

2.2.2.1 L *Let $(P, \sqsubseteq)$ be a cpo and $f : P \to P$ a continuous function. Then* $\mathrm{lfp}(f) = \bigsqcup\{f^i(\bot_P) \mid i \in \mathbb{N}\}$.

P From $\bot \sqsubseteq f(\bot)$ and monotonicity of f it follows by induction on i that $f^i(\bot) \sqsubseteq f^{i+1}(\bot)$ for every $i \geq 0$. So $\{f^i(\bot) \mid i \geq 0\}$ is a chain. Because f is continuous, we therefore have that $f(\bigsqcup\{f^i(\bot) \mid i \geq 0\}) = \bigsqcup f(\{f^i(\bot) \mid i \geq 0\}) = \bigsqcup\{f^i(\bot) \mid i \geq 1\}$. The latter term is equal to $\bigsqcup\{f^i(\bot) \mid i \geq 0\}$ and hence this is a fixpoint of f. Furthermore, for any fixpoint x of f, it follows with induction on i that $f^i(\bot) \sqsubseteq x$ for every $i \geq 0$; hence $\bigsqcup\{f^i(\bot) \mid i \geq 0\} \sqsubseteq x$. □

2.2.3 Galois connections

Let $(P, \sqsubseteq)$ and $(Q, \preceq)$ be posets.

2.2.3.1 D (f, g) *is a* Galois connection[7] *from P to Q i $f : P \to Q$, $g : Q \to P$, and for all $x \in P$ and $y \in Q$, $f(x) \preceq y \Leftrightarrow x \sqsubseteq g(y)$. f is called the* lower adjoint *(of g) and g the upper adjoint (of f).*

Galois connections have many properties, of which we mention a few. The interested reader is referred to [Ore44, Pic52, MSS86, CC92a, ABH+92]. A Galois connection (f, g) can alternatively be defined by the following conditions:

$$f \text{ and } g \text{ are monotonic} \tag{2.3}$$

$$f \circ g \text{ is reductive} \tag{2.4}$$

$$g \circ f \text{ is extensive} \tag{2.5}$$

Such f and g are each other's *semi-inverses*: $f \circ g \circ f = f$ and $g \circ f \circ g = g$. f uniquely determines its upper adjoint, if it exists. The latter is the case i f distributes over lubs and the set $\{x \mid f(x) \preceq y\}$ has a lub for every $y \in Q$; this lub is then $g(y)$. g uniquely determines its lower adjoint, if it exists. The latter is the case i g distributes over glbs and the set $\{y \mid x \sqsubseteq g(y)\}$ has a glb for every $x \in P$;

[6] The investigations reported in [LNS82] suggest that this lemma is a folk theorem in the sense of [Har80].

[7] In fact, Galois connections as defined here are semi-dual with respect to the original definition of [Ore44]: the definition given there requires f and g to be *antitone* (f is antitone if $x \sqsubseteq y \Rightarrow f(x) \succeq f(y)$) and both $f \circ g$ and $g \circ f$ to be extensive. An equivalent definition is obtained by requiring that $f(x) \succeq y \Leftrightarrow x \sqsubseteq g(y)$.

this glb is then $f(x)$. It follows that if P and Q are complete lattices, a function is a lower adjoint i it distributes over lubs and an upper adjoint i it distributes over glbs.

If in addition $f \circ g$ is the identity function, then (f, g) is called a *Galois insertion from P to Q*. We refer to Lemma 3.2.1.10 (page 43) for more details.

2.3 Temporal Logic

One of the ingredients of a "formal method" (see Section 1.1) is a formal language for expressing properties of programs. In particular, we want to be able to express typical properties of reactive systems — also called *reactive properties*. To this purpose, we use *temporal logic*. Examples are LTL (*linear temporal logic*, see [Pnu77]), CTL (*computation tree logic*, see [CES86]), CTL* ([EH86, EL87]), HML (*Henessy-Milner Logic*, see [HM85]) and L_μ (*the μ-calculus*, see [Koz83]). Such logics are all able, to some extent, to express both *universal* and *existential* properties (properties that have to hold along all executions of a program and properties that have to hold along some execution respectively), as well as *safety* ("nothing bad may happen") and *liveness* properties ("something good has to happen").

In this thesis, we use[8] CTL*, which combines a relatively high expressive power with good readability of formulae. Universal and existential properties are expressed through *path quantifiers* $\forall$ and $\exists$ that quantify over (infinite) execution sequences. The *temporal operators* X, U and V express properties of a single execution sequence. The *Next* operator X is used to express that something holds in the next state, while U, the *Until operator*, expresses that one property holds up to the point where another property holds. The latter operator can also be used to express safety or liveness alone (see below). V is called the *Release* operator and is the dual of the Until (i.e. $\mathsf{V}(\psi_1, \psi_2) \equiv \neg\mathsf{U}(\neg\psi_1, \neg\psi_2)$, see Section 2.4.1).

Given is a set Prop of *propositions*. We choose to define CTL* in its positive normal form, i.e. negations only appear in front of propositions. This facilitates the definition of universal and existential CTL*. The set of *literals* is defined by $\text{Lit} = \text{Prop} \cup \{\neg p \mid p \in \text{Prop}\}$.

2.3.0.1 D State formulae φ *and* path formulae ψ *are inductively defined by the following grammar, where* $p \in \text{Lit}$.

$$\textit{state formulae:} \quad \varphi \;:=\; p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \forall\psi \mid \exists\psi\,.$$

[8] In fact, there exist many versions of CTL*, exposing small but unnegligible di erences in syntax and semantics. In Section 2.4.1, where the interpretation of the formulae is defined, we briefly discuss this point.

path formulae: $\psi := \varphi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \mathsf{U}(\psi, \psi) \mid \mathsf{V}(\psi, \psi)$.

The logic CTL* *is defined as the set of state formulae.*

For $\varphi \in$ CTL**, the formula* $\neg\varphi$ *is considered to be an abbreviation of the equivalent* CTL* *formula in negation-normal form, which is obtained by applying the following "rewrite rules"*[9]*:*

$$\begin{array}{ll} \neg(\varphi_1 \wedge \varphi_2) \to \neg\varphi_1 \vee \neg\varphi_2 & \neg\mathsf{X}\psi \to \mathsf{X}\neg\psi \\ \neg(\varphi_1 \vee \varphi_2) \to \neg\varphi_1 \wedge \neg\varphi_2 & \neg\mathsf{U}(\psi_1, \psi_2) \to \mathsf{V}(\neg\psi_1, \neg\psi_2) \\ \neg\forall\psi \to \exists\neg\psi & \neg\mathsf{V}(\psi_1, \psi_2) \to \mathsf{U}(\neg\psi_1, \neg\psi_2) \\ \neg\exists\psi \to \forall\neg\psi & \end{array}$$

The abbreviations true*,* false *and* $\to$ *can then be defined as usual. For a path formula* ψ*,* $\mathsf{F}\psi$ *and* $\mathsf{G}\psi$ *abbreviate* $\mathsf{U}(\mathit{true}, \psi)$ *and* $\mathsf{V}(\mathit{false}, \psi)$ *respectively.*

$\forall$CTL* *and* $\exists$CTL* *(*universal *and* existential CTL**) are subsets of* CTL* *in which the only allowed path quantifiers are* $\forall$ *and* $\exists$ *respectively.*

Note that every state formula is also a path formula. For the meaning of formulae, we refer to the next section.

2.3.0.2 E *Let* $p, q \in$ Prop*. The formula* $\forall\mathsf{F}p \to \forall\mathsf{G}q$ *is not in* $\forall$CTL**, because (assuming that* false *abbreviates* $p \wedge \neg p$*) it is an abbreviation of the* CTL* *formula* $\exists\mathsf{V}(p \wedge \neg p, \neg p) \vee \forall\mathsf{V}(p \wedge \neg p, \neg q)$*, which contains an* $\exists$*.*

CTL* can alternatively be defined by allowing the negation $\neg$ to be applied to any formula. It is then not necessary to include the dual operators $\vee$, V and $\forall$ of $\wedge$, U and $\exists$ resp. In places where the universal and existential fragments do not play a role (e.g. Chapter 6), we use this alternative definition.

The *approximants* $\mathsf{U}_i(\psi_1, \psi_2)$ and $\mathsf{V}_i(\psi_1, \psi_2)$ correspond to the "unfolding" of the U and V operators (see Lemma 2.4.1.3 in the next section). They are abbreviations that are defined as follows:

$$\begin{array}{ll} \mathsf{U}_0(\psi_1, \psi_2) = \mathit{false} & \mathsf{U}_{i+1}(\psi_1, \psi_2) = \psi_2 \vee (\psi_1 \wedge \mathsf{X}\mathsf{U}_i(\psi_1, \psi_2)) \\ \mathsf{V}_0(\psi_1, \psi_2) = \mathit{true} & \mathsf{V}_{i+1}(\psi_1, \psi_2) = \psi_2 \wedge (\psi_1 \vee \mathsf{X}\mathsf{U}_i(\psi_1, \psi_2)) \end{array}$$

The fragment CTL is obtained by requiring that a quantifier is always directly followed by a single temporal operator:

[9] Note that X is its own dual; see the interpretation defined in Section 2.4.1.

2.3.0.3 D *The logic* CTL *is the set of (state) formulae φ defined inductively by the following grammar, where $p \in$ Lit.*

$$\varphi \;:=\; p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \forall \mathsf{X}\varphi \mid \exists \mathsf{X}\varphi \mid$$
$$\forall \mathsf{U}(\varphi, \varphi) \mid \exists \mathsf{U}(\varphi, \varphi) \mid \forall \mathsf{V}(\varphi, \varphi) \mid \exists \mathsf{V}(\varphi, \varphi).$$

For $\varphi \in$ CTL*, the formula $\neg\varphi$ is considered to be an abbreviation of the equivalent* CTL *formula in negation-normal form (obtained in the usual way). The abbreviations true, false and $\rightarrow$ can then be defined as usual. For a* CTL *formula φ, $\forall \mathsf{F}\varphi$ and $\forall \mathsf{G}\varphi$ abbreviate $\forall \mathsf{U}(\mathit{true}, \varphi)$ and $\forall \mathsf{V}(\mathit{false}, \varphi)$ respectively, and similarly for $\exists$.*

$\forall$CTL *and* $\exists$CTL *(*universal *and* existential CTL*) are subsets of* CTL *in which the only allowed path quantifiers are $\forall$ and $\exists$ respectively.*

Note that in CTL quantifiers and temporal operators appear only "glued together", so that these combinations can be viewed as single operators. There are several alternative definitions for CTL that all yield equivalent logics. One possibility is to allow arbitrary negation; in that case not all operators of Definition 2.3.0.3 are needed. For example, $\forall\mathsf{X}$, $\forall\mathsf{V}$ and $\exists\mathsf{V}$ may be dropped. Negation also allows to express $\forall\mathsf{U}$ in terms of $\exists\mathsf{U}$ and $\exists\mathsf{G}$ (see Lemma 2.4.1.2), suggesting another way to obtain a functionally complete set of operators. A third possibility is to allow the negation symbol to occur in between a quantifier and a temporal operator; see e.g. [DNV90b].

The *level* of $\varphi \in \forall$CTL is the maximal number of nested $\forall\mathsf{X}$ operators in φ if there are no $\forall\mathsf{U}$ and $\forall\mathsf{V}$ operators in φ, and ω otherwise, i.e., assuming $\omega + 1 = \omega$:

$$\mathit{level}(p) = 0 \quad \text{for } p \in \text{Lit} \tag{2.6}$$
$$\mathit{level}(\varphi_1 \wedge \varphi_2) = \mathit{level}(\varphi_1 \vee \varphi_2) = \max(\mathit{level}(\varphi_1), \mathit{level}(\varphi_2)) \tag{2.7}$$
$$\mathit{level}(\forall\mathsf{X}\varphi) \;=\; 1 + \mathit{level}(\varphi) \tag{2.8}$$
$$\mathit{level}(\forall\mathsf{U}(\varphi_1, \varphi_2)) = \mathit{level}(\forall\mathsf{V}(\varphi_1, \varphi_2)) = \omega \tag{2.9}$$

Similar definitions may be given for $\exists$CTL, however, we do not need these.

2.3.0.4 E *Let $p, q \in$ Lit. The level of p and of $p \wedge q$ is 0. The level of $\forall\mathsf{X}(p \wedge \forall\mathsf{X}q)$ is 2. The level of $\forall\mathsf{F}\varphi$ is ω for any φ, the level of $\forall\mathsf{X}(p \wedge \forall\mathsf{U}(p, q))$ is ω, and the level of $\forall\mathsf{X}(p \wedge \forall\mathsf{U}_5(p, q))$ is 6. In general, for $i \geq 2$, the level of $\forall\mathsf{U}_i(\varphi_1, \varphi_2)$ is $i - 1 + \max\{\mathit{level}(\varphi_1), \mathit{level}(\varphi_2), 1\}$.*

2.3.1 Nextless fragments

Another way to restrict the logics defined above is by dropping the Next operator. See Section 6.3.1 for a discussion of the motivations for doing this.

2.3.1.1 D *The logic* CTL*(U) *contains all* CTL* *formulae that do not contain* X*, i.e. it is the set of state formulae defined by the following grammar, where* $p \in \mathrm{Lit}$.

$$\textit{state formulae:} \quad \varphi := p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \forall\psi \mid \exists\psi.$$

$$\textit{path formulae:} \quad \psi := \varphi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathsf{U}(\psi, \psi) \mid \mathsf{V}(\psi, \psi).$$

Similarly, CTL(U) *consists of all* CTL *formulae that do not contain* X.

2.3.1.2 N *Throughout this thesis, we write* CTL° *whenever both* CTL* *and* CTL *may be read. A similar convention is used for all fragments of these logics.*

2.4 Transition Systems

A *transition system (over* Σ*)* is a pair $\mathcal{T} = (\Sigma, \mathrm{R})$ consisting of a set Σ of *states* and a *transition relation* $\mathrm{R} \subseteq \Sigma \times \Sigma$. A *path* in $\mathcal{T}$ is an *infinite* sequence $\pi = s_0 s_1 \cdots$ of states such that for every $i \in \mathbb{N}$, $\mathrm{R}(s_i, s_{i+1})$; we say that π *starts in* s_0. $\pi(i)$ denotes s_i. A subsequence of π is called a *part* of π or also a *block* sometimes; it is denoted π_I where I is a connected subset of $\mathbb{N}$ (i.e. for every two numbers in I, all numbers in between are also in I). A part of π that starts in s_0 is called a *prefix* of π and a part that continues infinitely is called a *su x* of π. Also finite sequences $\pi = s_0 s_1 \cdots s_k$ of states such that for every $0 \leq i < k$, $\mathrm{R}(s_i, s_{i+1})$, but for which there is no outgoing transition from s_k (i.e. they are not parts of paths), are called prefixes. In a more operational context, we sometimes use the term *computation* instead of prefix. The notation π^n denotes the su x of π that begins at s_n. A *partitioning* of π is a (finite or infinite) sequence $\pi_{\{0,\ldots,i_0\}}, \pi_{\{i_0+1,\ldots,i_1\}}, \ldots$ of parts of π; when it is finite, the last part must be a su x of π. The *length* of a prefix or part $\hat{s}$ of a path, denoted *length*($\hat{s}$), is the number of states on it; note that the last state of $\hat{s}$ is $\hat{s}(\textit{length}(\hat{s}) - 1)$ if it exists. For $s \in \Sigma$, a $(\mathcal{T}, s)$*-path* (or s*-path* when $\mathcal{T}$ is clear from the context) is a path in $\mathcal{T}$ that starts in s; similarly for prefixes. *paths*$(\mathcal{T}, s)$ (or simply *paths*(s)) denotes the set of all s-paths while *prefixes*$(\mathcal{T}, s)$ (*prefixes*(s)) contains all their prefixes. $\mathcal{T}$ is called *image-finite* i R is (Definition 2.1.0.3).

A transition system may have various attributes. Often, a subset $\mathrm{I} \subseteq \Sigma$ is designated to represent the *initial states*. In this case, a state $s \in \Sigma$ is *reachable* i , for some $t \in \mathrm{I}$, there exists a t-path containing s. Also, a transition system may come with an *interpretation function* $\|\cdot\|_{\mathrm{Lit}} : \mathrm{Lit} \to \mathcal{P}(\Sigma)$ that specifies the interpretation of literals (see Section 2.3, page 18) over states. Intuitively, $\|p\|_{\mathrm{Lit}}$ is the set of states where p holds. We assume that $\|p\|_{\mathrm{Lit}} \cap \|\neg p\|_{\mathrm{Lit}} = \emptyset$ for every proposition $p \in \mathrm{Prop}$. We

do *not* impose that $\|p\|_{\mathrm{Lit}} \cup \|\neg p\|_{\mathrm{Lit}} = \Sigma$; this means that, intuitively, the valuation of a literal may be undetermined or unknown in some state[10]. Alternatively, the valuation of literals in states may be given by a *labelling function* $\pounds : \Sigma \to \mathcal{P}(\mathrm{Lit})$ specifying the literals that hold in a state. We use $\|\cdot\|_{\mathrm{Lit}}$ and $\pounds$ intermixedly with the implicit understanding that $\pounds(s) = \{q \in \mathrm{Lit} \mid s \in \|q\|_{\mathrm{Lit}}\}$. Hence, we have for all $s \in \Sigma$ and all $p \in \mathrm{Prop}$, $\{p, \neg p\} \not\subseteq \pounds(s)$.

2.4.0.1 D *Let $\pounds : \Sigma \to \mathcal{P}(\mathrm{Lit})$ be a labelling function and $s, t \in \Sigma$. We define $s \equiv^0 t \Leftrightarrow \pounds(s) = \pounds(t)$.*

A transition system with initial states and an interpretation function is also called *Kripke structure* ([Kri63]). $\mathcal{KS}(\Sigma)$ ($\mathcal{KS}$ when Σ is implicit) denotes the set of all Kripke structures over Σ.

2.4.1 Interpretation of CTL*

CTL* formulae are interpreted over Kripke structures. Satisfaction of formulae is defined inductively as follows.

2.4.1.1 D *Let $\mathcal{K} = (\Sigma, \mathrm{R}, \mathrm{I}, \|\cdot\|_{\mathrm{Lit}})$ be a Kripke structure. Let $p \in \mathrm{Lit}$, $\varphi, \varphi_1, \varphi_2$ be state formulae and ψ, ψ_1, ψ_2 be path formulae, $s \in \Sigma$ and π a path in $\mathcal{K}$.*

1. $(\mathcal{K}, s) \models p$ *i* $s \in \|p\|_{\mathrm{Lit}}$.
2. $(\mathcal{K}, s) \models \varphi_1 \wedge \varphi_2$ *i* $(\mathcal{K}, s) \models \varphi_1$ *and* $(\mathcal{K}, s) \models \varphi_2$, $(\mathcal{K}, s) \models \varphi_1 \vee \varphi_2$ *i* $(\mathcal{K}, s) \models \varphi_1$ *or* $(\mathcal{K}, s) \models \varphi_2$.
3. $(\mathcal{K}, \pi) \models \varphi$, *where* $\pi = s_0 s_1 \cdots$ *i* $(\mathcal{K}, s_0) \models \varphi$.
4. $(\mathcal{K}, \pi) \models \psi_1 \wedge \psi_2$ *i* $(\mathcal{K}, \pi) \models \psi_1$ *and* $(\mathcal{K}, \pi) \models \psi_2$, $(\mathcal{K}, \pi) \models \psi_1 \vee \psi_2$ *i* $(\mathcal{K}, \pi) \models \psi_1$ *or* $(\mathcal{K}, \pi) \models \psi_2$.
5. (a) $(\mathcal{K}, \pi) \models \mathsf{X}\psi$ *i* $(\mathcal{K}, \pi^1) \models \psi$.
   (b) $(\mathcal{K}, \pi) \models \mathsf{U}(\psi_1, \psi_2)$ *i there exists* $n \in \mathbb{N}$ *such that* $(\mathcal{K}, \pi^n) \models \psi_2$ *and for all* $i < n$, $(\mathcal{K}, \pi^i) \models \psi_1$.
   (c) $(\mathcal{K}, \pi) \models \mathsf{V}(\psi_1, \psi_2)$ *i for all* $n \in \mathbb{N}$, *if* $(\mathcal{K}, \pi^i) \not\models \psi_1$ *for all* $i < n$, *then* $(\mathcal{K}, \pi^n) \models \psi_2$.

[10] The reason that we allow undetermined truth values is that in Chapter 4, we define *abstract states* as (representations of) sets of concrete states. Concrete states in which some proposition p holds may occur together in such a set with states in which p does not hold, so that the valuation of p in the abstract state is "unknown".

6. $(\mathcal{K}, s) \models \forall\psi$ *i for every* s*-path* π, $(\mathcal{K}, \pi) \models \psi$, $(\mathcal{K}, s) \models \exists\psi$ *i there exists an* s*-path* π *such that* $(\mathcal{K}, \pi) \models \psi$.

For a set S *of states or paths, the notation* $(\mathcal{K}, S) \models \varphi$ *abbreviates* $\forall_{s\in S}\ (\mathcal{K}, s) \models \varphi$. *When* $\mathcal{K}$ *is clear from the context, we write* $s \models \varphi$ *for* $(\mathcal{K}, s) \models \varphi$, *and similar for* $(\mathcal{K}, S) \models \varphi$. $\mathcal{K} \models \varphi$ *now is defined as* $(\mathcal{K}, \mathrm{I}) \models \varphi$.

Equivalence of formulae is defined as follows. Let φ_1 *and* φ_2 *be state or path formulae.* $(\mathcal{K}, s) \models \varphi_1 \equiv \varphi_2$ *(or* $s \models \varphi_1 \equiv \varphi_2$ *for short) i* $s \models \varphi_1 \Leftrightarrow s \models \varphi_2$. $\mathcal{K} \models \varphi_1 \equiv \varphi_2$ *i* $\mathcal{K} \models \varphi_1 \Leftrightarrow \mathcal{K} \models \varphi_2$. $\models \varphi_1 \equiv \varphi_2$ *(or* $\varphi_1 \equiv \varphi_2$ *for short) i for every* $\mathcal{K} \in \mathcal{KS}$, $\mathcal{K} \models \varphi_1 \equiv \varphi_2$.

CTL* formulae can be used to express a variety of properties of transition systems. Apart from state based properties expressed by formulae built from literals and boolean connectors, properties of paths may be expressed through the Next, Until and Release operators. For example, $\pi \models \mathsf{XU}(p, q)$ expresses that along path π, from the next state on, p will hold in all states until we eventually get to a state where q holds. $\pi \models \mathsf{XXX}p$ says that p holds in the fourth state of π. $\pi \models \mathsf{F}p$ and $\pi \models \mathsf{G}p$ state that p will hold eventually resp. always along π. Note that, strictly speaking, path formulae are not in CTL*: they have to be preceded by $\forall$ or $\exists$, resulting in state formulae. It can easily be seen that a sequence of path quantifiers directly following each other may be replaced by the last (rightmost) quantifier of the sequence under equivalence of the formula. E.g. $\exists\forall\forall\exists\forall\mathsf{U}(p, q) \equiv \forall\mathsf{U}(p, q)$. If literal r characterises reset states, then $s \models \forall\mathsf{G}\exists\mathsf{F}r$ means that along every possible execution path from s, in every state there is a possible continuation that will eventually reach a reset state.

Note that CTL* formulae containing path quantifiers express properties about the infinite computations of a system only, because the quantifiers are interpreted over paths (which are infinite computations by definition). Although in the original definition of CTL* in [EH86] also finite computations are taken into consideration, both [EL87] and the recent overview article [Eme90] revise the original definition by quantifying over ("full-")paths only. A similar definition is given in [CGL94]. As a consequence, this version of CTL* cannot express properties of computations that end in a "deadlock" state (i.e. a state from which no transitions are possible) and hence some care has to be taken in (intuitively) interpreting the formulae. For example, $s \models \forall\mathsf{G}p$ expresses that p is true in all states that lie on any path starting from s, which, by the definition of path as an infinite sequence, is not the same as the statement that p is true in all states that can be reached from s by taking finitely many transitions. Another approach, taken in [DNV90b] for example, interprets formulae along *maximal* prefixes, i.e. prefixes that cannot be extended because either they are infinite, or their last state has no outgoing transitions. In yet other articles

(e.g. [EC82, CES86]), transition relations are required to be *total*: every state must have an outgoing transition. This requirement avoids the type of misinterpretations discussed above, but imposes the obligation to ensure that every system being defined has a total transition relation. This may be cumbersome. In Section 4.7 of this thesis, a solution is proposed that relies on a "trick" to perform deadlock checking. Once the system has been checked to be free from deadlock, CTL* path formulae may be interpreted along maximal computations, which are then always infinite.

V is the dual of U ($\mathsf{V}(\psi_1, \psi_2)$ is semantically equivalent to the negation of $\mathsf{U}(\neg\psi_1, \neg\psi_2)$) and has the intuitive meaning of "release": ψ_2 must be true as long as ψ_1 is false, and only if ψ_1 becomes true, ψ_2 may become false afterwards. It has been added as the dual of U to compensate for the fact that formulae like $\neg\mathsf{U}(\psi_1, \psi_2)$ are not well-formed. For the same reason both $\wedge$ and $\vee$ are primitive in the logic.

We stress that the V operator is not the same as the *Weak-Until* or *Unless* operator W that is defined by $\mathsf{W}(\psi_1, \psi_2) = \mathsf{G}\psi_1 \vee \mathsf{U}(\psi_1, \psi_2)$. The relation between the two is given by $\models \mathsf{V}(\psi_1, \psi_2) \equiv \mathsf{W}(\psi_2, \psi_1 \wedge \psi_2)$ (or $\models \mathsf{V}(\psi_1, \psi_2) \equiv \mathsf{W}(\psi_2 \wedge \mathsf{X}\psi_2, \psi_1)$) and $\models \mathsf{W}(\psi_1, \psi_2) \equiv \mathsf{V}(\mathsf{X}\psi_2, \psi_1)$; these equivalences can easily be verified via the semantics of the operators, as defined in Definition 2.4.1.1 above.

2.4.1.2 L *Let $\psi_1, \psi_2, \psi \in \mathrm{CTL}^*$. We have $\models \forall\mathsf{U}(\psi_1, \psi_2) \equiv \neg\exists\mathsf{U}(\neg\psi_2, \neg\psi_1 \wedge \neg\psi_2) \wedge \neg\exists\mathsf{G}\neg\psi_2$. Reversely, $\exists\mathsf{G}\psi$ is equivalent to $\neg\forall\mathsf{U}(\mathit{true}, \neg\psi)$.*

P $\forall\mathsf{U}(\psi_1, \psi_2) \equiv \neg\neg\forall\mathsf{U}(\psi_1, \psi_2) \equiv \neg\exists\mathsf{V}(\neg\psi_1, \neg\psi_2) \equiv \neg\exists\mathsf{W}(\neg\psi_2, \neg\psi_1 \wedge \neg\psi_2) \equiv \neg\exists(\mathsf{G}\neg\psi_2 \vee \mathsf{U}(\neg\psi_2, \neg\psi_1 \wedge \neg\psi_2)) \equiv \neg\exists\mathsf{G}\neg\psi_2 \wedge \neg\exists\mathsf{U}(\neg\psi_2, \neg\psi_1 \wedge \neg\psi_2)$ (the third step uses the relation between V and W given above and the fourth step applies the definition of W.) The second fact is obvious. □

The following lemma shows that the approximants $\mathsf{U}_k(\psi_1, \psi_2)$ and $\mathsf{V}_k(\psi_1, \psi_2)$ correspond to the "depth-k versions" of the U and V operators:

2.4.1.3 L *Let $\mathcal{K}$, π, ψ_1 and ψ_1 be as in Definition 2.4.1.1, and $k \in \mathbb{N}$.*

1. *$(\mathcal{K}, \pi) \models \mathsf{U}_k(\psi_1, \psi_2)$ i there exists $0 \leq n < k$ such that $(\mathcal{K}, \pi^n) \models \psi_2$ and for all $i < n$, $(\mathcal{K}, \pi^i) \models \psi_1$.*

2. *$(\mathcal{K}, \pi) \models \mathsf{V}_k(\psi_1, \psi_2)$ i for all $0 \leq n < k$, if $(\mathcal{K}, \pi^i) \not\models \psi_1$ for all $i < n$, then $(\mathcal{K}, \pi^n) \models \psi_2$.*

P By induction on k. □

Distinguishing power

A logic L interpreted over the states of a transition system $\mathcal{T} = (\Sigma, \mathrm{R})$ induces an equivalence relation $\equiv_{\mathrm{L}} \subseteq \Sigma \times \Sigma$ defined by $s \equiv_{\mathrm{L}} t$ i $\forall_{\varphi \in \mathrm{L}}\, s \models \varphi \Leftrightarrow t \models \varphi$. $\equiv_{\mathrm{L}}$ is called the *logical equivalence induced by* L or L*-equivalence* for short. It captures the *distinguishing power* of L: states are equivalent i no L-property distinguishes them. This notion is not to be confused with the *expressive power* of L, which measures the capability to characterise *sets* of states by a single formula. The expressive power is normally used to compare logics, as follows: $\mathrm{L}_1 \leq \mathrm{L}_2$ i $\forall_{\varphi_1 \in \mathrm{L}_1}\, \exists_{\varphi_2 \in \mathrm{L}_2}\, \varphi_1 \equiv \varphi_2$ (see Definition 2.4.1.1).

The relation between distinguishing and expressive power is given by the following lemma.

2.4.1.4 L $\quad \mathrm{L}_1 \leq \mathrm{L}_2 \;\Rightarrow\; \equiv_{\mathrm{L}_1} \supseteq \equiv_{\mathrm{L}_2}$

P Assume that (1) $\mathrm{L}_1 \leq \mathrm{L}_2$ and $s \equiv_{\mathrm{L}_2} t$, i.e. (2) $\forall_{\varphi_2 \in \mathrm{L}_2}\, s \models \varphi_2 \;\Leftrightarrow\; t \models \varphi_2$. We have to show that then $s \equiv_{\mathrm{L}_1} t$, i.e $\forall_{\varphi_1 \in \mathrm{L}_1}\, s \models \varphi_1 \;\Leftrightarrow\; t \models \varphi_1$. Let $\varphi_1 \in \mathrm{L}_1$ and assume (3) $s \models \varphi_1$. By 1, we can choose $\varphi_2 \in \mathrm{L}_2$ such that (4) $\varphi_1 \equiv \varphi_2$. From 3 and 4 we have $s \models \varphi_2$, from which by 2, $t \models \varphi_2$. With 4 again, we get $t \models \varphi_1$. The other direction is symmetric. □

The other direction of the implication does not hold. The following small example[11] clarifies this. Consider the sets $\mathrm{L}_1 = \mathcal{P}(\mathbb{N})$ and $\mathrm{L}_2 = \{\mathbb{N} \setminus \{k\} \mid k \in \mathbb{N}\}$ of propositions. As states, over which the propositions are interpreted[12], take the natural numbers, defining for $i \in \mathbb{N}$ and $\varphi \in \mathrm{L}_1, \mathrm{L}_2$: $i \models \varphi$ i $i \in \varphi$. Clearly $\mathrm{L}_2 \lneq \mathrm{L}_1$. However, it is also easy to show that any two numbers that can be distinguished by L_1, can also be distinguished by L_2, implying that $\equiv_{\mathrm{L}_1} = \equiv_{\mathrm{L}_2}$.

2.4.2 Behavioural preorders and equivalences

In this subsection, $\mathcal{T}_1 = (\Sigma_1, \mathrm{R}_1)$ and $\mathcal{T}_2 = (\Sigma_2, \mathrm{R}_2)$ are transition systems with labelling functions $\pounds_1 : \Sigma_1 \to \mathcal{P}(\mathrm{Lit})$ and $\pounds_2 : \Sigma_1 \to \mathcal{P}(\mathrm{Lit})$ respectively.

2.4.2.1 D *Let $\sigma \subseteq \Sigma_1 \times \Sigma_2$ be a relation such that for every $s \in \Sigma_1$ and $t \in \Sigma_2$, $\sigma(s,t)$ implies:*

1. $\pounds_1(s) = \pounds_2(t)$.

[11] Thanks to Ruurd Kuiper.

[12] The sets of propositions induce the temporal logic CTL* and its fragments, which are the only logics that we have defined. Formally speaking, we have to define Kripke structures (and not just individual states) over which such formulae are interpreted. This may be done by turning each state i into the Kripke structure with state set (and initial-state set) $\{i\}$, transition relation $\emptyset$, and interpretation function as suggested by the definition of $\models$ in this example.

2. *For every s' such that $\mathrm{R}_1(s, s')$, there exists t' such that $\mathrm{R}_2(t, t')$ and $\sigma(s', t')$.*

Then σ is called a simulation *(from $\mathcal{T}_1$ to $\mathcal{T}_2$, or from[13] Σ_1 to Σ_2), and R_1 σ-*simulates R_1. *The largest simulation is denoted sim. s* simulates[14] *t i $sim(s, t)$. If sets I_1 and I_2 of initial states for $\mathcal{T}_1$ and $\mathcal{T}_2$ resp. are given, then a simulation σ is* proper *i $\sigma^{\forall\exists}(\mathrm{I}_1, \mathrm{I}_2)$. In this case we say that $\mathcal{T}_1$ (σ-)*simulates *$\mathcal{T}_2$.*

Sometimes we consider relations $\sigma \subseteq \Sigma_1 \times \Sigma_2$ for which 2 holds, but not 1. Such relations are called non-consistent *or* pseudo-*simulations. In that case we say that* R_1 *σ-*pseudo-simulates *R_1, etc.*

Because the empty relation is always a simulation and simulations are closed under union, the largest simulation *sim* always exists. It is easy to see that $sim(s, t)$ i there exists a simulation σ such that $\sigma(s, t)$.

An alternative, algebraic definition of simulations is suggested by the following lemma (recall Definition 2.4.0.1 of $\equiv^0$).

2.4.2.2 L *Let $\mathcal{T}_1$, $\mathcal{T}_2$, $\pounds_1$ and $\pounds_2$ be as above. $\sigma \subseteq \Sigma_1 \times \Sigma_2$ is a simulation i $\sigma \subseteq \equiv^0$ and $\sigma^{-1}\mathrm{R}_1 \subseteq \mathrm{R}_2\sigma^{-1}$.*

The following definition and lemma suggest an alternative definition of simulation for image-finite (page 21) transition systems.

2.4.2.3 D *The sequence $\{sim^n\}_{n\in\mathbb{N}}$ of relations on $\Sigma_1 \times \Sigma_2$ is defined as follows.*

1. *$sim^0(s, t)$ i $\pounds(s) = \pounds(t)$.*
2. *$sim^{n+1}(s, t)$ i for every s' such that $\mathrm{R}_1(s, s')$ there exists t' such that $\mathrm{R}_2(t, t')$ and $sim^n(s', t')$.*

2.4.2.4 L *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are image-finite, then for all $s \in \Sigma_1$, $t \in \Sigma_2$, we have $sim(s, t)$ i $\forall_{n\in\mathbb{N}}$ $sim^n(s, t)$.*

P Like, e.g., the proof of a similar property of bisimulation (see Definition 2.4.2.7 below) in [HM85]. □

2.4.2.5 P *If $sim^{k+1} = sim^k$ for some $k \geq 0$, then for all $j \geq 0$, $sim^{k+j} = sim^k$.*

[13] When confusion about the domain and range of a simulation may occur, we attach the type as a subscript.

[14] The intuition is that t can "mimic" everything that s can do. From this point of view, the terminology "s simulates t", introduced in [Mil71], is awkward.

P Define the function $\mathcal{F} : \mathcal{P}(\Sigma_1 \times \Sigma_2) \to \mathcal{P}(\Sigma_1 \times \Sigma_2)$ by $\mathcal{F}(\sigma) = \{(s,t) \in \sigma \mid \forall_{s' \in \Sigma_1} [\mathrm{R}_1(s,s') \Rightarrow \exists_{t' \in \Sigma_2} [\mathrm{R}_2(t,t') \wedge (s',t') \in \sigma]]\}$. Clearly, (*) $sim^{n+1} = \mathcal{F}(sim^n)$ for every $n \geq 0$. Now assume that $sim^{k+1} = sim^k$ for some $k \geq 0$. By induction on j, we prove that for all $j \geq 0$, $sim^{k+j} = sim^k$. The base case is obvious. For the induction step we have: $sim^{k+j+1} \overset{(*)}{=} \mathcal{F}(sim^{k+j}) \overset{\text{i.h.}}{=} \mathcal{F}(sim^k) \overset{(*)}{=} sim^k$. □

Two common behavioural equivalences are derived from simulations.

2.4.2.6 D *The relation* $simeq \subseteq \Sigma_1 \times \Sigma_2$ *is defined by* $simeq(s,t) \Leftrightarrow sim_{\Sigma_1 \times \Sigma_2}(s,t) \wedge sim_{\Sigma_2 \times \Sigma_1}(t,s)$. *The relations* $simeq^n$ *are defined similarly in term of* sim^n.

An equivalence relation $\sigma \subseteq \Sigma_1 \times \Sigma_2$ *is called a* simulation equivalence *i* $\sigma \subseteq simeq$. *s and t are* simulation equivalent *i* $simeq(s,t)$.

Given sets I_1 *and* I_2 *of initial states for* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *resp., a simulation equivalence* σ *is* proper *i* $\sigma^{\forall\exists}(\mathrm{I}_1, \mathrm{I}_2)$ *and* $\sigma^{-1\,\forall\exists}(\mathrm{I}_2, \mathrm{I}_1)$. *In this case we say that* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *are simulation equivalent.*

If $\Sigma_1 = \Sigma_2$, then $simeq = sim \cap sim^{-1}$.

2.4.2.7 D *The relation* $\equiv_{\text{bis}} \subseteq \Sigma_1 \times \Sigma_2$ *is defined as the largest simulation* σ *such that* σ^{-1} *is a simulation as well.*

An equivalence relation $\sigma \subseteq \Sigma_1 \times \Sigma_2$ *is called a* bisimulation *i* $\sigma \subseteq \equiv_{\text{bis}}$. *s and t are* bisimilar *i* $s \equiv_{\text{bis}} t$.

Given sets I_1 *and* I_2 *of initial states for* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *resp., a bisimulation* σ *is* proper *i* $\sigma^{\forall\exists}(\mathrm{I}_1, \mathrm{I}_2)$ *and* $\sigma^{-1\,\forall\exists}(\mathrm{I}_2, \mathrm{I}_1)$. *In this case we say that* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *are bisimilar.*

We return to bisimulation in Chapter 6.

2.4.3 Model checking

Given a formula φ and a Kripke structure $\mathcal{K}$, model checking is the process of verifying whether $\mathcal{K} \models \varphi$ holds. In a narrow sense, the term usually applies to the satisfiability of a temporal-logic formula by a finite transition system, being checked automatically. Many model-checking algorithms exist for several fragments of CTL*, as well as for di erent formalisms like the μ-calculus ([Koz83]). A number of approaches can be distinguished. A division often made is between *automata-theoretic* and *graph-traversal* algorithms. In the first, the formula φ is transformed into a (finite) automaton over infinite words that accepts exactly those behaviours that satisfy $\neg\varphi$. This automaton is then "multiplied" with the transition system $\mathcal{K}$. The resulting automaton accepts the behaviours in the intersection of $\mathcal{K}$ and $\neg\varphi$. Thus, if (and only if) this intersection is empty, $\mathcal{K} \models \varphi$ holds. See e.g. [VW86, Kur94].

Related is the *tableau-based* appoach — see e.g. [LP85, PZ86]. In the graph-traversal approach, φ is checked over $\mathcal{K}$ in a more direct fashion. For example, the states of $\mathcal{K}$ are labelled with subformulae of φ in an iterative fashion, starting with propositions and labelling with longer formulae in every subsequent step. Such an algorithm is presented in [CES86]. Another approach is presented in [QS82], in which a fix-point algorithm is used to compute sets of states satisfying the subformulae of φ. It relies on the availability of the pre-image function pre_R (Definition 2.1.0.1) of the transition relation R, which may be viewed as an "implicit" or "symbolic" representation of the transition system. Hence, such algorithms are often termed *symbolic*. Recently, reduced ordered binary decision diagrams (BDDs, see [Bry86]) have been proposed as e cient representations of both sets of states and transition relations ([BCM$^+$92, CBM89]), leading to a renewed interest in this symbolic approach.

Traditionally, automata-theoretic algorithms are used for linear-time temporal logic, while the graph-traversal algorithms were devised in the context of CTL and the μ-calculus. Indeed, the first model-checking algorithm for CTL*, which subsumes both LTL and CTL, combines both techniques in one algorithm (see [EL87]). Recent work, [BG93, BVW94, Kup95], shows that e cient branching-time model checking can be captured in the automata-theoretic framework too, using alternating-tree automata.

The complexity of model checking increases with the expressivity of the temporal logic. More particularly, the time complexity of model checking CTL is linear in both the size of the transition system and of the formula ([CES86]), while for LTL ([LP85]) and CTL* it is linear in the size of the transition system but exponential in the size of the formula ([EL87]).

Although this thesis is about techniques that aim to extend of the applicability of model checking, the particular model-checking approach that is chosen and the details of the algorithms are immaterial. The interested reader is referred to the articles mentioned above. Furthermore, a number of overviews have recently appeared: [Eme90, Sti92, CGL93, Kur94].

Chapter 3

Abstraction and Preservation

Besides introducing abstraction and Abstract Interpretation, this chapter presents a systematic reconstruction of several frameworks for abstraction that occur in the literature. We formalise the notion of property preservation, and investigate how weak and strong preservation lead to di erent frameworks. New results are the characterisation of weaker frameworks in terms of weakenings of the notion of Galois connection, and a generalisation of the "power construction".

3.1 Introduction

This chapter introduces the notion of an abstraction theory, which is a framework to construct and justify abstractions used to analyse the properties of complex objects. Special attention will be paid to Abstract Interpretation, which is a rather well-developed and popular framework — or rather, a class of frameworks — for the so-called weak preservation of properties. As the presentation in the rest of this chapter is on a fairly abstract level, we shall briefly introduce the ideas of Abstract Interpretation on an intuitive level first in Section 3.1.1 below.

A systematic approach to motivate, classify and present Abstract Interpretation frameworks has long been lacking. Indeed, Abstract Interpretation has long been a rich pool of ideas and results of which the sources were based on two articles ([CC77, CC79]) whose accessibility in some points is reduced due to the density of information. The approach that we follow in this chapter has been influenced by the (successful) e orts of Marriott and Søndergaard to give an orderly account of the theory and some of its applications: [MS89b, Søn90], and especially [Mar93]. More recently, the Cousots' overview articles [CC92a] and [CC92b] take a similar systematic approach in their presentation.

Although many of the results concerning Abstract Interpretation in this chapter are well-known, we do have a number of contributions that are new, to the best of our knowledge. First, we take a more general approach to abstraction, which accounts for various forms of property preservation. In particular, the viewpoint taken in Section 3.2.2 paves the way to Chapters 5 and beyond, which adopt a di erent approach to the construction of abstract models than the "non-standard semantics" of Abstract Interpretation. Second, in Section 3.2.1 we present a number of characterisations of weaker frameworks in terms of conditions placed on the abstraction and concretisation relations, being generalisations of Galois connections. Furthermore, the section on power construction exposes and formalises a common "dilemma" in Abstract Interpretation, giving a number of theoretical results.

3.1.1 Abstract Interpretation

The term Abstract Interpretation was introduced in [CC77] and denotes a unified approach to program analysis (also called *data-flow*, *static* or *compile-time* analysis) by formalising an analysis as an approximate computation operating on descriptions of data. The following informal definition is given in that paper.

> *"A program denotes computations in some universe of objects. Abstract interpretation of programs consists in using that denotation to describe computations in another universe of abstract objects, so that the results of abstract execution give some informations on the actual computations."*

A simple and commonly used example of Abstract Interpretation is the use of the rule of signs to determine the sign of an arithmetic expression. In order to say whether $-1515 * 17$ is positive or negative, we do not have to perform the multipication on the "concrete" level of numbers and then look at the sign of the result, but instead we can first "abstract" the individual operands to their signs and then apply $\overline{*}$, the rule of signs for multiplication: $\mathsf{neg}\ \overline{*}\ \mathsf{pos} = \mathsf{neg}$. This rule of signs for the product enjoys the property that its result always correctly describes the result of any concrete multiplication on any operands that it abstracts. The relation between the descriptions $\{\mathsf{neg}, \mathsf{pos}\}$ and the integers $\mathbb{Z}$ is commonly specified by a *concretisation function* $\gamma : \{\mathsf{neg}, \mathsf{pos}\} \to \mathcal{P}(\mathbb{Z})$ with $\gamma(\mathsf{neg}) = \{x \in \mathbb{Z} \mid x \leq 0\}$ and $\gamma(\mathsf{pos}) = \{x \in \mathbb{Z} \mid x \geq 0\}$[1]. The correctness property for $\overline{*}$, often called *safety*, can then be formalised by requiring $\gamma(a\ \overline{*}\ b) \supseteq \gamma(a) * \gamma(b)$ for all $a, b \in \{\mathsf{neg}, \mathsf{pos}\}$, where we assume that multiplication is pointwise extended to sets. This extension to sets already indicates that the concrete operation being actually "mimicked" is not multiplication between numbers, but between sets of numbers. This is also expressed by γ that maps the descriptions neg and pos to *sets*. At first, however, it is not clear why we should formalise things at the level of sets: why not take, for example, a description relation $\rho \subseteq \mathbb{Z} \times \{\mathsf{neg}, \mathsf{pos}\}$ relating negative numbers (and zero) to neg and positive (and zero) to pos? Several arguments can be given (see Section 3.2.1), but an important one is that to *construct* a safe abstraction $\overline{*}$ over, e.g., the pair (neg, neg) (suppose we did not know what it should be), we have to take into account the "behaviour" of the concrete multiplication over *all concrete objects that* (neg, neg) *describes*. In a sense, we want to have the most general object that (neg, neg) describes — and to be able to formalise this notion of generality, we lift $\mathbb{Z}$ to its power set: "more general than" then becomes "a superset of". This construction corresponds to the adoption of a description relation $\check{\rho} \subseteq \mathcal{P}(\mathbb{Z}) \times \{\mathsf{neg}, \mathsf{pos}\}$: each *set* of non-positive (non-negative) numbers is described by neg (pos). Note that $\check{\rho}(C, a) \wedge C' \subseteq C$ implies $\check{\rho}(C', a)$, i.e. $pre^{\bullet}_{\check{\rho}}(a)$ is $\subseteq$-downward closed. γ then maps a description to the largest set described. This shift of attention from single elements to sets is indeed explicitly made in much of the Cousots' work, where it occurs through the lifting of the "standard semantics" to the "collecting semantics" — see e.g. [CC92b] or [CC77], where the term "static semantics" is used instead of collecting semantics.

Alternatively, we may specify the relation between concrete and abstract objects through an *abstraction function* $\alpha \in \mathcal{P}(\mathbb{Z}) \to \{\mathsf{neg}, \mathsf{pos}\}$. Clearly, we would like to have, e.g., $\alpha(\{-7, -2\}) = \mathsf{neg}$ and $\alpha(\{x \in \mathbb{Z} \mid x \geq 0\}) = \mathsf{pos}$. But what should, e.g., $\alpha(\{-3, 1\})$ be? And what about $\alpha(\{0\})$? As to the first question: considering another operation that might have to be abstracted, addition, shows that it would be

[1] Note that we choose to let neg describe the negative integers as well as 0, and similarly for pos.

useful if α also provides a description for sets containing both negative and positive numbers. Denoting the rule of signs for addition by $\overline{+}$, the result of $\mathsf{neg}\ \overline{+}\ \mathsf{pos}$ should intuitively be "don't know". This is usually denoted by ${}_\alpha\top$, the *top* element of the abstract domain, with $\gamma({}_\alpha\top) = \mathbb{Z}$ and $\alpha(S) = {}_\alpha\top$ for any set $S \subseteq \mathbb{Z}$ that contains both negative and positive numbers. It is "larger" than the other descriptions in the sense that $\gamma({}_\alpha\top) \supseteq \gamma(\mathsf{neg})$ and $\gamma({}_\alpha\top) \supseteq \gamma(\mathsf{pos})$. We may formalise this and define an *approximation* or *precision ordering* $\preceq$ on the abstract domain by $a \preceq b \Leftrightarrow \gamma(a) \subseteq \gamma(b)$.

In dealing with $\{0\}$, we may resort to di erent solutions. First, we may just weaken the framework and allow α to be a relation, linking $\{0\}$ both to neg and to pos. Second, we may *choose* $\alpha(\{0\})$ to be one of neg and pos. A third solution consists in enriching the domain of descriptions with an element zero such that $\alpha(\{0\}) = \mathsf{zero}$. The abstraction function obtained by this last solution maps every non-empty set of numbers to its $\preceq$-least description. Symmetrically to γ giving the most general concrete element being described, α then returns the "most precise" description. It turns out that this renders (α, γ) a Galois connection — indeed even a Galois insertion, by the way $\preceq$ is defined (see Section 3.2.1) — from $(\mathcal{P}(\mathbb{Z}) \setminus \emptyset, \subseteq)$ to $(\{\mathsf{zero}, \mathsf{neg}, \mathsf{pos}, {}_\alpha\top\}, \preceq)$. Each solution has its pros and cons, which we will not discuss here. The interested reader is referred to [CC79], [Mar93] and [CC92b].

Another point is whether $\emptyset$ should have an α image. One may claim that this is not necessary because there is no need for a description of it. There are situations however where the empty set does fulfil an essential role in the concrete domain, e.g. to model the e ect of errors or non-termination. And even in absence of such reasons it may be useful from the point of view of symmetry between abstract and concrete domains to introduce an element ${}_\alpha\bot$ such that $\alpha(\emptyset) = {}_\alpha\bot$ and $\gamma({}_\alpha\bot) = \emptyset$. This turns $(\{{}_\alpha\bot, \mathsf{zero}, \mathsf{neg}, \mathsf{pos}, {}_\alpha\top\}, \preceq)$ into a lattice.

In the presence of an abstraction function, safety of some abstract operator $\overline{\otimes}$ with respect to the concrete operator $\otimes$ may be expressed in di erent ways, e.g. $\alpha(C)\ \overline{\otimes}\ \alpha(D) \succeq \alpha(C \otimes D)$ for all $C, D \subseteq \mathbb{Z}$, or $\gamma(\alpha(C)\ \overline{\otimes}\ \alpha(D)) \supseteq C \otimes D$, or $a\ \overline{\otimes}\ b \succeq \alpha(\gamma(a) \otimes \gamma(b))$ for all descriptions a, b. All formulations, including the one given earlier, are easily shown to be equivalent for any Galois connection (α, γ). The last formulation is probably the most helpful when *designing* abstract operators: for each a and b it specifies how "large" $a\ \overline{\otimes}\ b$ should be at least to be safe. A di erent aspect that then comes to mind is *optimality*: preferably, the result of $a\ \overline{\otimes}\ b$ should also be as precise as possible. Hence, $\overline{\otimes}$ should be defined by $a\ \overline{\otimes}\ b = \alpha(\gamma(a) \otimes \gamma(b))$.

In this example, arithmetic operations are interpreted over descriptions, or abstractions: they are "abstractly interpreted". The term abstract interpretation, written

in lower case, is used with this connotation in this thesis. The capitalised term Abstract Interpretation, on the other hand, denotes a particular theory (or: collection of theories), which will be presented in Section 3.2.1.

In the definition from [CC77] quoted above, as well as in the example, the main concern of Abstract Interpretation is the *construction* of descriptions of concrete objects by "mimicking" the e ect of concrete operations with suitable (safe) abstract operations defined over descriptions. An aspect that is also covered, but less explicitly, is the nature of *property preservation*: what does the description tell us about the object being described? In most treatments and applications of Abstract Interpretation, this point remains rather unexposed. A formalisation of property preservation would involve the definition of a language to express properties about concrete and abstract objects, and satisfaction relations to express when a property holds for an object. Then, the soundness of the abstraction with respect to the properties could be expressed by formally stating that whenever $\alpha(C) = a$ and a satisfies φ (in the above example, φ could denote, e.g. the property "is positive" or "is negative"), then C (or rather all $c \in C$) satisfies φ as well.

The main reason for this attitude in most works on Abstract Interpretation is that the properties being analysed are often of the same kind: they express invariance properties[2] about objects. Also, the type of preservation considered is always of the form "if φ holds for a, then it holds for C"; conversely, the absence of property φ for a does in general not justify the conclusion that it is false for C as well. One may see this as the very nature of abstraction theories — however, we show that this is not necessarily so. As argued in Section 1.1, when verification rather than analysis of programs is our goal, interest shifts towards preservation of truth as well as falsehood of properties (strong preservation).

Chapter 4 of this thesis shows how the framework of Abstract Interpretation may be extended to cover the preservation of other properties than invariance. It is demonstrated that for these cases, descriptions may also be constructed through abstract interpretation of operations in the program text. Then, in Chapters 5 and onwards, the focus is shifted towards strong preservation. There, it turns out that di erent techniques for the computation of abstractions may be useful. For this reason, we use the more general term *abstraction theory* to capture this case too.

In conclusion, we give the following working definition:

[2] An invariance property states that "nothing bad can happen" (cf. [Lam83]). More to the point, it does not say whether anything will happen at all.

An abstraction theory:

- Provides ways to relate concrete to abstract objects and, for each of these, specifies how properties are preserved.
- Provides ways to compute abstract objects in an e cient fashion.

3.1.2 Overview of the chapter

The rest of this chapter presents a formalisation of this notion. Section 3.2 concentrates on the first point, while Section 3.3 deals with the second.

This chapter is not only an introduction to the rest of the thesis, but also contains some results that are interesting in their own right. These are the outcome of an attempt to set up a general theory of abstraction that explains the choices made in the more specific Galois-insertion framework. Therefore, by their level of abstractness, certain parts may be less relevant to the understanding of the chapters to come. The reader whose main purpose is to understand the application of abstract-interpretation techniques to model checking, rather than to get involved in the background of Abstract Interpretation, may safely skip the part from the paragraph "Approximative description" (page 39) until Section 3.2.2 on strong preservation (page 49) at first. In Section 3.3, the theory about fixpoints, including Lemmata 3.3.1.1 and 3.3.1.2, has been included for completeness and will not be used in the remainder of the thesis.

3.2 Preservation Results

As explained above, one of the two pillars that an abstraction theory should be built on is concerned with preservation of properties between objects. In general, a preservation result relates some correspondence (expressed as a relation for example) between two objects to the fact that certain properties that hold for one object also hold in the other. A common example is found in Model Theory: if two structures are isomorphic, then they satisfy exactly the same first-order properties. In this example:

1. The correspondence between the structures is an equivalence relation (isomorphism).

2. The relation between the satisfaction of first-order formulae is bi-implication (a formula is satisfied in one structure i it is in the other).

3. The connection between the existence of the equivalence relation (point 1) and the satisfaction of the same first-order formulae (point 2), is a strict implication: in general, the result does not hold in the reverse direction.

This theme can be varied, giving rise to a hierarchy of preservation types. The dichotomy *weak* vs. *strong* preservation determines the structure of the following subsections (and in fact constitutes one of the major themes of this thesis). In Subsection 3.2.1, we consider weak preservation and give a general presentation of it that captures many frameworks that have been proposed in the literature. Subsection 3.2.2 focusses on various types of strong preservation and lists some consequences for the previously discussed frameworks. First, we provide some basic ingredients.

Description relations We assume sets C and A of concrete and abstract objects respectively and a relation $\rho \subseteq C \times A$ linking concrete objects to their descriptions. The fact that objects have properties is formalised by assuming a logic L in which properties are expressed, and satisfaction relations $\models \subseteq C \times L$ and $\models^{\alpha} \subseteq A \times L$ specifying when a property holds for a concrete resp. abstract object (logically speaking: when a concrete resp. abstract object is a model for a formula).[3]

As the goal is to be able to "reason abstractly" about any concrete object, we require that every concrete object in C has at least one description[4]:

$$\rho \text{ is total on C.} \tag{3.1}$$

Conversely, we do not want to have abstract objects around that describe nothing[5]. Hence we also require

$$\rho \text{ is total on A.} \tag{3.2}$$

A relation ρ that is total on its domain and range is called a *description relation (from C to A)*.

3.2.1 Frameworks for weak preservation

Safety An abstract object a that acts as a description of a concrete object c should be *safe*[6] with respect to its properties from L. This means that every L-property that

[3] We could have chosen a set-up with two logics, L_γ for concrete and L_α for abstract objects, together with a translation function $t : L_\gamma \to L_\alpha$, with the idea that to analyse a property $\varphi \in L_\gamma$ over some $c \in C$ ($c \models \varphi$), one analyses the translation $t(\varphi)$ over a description a of c ($a \models' t(\varphi)$). This set-up is equivalent to the current, with $a \models^{\alpha} \varphi$ defined by $a \models' t(\varphi)$.

[4] cf. condition 4.5 in [CC92b].

[5] There might be a use in having such objects around, though. It could be that A can easily be defined, but the restriction to the range of ρ is difficult or impossible. In such a case the requirement that ρ be total on A may be satisfied by extending ρ (to the "superfluous" elements) and possibly also C.

[6] This notion of safety should not be confused with safety as introduced in Section 1.2.for c

holds for a, should hold for c as well; in other words, a should give faithful information about c. We may choose the following condition, expressing that descriptions should be safe[7].

$$\forall_{c\in \mathrm{C}, a\in \mathrm{A}}\ [\rho(c,a)\ \Rightarrow\ \forall_{\varphi\in \mathrm{L}}\ [c \models \varphi\ \Leftarrow\ a \models^{\alpha} \varphi]] \tag{3.3}$$

This requirement is referred to as *(weak) preservation of* L *under* ρ; "weak" because the preservation only holds from abstract to concrete objects (the $\Leftarrow$ implication in the above formula).

From this starting point, we discuss a number of additional conditions that may be assumed on top of the basic ingredients given above. This results in an incremental reconstruction, "from scratch", of various frameworks for Abstract Interpretation that have been put forward in the literature, carefully motivating each of the design decisions taken. Along the way, new results and insights are given.

Approximative description relations

Approximation In order to study properties of some concrete object c we need to construct a description a of it. Requirement 3.1 then ensures that such an a exists, and by 3.3 we can learn properties of c by studying a. However, the fact that a describes c does not give a clue as to *how useful* the object a is in analysing properties of c. For example, it may be that a enjoys no properties from L (i.e. $\forall_{\varphi\in \mathrm{L}}\ a \not\models^{\alpha} \varphi$). So as to be able to compare descriptions, we assume an *approximation order* $\preceq\ \subseteq \mathrm{A}\times\mathrm{A}$. If $a \preceq a'$, we say that a is approximated by a', or that a is more precise than a'. This ordering should be such that

$$a \preceq a'\ \Rightarrow\ \forall_{\varphi\in \mathrm{L}}\ [a \models^{\alpha} \varphi\ \Leftarrow\ a' \models^{\alpha} \varphi] \tag{3.4}$$

Furthermore, it should be possible to use a less precise object as a description, hence[8]:

$$\rho(c,a) \wedge a \preceq a'\ \Rightarrow\ \rho(c,a') \tag{3.5}$$

A notion of approximation should be reflexive and transitive and hence $\preceq$ is a preorder. An example can be found in Section 4.4. The approximation relation $\preceq$ between abstract transition systems defined there is a preorder but not a partial order.

[7] A di erent choice would be to *identify* the description relation (ρ) with the safety relation ($\forall_{\varphi\in \mathrm{L}}\ [c \models \varphi\ \Leftarrow\ a \models^{\alpha} \varphi]$); see e.g. [CC79, CC92b]. All abstract objects that are safe for c are then captured in $post^{\bullet}_{\rho}(c)$. Distinguishing the notions of description and safety yields a more general setup, which allows to ignore some of the safe elements when defining which elements may be used as descriptions.

[8] cf. condition 4.19 in [CC92b].

Of course any preorder can be turned into a partial order by identifying equivalent elements, in the usual way. In the following, we assume that (A, $\preceq$) a poset.

An alternative approach to approximation that deviates from "standard" Abstract Interpretation theory would have been to postulate a *metric* on A, quantifying the precision of descriptions. Another choice is to have approximation relations $\preceq_c$ for each concrete object c, so that the precision of a description can be measured relative to the object it describes. We will not follow these directions any further.

In most theories of Abstract Interpretation, the concrete objects are partially ordered too, usually even in a complete lattice. The reasons for doing so may be very diverse; we list a few.

- Defining the abstract counterpart ${}_\alpha f$ of a concrete function (or relation) f for some $a \in$ A, the values $f(c)$ for all c decribed by a have to be taken into account. It is convenient if a "most general" c can be selected in such a way that mimicking the behaviour of f over c su ces to capture all other cs that are described. This motivates the introduction of a "generality order", such that there is a unique most general one among every non-empty set of concrete objects being described by some a.

  If no such ordering preexists, then a natural solution is to move to the power set lattice over the concrete objects, as was exemplified in the previous section. The subsection on power construction (page 44) elaborates on this point.

- Often, Abstract Interpretation is used to approximate *least fixpoints* of functions over C, expressing the semantics of a certain program, by computing least fixpoints of corresponding abstract functions over A. In order for fixpoints to exist, it su ces if C and A are complete partial orders and the functions are monotonic [Tar55].

  Indeed, Abstract Interpretation, in a narrow sense, is often understood to be the analysis of programs by approximating the (least) solution to a system of recursive equations representing the behaviour of a (repetitive or recursive) program, see for example [CC92c, Bou92, LV92].

- The third motivation is closely related to the second one. In computer science applications, the concrete objects are usually semantic models of programs. In most traditional approaches to semantics, models are partially ordered (sometimes for di erent reasons than the existence of fixpoints). In denotational semantics ([Sto77, Sch86]), so-called *domains*, which are partially ordered algebraic structures, are used to be able to correctly capture the behaviour of functional programming languages with self-application — and also Scott's

information systems ([Sco82]) come with a notion of "informativeness" that is a partial order. Approaches to devise models for concurrent/nondeterministic programs are often based on sets ordered by inclusion ([Dil89]), or, if one sticks to the denotational tradition, on power domains.

- The last reason for introducting a partial order on C that we mention is the *symmetry* between A and C that may be desirable. Such a symmetry is needed for example when abstractions are to be composed, allowing for a stepwise design of an analysis. Examples of such composed abstractions may be found in [Mer90], [CDY91] and also in Section 4.8.1 in this thesis.

In general, several orderings on the concrete objects may be introduced, one for each of these reasons. Because, in this section, our interest is in frameworks for abstract interpretation rather than in the structure of the (concrete and abstract) semantic domains, we henceforth assume a partial order $\sqsubseteq \subseteq \mathrm{C} \times \mathrm{C}$ to express the notion mentioned in the first point above. $c \sqsubseteq c'$ expresses that c' is more general than c (also: c is approximated by c', c is more precise than c'). It turns out that in many applications of abstraction theories, this order coincides with the orderings as meant under the other points (see also Section 3.3.1). In fact, the framework described in the seminal [CC77] does not distinguish between those orderings; [CC92b] does.

Symmetrically to 3.4, $\sqsubseteq$ should satisfy:

$$c \sqsubseteq c' \;\Rightarrow\; \forall_{\varphi \in \mathrm{L}}\, c \models \varphi \;\Leftarrow\; c' \models \varphi \tag{3.6}$$

and its relation to ρ is specified as follows.

$$\rho(c, a) \wedge c' \sqsubseteq c \;\Rightarrow\; \rho(c', a) \tag{3.7}$$

Thus, if $\rho(c, a)$ then a may be replaced by a less precise object (requirement 3.5) and c by a more precise object (requirement 3.7) without violating the description relation between them.

Minimal and maximal elements; optimality We have started from sets C and A, a language L interpreted over them, a total relation ρ between them and added approximation orderings $\preceq$ and $\sqsubseteq$ satisfying the requirements 3.4, 3.6, 3.5 and 3.7. Note that by the latter two, $post^{\bullet}_{\rho}$ is $\preceq$-upward-closed[9] and $pre^{\bullet}_{\rho}$ is $\sqsubseteq$-downward-closed (Definition 2.1.0.1 in Section 2.2).

[9]The properties "has a minimal element", "has a maximal element", "is $\preceq$-upward-closed" and "is $\preceq$-downward-closed" are extended pointwise to functions in this chapter. That is to say, "$post^{\bullet}_{\rho}$ has a minimal element" abbreviates "for all $c \in \mathrm{C}$, $post^{\bullet}_{\rho}(c)$ has a minimal element", etc.

A common issue in Abstract Interpretation is the existence of *maximally precise*, i.e. $\preceq$-minimal descriptions. When abstracting a concrete object, one usually looks for an $\preceq$-minimal description (indeed, it may be convenient if there exists a unique such description, i.e. an $\preceq$-least description).

3.2.1.1 D *Let $a \in$ A be a description of $c \in$ C, i.e. $\rho(c, a)$. Then a is* optimal[10] *for c i a is $\preceq$-minimal in $post_\rho(c)$, i.e. there is no a' such that $\rho(c, a')$ and $a' \prec a$.*

When performing a computation in the abstract domain, the outcome should preferably be an optimal description of a corresponding operation on the concrete side. Symmetrically, we are interested in *maximally general*, i.e. *minimally precise* concrete objects being described by a given description. Totality of ρ on C and A guarantees the existence of *some* description for any concrete object, and reversely, the existence of *something* being described by a given abstract object. However, optimal elements may not exist, e.g. because all the possible candidates are arranged in infinite chains. We exclude this by requiring:

$$post^\bullet_\rho \text{ has a minimal element} \tag{3.8}$$

and

$$pre^\bullet_\rho \text{ has a maximal element.} \tag{3.9}$$

Approximative description An abstraction that satisfies the conditions motivated so far deserves a name.

3.2.1.2 D *A description relation $\rho \subseteq$ C $\times$ A is* approximative *i* (C, $\sqsubseteq$) *and* (A, $\preceq$) *are posets, $post^\bullet_\rho$ is $\preceq$-upward-closed and has a minimal element, and $pre^\bullet_\rho$ is $\sqsubseteq$-downward-closed and has a maximal element.*

[10] Note that in our setting, if a is an optimal description of an object c, it may still be the case that there exists another description of c that enjoys strictly more L-properties and hence is a more "useful" description of c. Only when the approximation order $\preceq$ on A coincides with the "L-property ordering", i.e. when the $\Rightarrow$ in condition 3.4 on page 36 may be replaced by $\Leftrightarrow$, then optimal descriptions are also "maximally useful". We have chosen for this setting because we think that in applications of Abstract-Interpretation frameworks this decoupling sometimes occurs naturally. For example, the precision order for Abstract Kripke structures to be defined in Section 4.4 is in terms of the notion of simulation relations (Definition 2.4.2.1). If we would want to prove that this order coincides with the "CTL*-property ordering" on Abstract Kripke structures, this would constitute a proof obligation that we prefer to view as a separate concern. (Indeed, this proof would not be straightforward and would probably only be valid for a restricted class of structures — see the remarks in Section 4.4 on page 78 and cf. [CIY95].) Other authors sometimes identify the precision ($\preceq$) and property orderings. E.g. in [CC79] and [CC94], where abstract objects are identified with properties, the precision order on abstract objects coincides with logical implication.

An alternative way of specifying an approximative description ρ is through an *abstraction relation* α that associates a concrete object with descriptions for it, and a *concretisation relation* γ that associates an abstract object with objects it describes, in such a way that ρ can be "recovered" from (α, γ):

3.2.1.3 D *Let* $\alpha \subseteq \mathrm{C} \times \mathrm{A}$ *and* $\gamma \subseteq \mathrm{A} \times \mathrm{C}$. *The relation* generated *by* (α, γ) *is defined* $gen((\alpha, \gamma)) = \{(c, a) \in \mathrm{C} \times \mathrm{A} \mid \exists_{c', a'} \, (\alpha(c', a') \vee \gamma(a', c')) \wedge c \sqsubseteq c' \wedge a' \preceq a\}$.

We obviously have the following

3.2.1.4 P $post^{\bullet}_{gen((\alpha,\gamma))}$ *is* $\preceq$*-upward-closed and* $pre^{\bullet}_{gen((\alpha,\gamma))}$ *is* $\sqsubseteq$*-downward-closed.*

The reader may wonder why we introduce these abstraction and concretisation relations as a substitute for ρ. The reason is that we want to illustrate how the introduction of further constraints on ρ influences the connection between the (as yet unrelated) α and γ, eventually turning this pair into a Galois connection — but first passing through some "pre-Galois" phases.

We start with the following requirements. If α only gives $\preceq$-minimal descriptions (at least one), and, symmetrically, γ only $\sqsubseteq$-maximal described objects, then (α, γ) is called a *base* for ρ:

3.2.1.5 D *Let* ρ *be an approximative description relation from* C *to* A. *The pair* (α, γ) *is a* base *for* ρ *i* $\alpha \subseteq \mathrm{C} \times \mathrm{A}$, $\gamma \subseteq \mathrm{A} \times \mathrm{C}$, *and:*

1. $gen((\alpha, \gamma)) = \rho$.
2. α *is total on* C *and* γ *is total on* A.
3. $\forall_{c \in \mathrm{C}} \, post^{\bullet}_{\alpha}(c) \subseteq \min_{\preceq}(post^{\bullet}_{\rho}(c))$ *and* $\forall_{a \in \mathrm{A}} \, post^{\bullet}_{\gamma}(a) \subseteq \max_{\sqsubseteq}(pre^{\bullet}_{\rho}(a))$.

A given approximative description may have several di erent bases. Every pair (α, γ) that is a base of some approximative description is characterised by the properties given in the following lemma. Property 3.10 can be illustrated as follows (3.11 is symmetrical). Start from a concrete object c and take an abstract object a that is related to c via α or γ. Then, move to a less precise description $a' \succeq a$ and from there, return to the concrete side via γ, getting to c'. Now it cannot be the case that you end up strictly below the starting point c.

3.2.1.6 L *Let $\alpha \subseteq \mathrm{C} \times \mathrm{A}$ and $\gamma \subseteq \mathrm{A} \times \mathrm{C}$. Then (α, γ) is a base of an approximative description relation if and only if α is total on* C*, γ is total on* A*, and for all $c, c' \in \mathrm{C}$ and $a, a' \in \mathrm{A}$ both of the following hold:*

$$(\alpha(c,a) \vee \gamma(a,c)) \wedge a \preceq a' \wedge \gamma(a',c') \;\Rightarrow\; c' \not\sqsubset c \tag{3.10}$$

$$(\gamma(a,c) \vee \alpha(c,a)) \wedge c \sqsupseteq c' \wedge \alpha(c',a') \;\Rightarrow\; a' \not\prec a \tag{3.11}$$

P .

$\Rightarrow$ Suppose that ($\alpha \subseteq \mathrm{C} \times \mathrm{A}, \gamma \subseteq \mathrm{A} \times \mathrm{C}$) is a base of the approximative description relation ρ. By point 2 of Definition 3.2.1.5, α is total on C and γ is total on A. Next, we show that 3.10 is satisfied; 3.11 is symmetric. Suppose $\alpha(c,a) \vee \gamma(a,c)$, $a \preceq a'$ and $\gamma(a',c')$. Because (α, γ) is a base of ρ, we have by point 1 of Definition 3.2.1.5 that $\mathit{gen}((\alpha,\gamma)) = \rho$. So, from $\alpha(c,a) \vee \gamma(a,c)$ we have $\rho(c,a)$ and from $\gamma(a',c')$ we have $\rho(c,a')$. Because ρ is an approximative description, we have from $\rho(c,a)$ and $a \preceq a'$ (see requirement 3.5) that $\rho(c,a')$. So both c and c' are in $\mathit{pre}^{\bullet}_{\rho}(a')$. If furthermore $c' \sqsubset c$, then c' cannot be maximal in $\mathit{pre}^{\bullet}_{\rho}(a')$, so by the requirement on γ in point 3 of Definition 3.2.1.5, it cannot be that $\gamma(a',c')$. So $c' \not\sqsubset c$.

$\Leftarrow$ Suppose that ($\alpha \subseteq \mathrm{C} \times \mathrm{A}, \gamma \subseteq \mathrm{A} \times \mathrm{C}$) with α total on C and γ total on A satisfy 3.10 and 3.11. Let $\rho = \mathit{gen}((\alpha,\gamma))$. First, we show that ρ is an approximative description relation. From the totality of α and γ it follows easily, by the definition of *gen*, that ρ is total on C and A; so ρ is a description relation. By Property 3.2.1.4 we have that $\mathit{post}^{\bullet}_{\rho}$ is $\preceq$-upward-closed and $\mathit{pre}^{\bullet}_{\rho}$ is $\sqsubseteq$-downward-closed. Let $c \in \mathrm{C}$. Let $a \in \mathrm{A}$ be such that $\alpha(c,a)$; we show that a is a minimal element of $\mathit{post}^{\bullet}_{\rho}(c)$. Namely, suppose there exists $a' \prec a$ such that $\rho(c,a')$. Then, by the definition of *gen*, there must exist $c' \sqsupseteq c$ such that $\alpha(c',a')$ or $\gamma(a',c')$. This is excluded by 3.11. So $\mathit{post}^{\bullet}_{\rho}(c)$ has a minimal element for every c. Symmetrically, using 3.10, it can be shown that $\mathit{pre}^{\bullet}_{\rho}(a)$ has a maximal element for every a.

Furthermore, it is now easy to show that (α, γ) is a base of ρ. □

Properties 3.10 and 3.11 are reminiscent of those of a Galois connection (2.3, 2.4 and 2.5 in Section 2.2.3) — it is not di cult to see that they imply weak versions of monotonicity, reductiveness and extensiveness (Section 2.2.2), namely:

1. α and γ are pre-monotonic.
2. (a) The composed[11] relation $\gamma\alpha$ is pre-reductive.
   (b) The composed relation $\alpha\gamma$ is pre-extensive.

[11] Recall that $\gamma\alpha$ means "first γ, then α" while $\gamma \circ \alpha$, which is only used for functions, means "first α, then γ".

Conversely however, pre-monotonicity and pre-reductive/extensiveness are not su - cient to guarantee properties 3.10 and 3.11. Another di erence with Galois connections is that α and γ do not uniquely determine one another in the above situation.

Least descriptions, greatest concretisations, and Galois connections

The next cases that we consider are those in which it is possible to pick the $\preceq$-least (sometimes called *best*) description for any given concrete object, or to pick the $\sqsubseteq$-greatest concretisation for a given description.

3.2.1.7 D *An approximative description relation $\rho \subseteq C \times A$ is* A-principal (C-principal) *i $post^{\bullet}_{\rho}$ has a $\preceq$-least element ($pre^{\bullet}_{\rho}$ has a $\sqsubseteq$-greatest element).*

Note that for an A-principal ρ, $post^{\bullet}_{\rho}(c)$ is a principal filter for any c, while for a C-principal ρ, $pre^{\bullet}_{\rho}(a)$ is a principal ideal for any a (Section 2.2.1).

Again, we study necessary and su cient properties of a pair (α, γ) to be a base of (A- or C-)principal description relations. The corresponding strengthened versions of properties 3.10 and 3.11 are:

$$(\alpha(c, a) \vee \gamma(a, c)) \wedge a \preceq a' \wedge \gamma(a', c') \;\Rightarrow\; c' \sqsupseteq c \tag{3.12}$$

$$(\gamma(a, c) \vee \alpha(c, a)) \wedge c \sqsupseteq c' \wedge \alpha(c', a') \;\Rightarrow\; a' \preceq a \tag{3.13}$$

We have:

3.2.1.8 L *Let $\alpha \subseteq C \times A$ and $\gamma \subseteq A \times C$. Then (α, γ) is a base of a* C-*principal (*A-*principal) approximative description relation if and only if α is total on* C*, γ is total on* A*, and for all $c, c' \in C$ and $a, a' \in A$, properties 3.12 and 3.11 (resp. properties 3.10 and 3.13) hold.*

P .

$\Rightarrow$ Suppose that ($\alpha \subseteq C \times A$, $\gamma \subseteq A \times C$) is a base of the C-principal description relation ρ. By Lemma 3.2.1.6, the conditions on totality as well as 3.11 are satisfied. We show that 3.12 is satisfied. Suppose $\alpha(c, a) \vee \gamma(a, c)$, $a \preceq a'$ and $\gamma(a', c')$. Like in the proof of Lemma 3.2.1.6, both c and c' are in $pre^{\bullet}_{\rho}(a')$. Because, by the C-principality of ρ, $pre^{\bullet}_{\rho}(a')$ has a (unique) greatest element, and hence $\max_{\sqsubseteq}(pre^{\bullet}_{\rho}(a'))$ is a singleton, it follows from $\gamma(a', c')$ and the requirement on γ in point 3 of Definition 3.2.1.5 that this greatest element must be c'. Hence $c' \sqsupseteq c$.

The case of an A-principal description relation is similar.

⇐ Suppose that ($\alpha \subseteq C \times A, \gamma \subseteq A \times C$) with α total on C and γ total on A satisfy 3.12 and 3.11. Let $\rho = gen((\alpha, \gamma))$. First, we show that ρ is a C-principal description relation. The fact that ρ is an approximative description relation follows from Lemma 3.2.1.6. Let $a \in A$. Let $c \in C$ be such that $\gamma(a, c)$; we show that c is the greatest element of $pre^{\bullet}_{\rho}(a)$. Namely, let c'' be an aritrary element in $pre^{\bullet}_{\rho}(a)$. Then, by the definition of *gen*, there must exist $c' \sqsupseteq c''$ and $a' \preceq a$ such that $\alpha(c', a')$ or $\gamma(a', c')$. In that case, 3.12 implies that $c \sqsupseteq c'$, and hence, by transitivity, $c \sqsupseteq c''$. It is now easy to show that (α, γ) is a base of ρ.

The case that properties 3.10 and 3.13 hold is similar. □

From property 3.12 it can easily be seen that for a base (α, γ) of a C-principal description, γ is functional and monotonic, and $\alpha\gamma$ is extensive. Symmetrically, we have for a base of an A-principal description: α is functional and monotonic, and $\gamma\alpha$ is reductive. As long as a description relation is not both C- and A-principal, α and γ do not determine each other. However:

3.2.1.9 C *Let $\alpha \subseteq C \times A$ and $\gamma \subseteq A \times C$. Then (α, γ) is the base of a* C- *and* A-*principal description relation if and only if it is a Galois connection.*

A Galois connection (α, γ) enjoys many properties, see Section 2.2.3. We recall that α distributes over arbitrary lubs (if they exist) and γ over arbitrary glbs (if they exist), and that the two adjoints determine each other by $\alpha(c) = \bigwedge\{a \mid c \sqsubseteq \gamma(a)\}$ and $\gamma(a) = \bigsqcup\{c \mid \alpha(c) \preceq a\}$, where $\bigwedge$ and $\bigsqcup$ denote the glb on $(A, \preceq)$ and the lub on $(C, \sqsubseteq)$ respectively.

Galois insertions

A situation that occurs in many applications (e.g. see [CC77], [AH87] and also Chapter 4 of this thesis) is that (α, γ) forms a Galois connection from C to A, while in addition $\gamma\alpha$ is the identity function. One reason for this is that $\preceq$ is often defined via γ and $\sqsubseteq$ by $a \preceq a' \Leftrightarrow \gamma(a) \sqsubseteq \gamma(a')$; that this renders $\gamma\alpha$ the identity is stated in the lemma below. Another motivation is based on the observation that descriptions a may always safely be replaced by $\alpha(\gamma(a))$, which yields the same or a better description. Thus, the abstract domain may be *normalised* to $\alpha(\gamma(A))$. This is called *reduction* in [CC92a].

In [Ore44], such a Galois connection is called *perfect in* A, [CC92a] calls it a *Galois surjection* and [MSS86] a *Galois insertion from* A *to* C.

3.2.1.10 L (T 5.3.0.6 [CC79]) *Let (α, γ) be a Galois connection from $(C, \sqsubseteq)$ to $(A, \preceq)$. Then the following are equivalent.*

1. $\gamma\alpha = id$.

2. γ *is an embedding.*

3. α *is surjective.*

4. γ *is injective.*

Another consequence is that if C is a complete lattice and (α, γ) is perfect in A then (by the fact that α distributes over lubs and is surjective) A is a complete lattice.

Things are symmetrical for perfectness in C.

The case that (α, γ) is perfect in both A and C does not occur often in the practice of Abstract Interpretation. It implies that the posets $(A, \preceq)$ and $(C, \sqsubseteq)$ are isomorphic. The goal of abstraction, to reason on a less complex model, can clearly not be attained in such a setting.

Abstraction and concretisation frameworks

Following terminology of [Mar93], we refer to the case in which an abstraction function exists as the *abstraction framework*, and the case in which a concretisation function exists as the *concretisation framework*. If none exists, we speak of the *relational framework*. If both exist, Marriott speaks of the *adjoint framework*, however, to distinguish the varieties of this case, we prefer the terms *Galois-connection framework* and *Galois-insertion framework*.

We make a slight generalisation: in case an approximation ordering is not defined (on either side), we silently assume the identity relation as approximation ordering. This allows us to also capture these cases in the relational, abstraction and/or concretisation frameworks.

Power construction

A situation that often occurs is the following. Concrete and abstract spaces C and A resp. are given together with a description relation ρ, but no obvious approximation orderings pre-exist. For example, $C = \mathbb{Z}$, $A = \{\mathsf{neg}, \mathsf{pos}, \mathsf{odd}, \mathsf{even}\}$ and $\rho = \{\ldots, (-2, \mathsf{neg}), (-1, \mathsf{neg}), (0, \mathsf{neg}), (0, \mathsf{pos}), (1, \mathsf{pos}), (2, \mathsf{pos}), \ldots\} \cup \{(1, \mathsf{odd}), (3, \mathsf{odd}), (5, \mathsf{odd}), \ldots\} \cup \{(0, \mathsf{even}), (2, \mathsf{even}), (4, \mathsf{even}), \ldots\}$. In this section, we discuss a construction that may be used to embed such a situation into one of the frameworks mentioned above. In particular, we show how the "elementary" ρ may be turned into a description relation in the concretisation or Galois-insertion frameworks by shifting our point of view from C to $\mathcal{P}(C)$. Such a construction underlies many of the abstract domains used in program analyses, but often remains

implicit[12] as its result is taken as the point of departure. In the Cousots' work, this shifting from C to $\mathcal{P}$(C) is referred to as the construction of the *collecting semantics*[13], see e.g. [CC92b]. [CC94] discusses a large variety of possibilities to lift elementary domains to power sets (see Section 3.4).

A natural way to define an ordering that relates the "precision" of abstract objects is via the subset ordering on *sets* of described concrete elements: $a \preceq b \Leftrightarrow pre^{\bullet}_{\rho}(a) \subseteq pre^{\bullet}_{\rho}(b)$ (note that this ordering satisfies requirement 3.5). In the case of our example, odd $\preceq$ pos and even $\preceq$ pos, while other pairs of descriptions are incomparable. On the concrete side, we see that our interest is in fact shifted to $\mathcal{P}$(C). We might say that ρ is lifted to $\check{\rho} \subseteq \mathcal{P}(\mathrm{C}) \times \mathrm{A}$ defined by $\check{\rho}(D, a)$ i $D \subseteq pre^{\bullet}_{\rho}(a)$. This construction is natural for a number of reasons. First, $\check{\rho}$ still has the intuition of a "description": if $a \in$ A describes both c and c' in C (by ρ), then we may as well say that a describes the set $\{c, c'\}$; this is captured by $\check{\rho}$. Second, the associated "concretisation relation" $pre^{\bullet}_{\rho}(a)$ is functional and monotonic. However, $\check{\rho}$ is not necessarily total on $\mathcal{P}$(C) and hence is not a description relation. In our example, $\{-1, 1\}$ is not described by any abstract element.

We proceed to investigate what is needed to embed this $\check{\rho}$ in one of the frameworks mentioned before. In order to turn $\check{\rho}$ into a description relation, we extend (A, $\preceq$) to (A′, $\preceq'$) and ρ to the description relation ρ' in such a way that for every subset of A there exists a $\preceq'$-upper bound in A′. In order not to alter the intuitive meanings of the descriptions in A, ρ' should extend ρ in a conservative way, i.e. for every $a \in$ A, $pre^{\bullet}_{\rho'}(a) = pre^{\bullet}_{\rho}(a)$. $\preceq$ is then also conservatively extended by defining $a \preceq' b \Leftrightarrow pre^{\bullet}_{\rho'}(a) \subseteq pre^{\bullet}_{\rho'}(b)$ for $a, b \in$ A′. The inclusion of a top element ${}_{\alpha}\top$ in A′ describing all of C is both necessary and su cient for the extension of A; however, more descriptions may be added to provide a richer domain. In particular, combinations of descriptions may be used to describe the unions of the sets they describe individually. In the case of our example, negpos would then describe all integers while oddeven describes all non-negative numbers[14]. We extend $\check{\rho}$ to $\check{\rho}'$ in a conservative fashion by defining $\check{\rho}'(D, a)$ i $D \subseteq pre^{\bullet}_{\rho'}(a)$; now it is total on both $\mathcal{P}$(C) and A′, so it is a description relation.

However, $\check{\rho}'$ still is not necessarily approximative. First, $\preceq'$ is a pre-order. To turn it into a partial order, we consider the quotient A′/$\equiv$ where $a \equiv b$ i $a \preceq' b \wedge b \preceq'$

[12] And justifiedly so: one would not want to complicate the presentation of an application by exposing these technicalities.

[13] The meaning of the adjective "collecting" in the context of semantics often has di erent meanings in other work.

[14] This enrichement indeed boils down to also lifting A to its power set. The interpretation that we gave to the new elements corresponds to the *disjunctive completion* from [CC92b]; we refer to that paper for a discussion of this particular and similar constructions.

a; the definitions of ρ', $\preceq'$ and $\check{\rho}'$ are adapted in the usual way[15]. For our example, this would mean that pos and oddeven are identified by $\equiv$. Second, $post^{\bullet}_{\check{\rho}'}$ should have a $\preceq'$-minimal element, i.e. for every $D \in \mathcal{P}(\mathrm{C})$, there should be a $\preceq'$-minimal element in $\{a \in \mathrm{A}' \mid \check{\rho}'(D, a)\}$. This is satisfied in our example (note that $\{0\}$ has two minimal descriptions in $\mathrm{A}'/\equiv$, namely {neg} and {pos, oddeven}). In general, if $\mathrm{A}'/\equiv$ is finite, $post^{\bullet}_{\check{\rho}'}$ has a $\preceq'$-minimal element.

Under this additional assumption, we have in fact ended up in the concretisation framework:

3.2.1.11 L ( ) *Let* $(\mathrm{A}'/\equiv, \preceq')$, ρ' *and* $\check{\rho}'$ *be as above, and assume that* $post^{\bullet}_{\check{\rho}'}$ *has a* $\preceq'$*-minimal element. Then* $\check{\rho}'$ *is a* $\mathcal{P}(\mathrm{C})$*-principal description relation from* $(\mathcal{P}(\mathrm{C}), \subseteq)$ *to* $(\mathrm{A}'/\equiv, \preceq')$.

P Because A' is a superset of A that is closed under $\preceq'$-upper bounds, $\check{\rho}'$ is total on $\mathcal{P}(\mathrm{C})$. $\check{\rho}'$ is total on A' too, as we have $\check{\rho}'(\emptyset, a)$ for every $a \in \mathrm{A}'$. Hence, $\check{\rho}'$ is a description relation. $(\mathcal{P}(\mathrm{C}), \subseteq)$ is clearly a poset, while from the definition of $\check{\rho}'$ it can easily be seen that $pre^{\bullet}_{\check{\rho}'}$ is $\subseteq$-downward-closed and has a *greatest* element (namely, $pre^{\bullet}_{\rho'}(a)$ for every $a \in \mathrm{A}'/\equiv$). Furthermore, $(\mathrm{A}'/\equiv, \preceq')$ is a poset by construction. $post^{\bullet}_{\check{\rho}'}$ is $\preceq'$-upward-closed: from $\check{\rho}'(D, a)$ and $a \preceq' b$ we have, by the definitions of $\check{\rho}'$ and $\preceq'$ resp.: $D \subseteq pre^{\bullet}_{\rho'}(a)$ and $pre^{\bullet}_{\rho'}(a) \subseteq pre^{\bullet}_{\rho'}(b)$, from which it follows that $D \subseteq pre^{\bullet}_{\rho'}(b)$, i.e. $\check{\rho}'(D, b)$. Finally, $post^{\bullet}_{\check{\rho}'}$ has a $\preceq'$-minimal element by assumption. □

A $\mathcal{P}(\mathrm{C})$-principal description relation determines a concretisation function (see the remark about the base of a C-principal description below Lemma 3.2.1.8) that maps each abstract object to the $\subseteq$-largest set that is described. For our example, such a function γ maps, e.g., {neg} to the set of all nonpositive integers and {pos, oddeven} to the set of all nonnegative integers.

So far, we have considered conditions for $\check{\rho}'$ to be an approximative description relation and shown that in that case it is automatically $\mathcal{P}(\mathrm{C})$-principal. We may want in addition that $\check{\rho}'$ is $\mathrm{A}'/\equiv$-principal as well. The following lemma gives a necessary and su cient condition.

3.2.1.12 L (G -) *Assume* $(\mathrm{A}'/\equiv, \preceq')$, ρ' *and* $\check{\rho}'$ *to be as in Lemma 3.2.1.11 (including the condition that* $post^{\bullet}_{\check{\rho}'}$ *has a* $\preceq'$*-minimal element), and assume that* $(\mathrm{A}'/\equiv, \preceq')$ *is a complete lattice (with lub* $\bigvee$*). Then the following are equivalent:*

1. $\check{\rho}' \subseteq \mathcal{P}(\mathrm{C}) \times (\mathrm{A}'/\equiv)$ *is a* $\mathcal{P}(\mathrm{C})$*- and* $\mathrm{A}'/\equiv$*-principal description relation.*
2. ρ' *is an approximative description relation from* $(\mathrm{C}, =)$ *to* $(\mathrm{A}'/\equiv, \preceq')$ *that is* $\mathrm{A}'/\equiv$*-principal.*

[15] I.e. they apply to elements $\tilde{a}$ of the quotient $\mathrm{A}'/\equiv$ as to a representative $a \in \tilde{a}$.

P For the implication $1 \Rightarrow 2$ it su ces to observe that $post^{\bullet}_{\rho'}(c) = post^{\bullet}_{\breve{\rho}'}(\{c\})$ for every $c \in \mathrm{C}$.

We consider the implication $2 \Rightarrow 1$. The fact that $\breve{\rho}'$ is $(\mathcal{P}(\mathrm{C}), \subseteq)$-principal follows from Lemma 3.2.1.11. Next, we show that for every $D \in \mathcal{P}(\mathrm{C})$, $post^{\bullet}_{\breve{\rho}'}(D)$ has a least element. The assumption that ρ' is $\mathrm{A}'/\equiv$-principal determines an abstraction function α' from $(\mathrm{C}, =)$ to $(\mathrm{A}'/\equiv, \preceq')$ (see the remark about the base of an A-principal description below Lemma 3.2.1.8). Define the function $\alpha : \mathcal{P}(\mathrm{C}) \to \mathrm{A}'/\equiv$ by $\alpha(D) = \bigvee\{\alpha'(c) \mid c \in D\}$. Then $\alpha(D)$ is the least element in $post^{\bullet}_{\breve{\rho}'}(D)$ for every $D \in \mathcal{P}(\mathrm{C})$, as we will now show. First, we show that $\alpha(D) \in post^{\bullet}_{\breve{\rho}'}(D)$, i.e., by the definitions of α and $post^{\bullet}_{\breve{\rho}'}$ resp., we have to show that for every $D \in \mathcal{P}(\mathrm{C})$, $\breve{\rho}'(D, \bigvee\{\alpha'(c) \mid c \in D\})$, i.e. $D \subseteq \bigcup\{pre^{\bullet}_{\rho'}(\alpha'(c)) \mid c \in D\}$, i.e. $\forall_{c \in D}\, c \in \bigvee pre^{\bullet}_{\rho'}(\alpha'(c))$. By definition of $\bigvee$, the latter follows from $\forall_{c \in D}\, c \in pre^{\bullet}_{\rho'}(\alpha'(c))$, which is easily seen to be true. Furthermore, the fact that $\alpha(D)$ is the $\preceq'$-least such element follows from the definition of $\bigvee$. □

Note that by the definition of $\preceq'$, the base of $\breve{\rho}'$ is a Galois *insertion* in this case. In the case of our running example, condition 2 of this lemma does not hold. ρ' is not $\mathrm{A}'/\equiv$-principal, because the element $0 \in \mathbb{Z}$ has two incomparable $\preceq'$-minimal descriptions: {neg} and {pos, oddeven}.

This power construction may be viewed as a guideline in designing abstract domains. Given an "elementary" notion of description, as captured by ρ, it suggests conditions on the form of the abstract domain that ensure the existence of a "concretisation connection" (in which only γ needs to be functional) (Lemma 3.2.1.11) or Galois insertion (Lemma 3.2.1.12). The many properties that Galois connections enjoy, turn out to be very useful in practice, e.g. when abstract operators are defined, safety and optimality are to be proven, or abstractions are defined compositionally.

The shift from C to $\mathcal{P}(\mathrm{C})$ occurs naturally in this construction. The goal of abstracting is to reduce complexity in the concrete domain by identifying sets of concrete objects. These are then "replaced" by abstract objects each of which acts as a representation of such a set. However, this does *not* mean that there is a di erent representation for each element of $\mathcal{P}(\mathrm{C})$. If reduction of complexity really has to be achieved, many di erent subsets of C will indeed have the same abstraction. The point is that in the power-set setting, the collection of concrete objects (where these concrete objects are now elements of $\mathcal{P}(\mathrm{C})$, i.e. subsets of C) that map to the same description has a nicer structure: it is partially ordered (by $\subseteq$), the decription relation is $\subseteq$-downward closed (so an abstract object representing some set will also represent every smaller set), and there is a $\subseteq$-greatest set being described.

Similarly, it is a misunderstanding to think that the shift of attention from C to $\mathcal{P}(\mathrm{C})$ causes a "blow-up" of the abstract domain, or makes it more di cult to reduce complexity. Closing A under upper bounds requires in the worst case the addition of a single (top) element. The quotient construction $(\mathrm{A}'/\equiv)$ only reduces

the size of A, as elements are being identified. Requiring $post^{\bullet}_{\check{\rho}'}$ to have a $\preceq'$-minimal element only a ects the abstract domain when it is infinite; in that case lower bounds have to be provided for chains that are infinitely decreasing. Finally, the requirement that $(A'/\equiv, \preceq')$ is a complete lattice (needed to get a Galois insertion) also does not necessarily complicate its structure. In case of our example, this may be fulfilled by adding a best description zero for $\{0\}$, but could also be achieved by replacing neg by sneg only decribing the strictly negative numbers for example.

In the beginning of the next chapter, we will also consider a Galois insertion between *sets* of concrete objects and single abstract objects that is defined in terms of an elementary description relation between concrete and abstract objects. The following example, which returns in Section 4.2.3, serves as a leg-up while briefly summarising the relevant aspects of the power construction.

3.2.1.13 E *See Figure 3.1.* $C = \{d_0, d_1, d_2, d_3\}$ *is the set of concrete ob-*

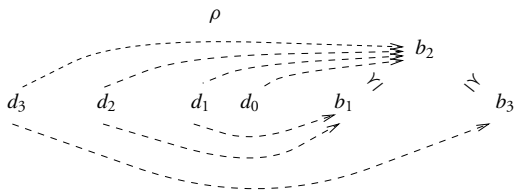


Figure 3.1: Description relation inducing a Galois insertion.

jects, each of which is described by one or more abstract objects in $A = \{b_1, b_2, b_3\}$*, as indicated by the dashed arrows that represent the description relation* ρ*. We embed this in the Galois-insertion framework, as follows. Consider* $(\mathcal{P}(C), \subseteq)$ *as concrete and* $(A, \preceq)$ *as abstract domain where* $b_i \preceq b_j$ *i* $pre^{\bullet}_{\rho}(b_i) \subseteq pre^{\bullet}_{\rho}(b_j)$*. The lifted description relation* $\check{\rho}$ *relates* $D \in \mathcal{P}(C)$ *to* $b \in A$ *i* $D \subseteq pre^{\bullet}_{\rho}(b)$*, i.e. i* $\rho(d, b)$ *for every* $d \in D$*. In the case of this example,* $\check{\rho}$ *is total on both* $\mathcal{P}(C)$ *and* A *so it is a description relation (and so it is not necessary to extend* A*,* ρ *and* $\check{\rho}$ *to* A'*,* ρ' *and* $\check{\rho}'$ *respectively as above). Furthermore,* $\preceq$ *is a partial ordering (so no quotient construction is needed) and for every* $D \in \mathcal{P}(C)$ *there is a* $\preceq$*-least description* a *such that* $\check{\rho}(D, a)$*. So* ρ *satisfies the conditions of point 2 of Lemma 3.2.1.12 and hence, because point 1 now follows,* $\check{\rho}$ *is* $\mathcal{P}(C)$*- and* A*-principal, implying by Corollary 3.2.1.9 that it has a unique base* (α, γ) *that is a Galois insertion from* $(\mathcal{P}(C), \subseteq)$ *to* $(A, \preceq)$*. The concretisation function* γ *maps every description in* A *to the* $\subseteq$*-largest element of* $\mathcal{P}(C)$ *that it describes (via* $\check{\rho}$*). Thus, by definition of* $\check{\rho}$*,* γ *is equal to* $pre^{\bullet}_{\rho}$*, so* $\gamma(b_1) = \{d_1, d_2\}$ *and* $\gamma(b_2) = \{d_0, d_1, d_2, d_3\}$ *for example. The abstraction function* α *maps every* $D \in \mathcal{P}(C)$ *to its* $\preceq$*-least description. Because*

(α, γ) is a Galois connection, $\alpha(D)$ is for every D equal to $\bigwedge\{a \mid D \subseteq \gamma(a)\}$. For example, $\alpha(\{d_2\}) = \alpha(\{d_1\}) = \alpha(\{d_2, d_1\}) = b_1$ and $\alpha(\{d_3, d_1\}) = b_2$.

3.2.2 Strong preservation

The reconstruction of the hierarchy of frameworks in the previous subsection was founded on the assumption of weak preservation of properties in L: condition 3.3. This preservation type is used most commonly in "traditional" abstraction theories like Abstract Interpretation, which is geared towards applications like code optimisation. However, as explained in Section 1.1, if we are interested in verification, a stronger type of preservation may be needed. Below, we will define a number of such strong preservation types and indicate briefly how they a ect the characteristics of the abstraction frameworks.

The main point of *strong perservation* is that not only truth, but also falsehood of (L-)properties is preserved from abstract to concrete objects. We define two types.

3.2.2.1 D

- *ρ is* fine *for* L *i :*

$$\forall_{c\in\mathrm{C},a\in\mathrm{A}}\,[\rho(c,a)\ \Rightarrow\ \forall_{\varphi\in\mathrm{L}}\,[c \models \varphi\ \Leftrightarrow\ a \models^{\alpha} \varphi]] \tag{3.14}$$

- *ρ is* adequate *for* L *i :*

$$\forall_{c\in\mathrm{C},a\in\mathrm{A}}\,[\rho(c,a)\ \Leftrightarrow\ \forall_{\varphi\in\mathrm{L}}\,[c \models \varphi\ \Leftrightarrow\ a \models^{\alpha} \varphi]] \tag{3.15}$$

Fineness is obtained from condition 3.3 by replacing the rightmost implication by a bi-implication. An adequate ρ is, in addition, the maximal description relation that is fine. It relates a concrete object to *every* abstract object that has the same L-properties.

At a first glance, it might seem that the requirement of strong preservation does not leave any room for a reduction of the complexity of a concrete object, because the description should have all the same properties. However, the properties that may be expressed are taken from a limited class (viz., L) and hence any properties outside this class may be abstracted away in the abstract object. As an example, consider transition systems with properties expressed in a branching-time temporal logic like CTL*. A given transition system is described by a minimal bisimilar system. Clearly, the description satisfies exactly the same CTL*-properties, while it is often considerably smaller than the "concrete" system (and in any case not larger).

Let us reconsider the development of the frameworks in Section 3.2.1. The first observation is that the motivation for introducing the orderings $\sqsubseteq$, $\preceq$ has disappeared: we do not want to "trade precision for reduction of complexity" any longer, because this would violate the strong preservation. The point is that if a enjoys strictly more L-properties than a' (note that this does not necessarily imply that $a \preceq a'$), then a may not be replaced by a', because approximation is not safe anymore under the requirement of strong preservation. So, the description relation ρ plays a solitary role in frameworks for strong preservation. We investigate some properties it enjoys.

In the variants of strong preservation defined above, ρ is either equal to or a subset of the relation σ defined by $\sigma(c, a) \Leftrightarrow \forall_{\varphi \in \mathrm{L}} [c \models \varphi \Leftrightarrow a \models^{\alpha} \varphi]$. Clearly, if C and A coincide (and also $\models$ and $\models^{\alpha}$), then σ is an equivalence relation. For disjoint C and A however, the notion of an equivalence relation does not make sense. A better alternative is the notion of a difunctional (Definition 2.1.0.2). It is not difficult to see that σ is a difunctional, i.e. it satisfies a kind of "extended transitivity": if $\sigma(c, a)$, $\sigma(c', a)$ and $\sigma(c', a')$, then $\sigma(c, a')$. A property of a difunctional σ is that the relations $\sigma\sigma^{-1}$ and $\sigma^{-1}\sigma$ are equivalence relations on C and A respectively ([Mal73]), partitioning C and A into the same number of classes, related by a bijection $f : \mathrm{C}/\sigma\sigma^{-1} \to \mathrm{A}/\sigma^{-1}\sigma$ with the property that if $f(C') = A'$, then for all $\varphi \in \mathrm{L}$, for all $c \in C'$ and $a \in A'$, either $c \models \varphi$ and $a \models \varphi$, or $c \not\models \varphi$ and $a \not\models \varphi$. In particular, if $\mathrm{C} = \mathrm{A}$, then this means that σ coincides with the equivalence that is induced by L (page 25).

ρ, being a subset of σ, need not be a difunctional in general. However, it is easily seen that it has to be *consistent* (cf. Definition 9 in [LGS$^+$95]) with L:

3.2.2.2 D ρ *is* consistent with L in $a \in \mathrm{A}$ *i for all* $\varphi \in \mathrm{L}$, $\forall_{c \in \mathrm{C}} [\rho(c, a) \Rightarrow c \models \varphi]$ *or* $\forall_c [\rho(c, a) \Rightarrow c \not\models \varphi]$. ρ *is* consistent with L *i it is consistent with* L *in every* $a \in \mathrm{A}$.

It is this observation that forms the starting point for the Chapters 5 and onwards, which investigate the construction of strongly preserving models.

3.3 Abstract Semantics

The previous section discussed several possibilities for the structure of concrete and abstract domains, as well as for the correspondence relation between them. It concerned, in a sense, the "static" aspects of abstraction theories: we concentrated on the existence and form of elements rather than on their construction. The current section will focus on "dynamic" aspects of frameworks. We assume some notion of

computation that, on the concrete side, reflects the evaluation or interpretation of some computer program.

One could say that there are two reasons to use abstraction to reason about some object. One is that an abstract description of the object is less complex and hence easier to reason about. The second is that the *construction* of a description is less complex. For both reasons, one would not want to construct a description by first computing the concrete object and then abstracting it. Rather, we also want to perform the construction in an "abstract fashion".

For the construction of weakly preserving models, Abstract Interpretation o ers an established theory that is based on the idea of "mimicking" the concrete interpretation function, $\mathcal{I}$, by a similar function that is, roughly, obtained by replacing the operators used in $\mathcal{I}$ by appropriate abstract operators. The resulting abstract interpretation function can be viewed as specifying a way to "abstractly execute" programs[16], so that, in principle, the same mechanisms used in the interpreter/inference engine for "running" a program can be used. This theory is briefly explained in the following subsection.

For the construction of strongly preserving models we propose a di erent approach. We postpone this topic until Chapter 5.

3.3.1 Approximation of fixpoints

Let us move this discussion to a somewhat more concrete and more formal level. The concrete object to be analysed consists of the interpretation $\mathcal{I}(P) \in \mathrm{C}$ of some program[17] $P \in \pounds$. Typically, the function $\mathcal{I}$ is defined in terms of operations like function application and fixpoint computation (reflecting the e ect of the programming constructs in $\pounds$) on (interpretations of) values occurring in P. Of course we may opt to compute the description of $\mathcal{I}(P)$ that we are interested in, in a completely di erent way. However, the idea behind Abstract Interpretation is that this abstract computation is done in a manner that "mimics" the computation of $\mathcal{I}(P)$ as closely as possible. Such a computation can then often be viewed as an "abstract evaluation" of P, in which the values occurring in P are given abstract meanings, while the operations occurring in the definition of $\mathcal{I}$ are replaced by abstract counterparts yielding the *abstract interpretation function* ${}_{\alpha}\mathcal{I}$. The question is then how these abstract meanings and counterparts should be chosen such that the resulting abstract object ${}_{\alpha}\mathcal{I}(P)$ in A is a description of $\mathcal{I}(P)$, i.e. $\rho(\mathcal{I}(P), {}_{\alpha}\mathcal{I}(P))$.

[16] Of course this depends on how far the computation of an interpretation function resembles an execution of the program. In general we could say that the more the programming language is "declarative", the more an execution corresponds to an evaluation of its standard semantics.

[17] $\pounds$ denotes a programming language, whose exact form is irrelevant at this point.

The answer to this question depends very much on the programming language, on the way $\mathcal{I}(P)$ is defined, and on the abstraction framework that is assumed. We review a situation that often occurs in the literature — although it will not play a role in the remainder of this thesis —, namely where $\mathcal{I}(P)$ is defined as the least fixpoint of a monotonic function $f : \mathrm{C} \to \mathrm{C}$ (where f depends on P in some way). The computational ordering, with respect to which the least fixpoint is defined, is assumed to coincide with the concrete approximation order $\sqsubseteq$. Furthermore, we assume that $(\mathrm{C}, \sqsubseteq)$ (which is supposed to be a cpo so that least fixpoints exist) and $(\mathrm{A}, \preceq)$ are related through a Galois connection (α, γ), so that $\rho(\mathcal{I}(P), {}_{\alpha}\mathcal{I}(P))$ is equivalent to $\alpha(\mathcal{I}(P)) \preceq {}_{\alpha}\mathcal{I}(P)$. We do not make any further assumptions about the form of the programming language or the structure of the function f — hence, we stay at a very general level. At the end of this section, the interested reader finds some pointers to the literature on this subject.

So, suppose that $\mathcal{I}(P) = \mathrm{lfp}(f)$. To "mimic" $\mathcal{I}(P)$ in the abstract domain, we look for abstract counterparts ${}_{\alpha}\mathrm{lfp}$ and ${}_{\alpha}f$ such that

$$\alpha(\mathrm{lfp}(f)) \preceq {}_{\alpha}\mathrm{lfp}({}_{\alpha}f) \tag{3.16}$$

We choose to take for ${}_{\alpha}\mathrm{lfp}$ the least fixpoint operator on the abstract domain (although different choices exist, for example in the form of widening and narrowing techniques, see [CC77, CC92c, CC92b]). The following result specifies a condition on ${}_{\alpha}f$ in order for 3.16 to hold. It appears in many of the papers by the Cousots and is sometimes referred to as the *fundamental theorem*[18]. The elegant proof included below is reproduced from [Mat94], where the fundamental theorem is called *μ-fusion*.

3.3.1.1 L *Let $(\mathrm{C}, \sqsubseteq)$ be a cpo and $f : \mathrm{C} \to \mathrm{C}$ a monotonic function. Let $(\mathrm{A}, \preceq)$ be a poset, ${}_{\alpha}f : \mathrm{A} \to \mathrm{A}$ be monotonic and (α, γ) a Galois connection from $(\mathrm{C}, \sqsubseteq)$ to $(\mathrm{A}, \preceq)$. Suppose that*

$$\alpha \circ f \preceq {}_{\alpha}f \circ \alpha \tag{3.17}$$

Then $\alpha(\mathrm{lfp}(f)) \preceq \mathrm{lfp}({}_{\alpha}f)$.

P .

$\alpha \circ f \preceq {}_{\alpha}f \circ \alpha$

$\Rightarrow$ { γ is monotonic }

$\gamma \circ \alpha \circ f \circ \gamma \sqsubseteq \gamma \circ {}_{\alpha}f \circ \alpha \circ \gamma$

[18] The earliest place it is hinted at is paragraph 8.1 of [CC77], as far as we know. Theorem 7.1.0.4 of [CC79] contains an explicit statement of the result. Proofs, be it of slight variations, appear in [CC92a] (before Proposition 23) and [CC92b] (Proposition 6.12).

$\Rightarrow$ { $\gamma \circ \alpha$ is extensive; $\alpha \circ \gamma$ is reductive }
$f \circ \gamma \sqsubseteq \gamma \circ {}_\alpha f$
$\Rightarrow$ { instantiate pointwise ordering }
$f(\gamma(\mathrm{lfp}({}_\alpha f))) \sqsubseteq \gamma({}_\alpha f(\mathrm{lfp}({}_\alpha f)))$
$\equiv$ { by 2.1 on page 16, ${}_\alpha f(\mathrm{lfp}({}_\alpha f)) = \mathrm{lfp}({}_\alpha f)$ }
$f(\gamma(\mathrm{lfp}({}_\alpha f))) \sqsubseteq \gamma(\mathrm{lfp}({}_\alpha f))$
$\Rightarrow$ { by 2.2 on page 16, for all $c \in \mathrm{C}$, $f(c) \sqsubseteq c \Rightarrow \mathrm{lfp}(f) \sqsubseteq c$ }
$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}({}_\alpha f))$
$\equiv$ { (α, γ) is a Galois connection (Section 2.2.3) }
$\alpha(\mathrm{lfp}(f)) \preceq \mathrm{lfp}({}_\alpha f)$ □

Note that similar to the third through fifth steps in this proof, it can be shown that also $\alpha \circ f \succeq {}_\alpha f \circ \alpha$ implies $\alpha(\mathrm{lfp}(f)) \succeq \mathrm{lfp}({}_\alpha f)$; this result is called *simple μ-fusion* in [Mat94]. A function ${}_\alpha f$ for which 3.17 holds is said to be *safe* for f, or a *safe approximation* of f. Indeed, the description relation ρ may be lifted to the function spaces over C and A, e.g. by defining $\rho(f, {}_\alpha f)$ as 3.17. Just like we did in Section 3.2.1 for the description relation between C and A, we may now again investigate the existence of approximation orderings, of minimal/least descriptions, and of maximal/greatest described objects for the (monotonic-) function spaces $\mathrm{C} \to \mathrm{C}$ and $\mathrm{A} \to \mathrm{A}$. These notions should preferably be "consistent" with the same notions on the level of individual elements of C and A. For example, for a notion $\preceq$ (we use the same symbol) of approximation between abstract functions, ${}_\alpha f \preceq {}_\alpha f'$ should imply that $\mathrm{lfp}({}_\alpha f) \preceq \mathrm{lfp}({}_\alpha f')$. And: if the $\preceq$-least description ${}_\alpha f$ of f exists, it should be such that[19] $\mathrm{lfp}({}_\alpha f) = \alpha(\mathrm{lfp}(f))$.

3.3.1.2 L *Assume the same preconditions as in Lemma 3.3.1.1. If in addition we have*

$$\alpha \circ f = {}_\alpha f \circ \alpha \tag{3.18}$$

Then $\alpha(\mathrm{lfp}(f)) = \mathrm{lfp}({}_\alpha f)$.

P From Lemma 3.3.1.1 and simple μ-fusion. □

[19] Various notions of *optimality* of an abstract function ${}_\alpha f$ with respect to f may be found in the literature; the requirement that $\alpha(\mathrm{lfp}(f)) = \mathrm{lfp}({}_\alpha f)$ is just one. In this thesis, optimality is only used for abstract objects, in the sense as defined in Definition 3.2.1.1, and not for functions over such objects.

In the Galois-*insertion* framework, if such a function ${}_\alpha f$ exists[20], it is defined by ${}_\alpha f = \alpha \circ f \circ \gamma$. The "practical" reason for imposing the stronger condition 3.18 instead of 3.17 is that the requirement $\alpha(\mathcal{I}(P)) \preceq {}_\alpha\mathcal{I}(P)$ can easily be satisfied by defining ${}_\alpha\mathcal{I}$ to be the constant function mapping everything to ${}_\alpha\top$, the "least precise" abstraction that gives no information at all. Such solutions should clearly be avoided; hence a criterion for reasoning about the quality of abstractions is introduced.

3.4 Related Work

For an overview of various examples of, and approaches to, program analysis before the conception of the unifying framework of Abstract Interpretation the reader is referred to the bibliographies in [CC77], [MJ81], [AH87] and [AU77]. A more recent overview article is [JN95].

The idea of viewing a program analysis as an approximate computation operating on descriptions of data appears in computer science as early as 1963 in the work of Naur ([Nau63]). Another early example of the idea is given in [Sin72]. The Cousots are the first to relate standard to non-standard semantics by a Galois insertion [CC77]. They published many papers on the subject around the end of the 70s (see the bibliography of [CC92b]), and, after a period of relative silence (but see [CC84]), displayed a renewed interest in the subject at the beginning of the 90s: [CC92a, CC92c, CC92b]. [CC92a] contains a wealth of examples of "everyday" abstract interpretations, as well as more advanced program analyses. In [CC94], many results from their older articles as well as from others' works are combined and extended, resulting in an abundance of constructions suggesting how to lift relations, functions, pairs, etc. from "elementary" domains to domains consisting of sets (cf. our power construction). It is claimed that by avoiding power-domain constructions and by keeping the computational orders separate from the approximation orders (both on concrete and abstract side), a general approach to the abstract interpretation of higher-order functions is obtained. This is illustrated by a unified analysis that captures many analyses used in the field of functional languages.

Adaptations of the original Galois-connection framework in the realm of various programming paradigms are: [BHA86, Nie88] in the context of functional programs, which drop the requirement of best concretisations; [MS89a, MS92a] in the context of logic programs, which drop the requirement of best descriptions; and [MJ86] in the context of non-recursive programs, which drops both the requirements of best concretisations and best descriptions. See the extensive bibliography of [AH87] for pointers to applications of Abstract Interpretation in the realm of declarative programming languages.

[20] Note that for the function $g = \alpha \circ f \circ \gamma$, the equality $\alpha \circ f = g \circ \alpha$ does not necessarily hold.

Papers presenting comparative studies of the various frameworks are [Mar93] and [CC92b].

In [Mar93], Marriott elaborates a framework that we only sketched superficially in Section 3.3 above. He assumes a *metalanguage* M that is a variant of the simple typed lambda calculus consisting of base types, and product and function spaces, with expressions built from pairing, projection, λ-abstraction, function application (β-reduction), conditional, and least fixpoint operators, over basic expressions and variables. The purpose of this metalanguage is to supply the operators to define concrete and abstract semantic functions (cf. our interpretation functions $\mathcal{I}$ and ${}_{\alpha}\mathcal{I}$). The idea of abstract semantics "mimicking" the concrete semantic function is enforced by the introduction of a standard interpretation I mapping expressions in M to some (concrete) algebra and a non-standard (abstract) interpretation I' from M to an abstract algebra, which is connected to the concrete algebra via a *concretisation family* γ (corresponding to our description relations). General conditions are investigated that ensure that γ is consistent with the interpretations I and I'. The advantage of this metalanguage approach is that correctness is guaranteed of any abstract semantics defined in M. The idea of using a metalanguage was advocated by Nielson in [Nie82, Nie88].

3.5 Concluding Remarks

This chapter has given an overview of abstraction theories. To start with, we have identified two major concerns of such theories. One is how to relate objects to descriptions, and the consequences of this for the preservation of properties. The other is how to construct descriptions in a practical way. Throughout, the distinction between weak and strong preservation has formed a border line.

For the case of weak preservation, Section 3.2 takes a systematic approach by reconstructing a number of frameworks often encountered in literature from scratch. Abstract Interpretation was originally introduced ([CC77]) in the context of the Galois-insertion framework, and in many articles about applications that set-up has been assumed to be *the* framework. Section 3.2 may be viewed as an attempt to understand why exactly such Galois insertions are needed (or convenient). By starting from the basic notion of a description relation and introducing extra conditions one by one, we have been able to unravel several motivations leading to the choice of Galois insertions, while passing by several weaker frameworks, also proposed in the literature, on the way. Also — and this is a new result — we have managed to characterise these weaker frameworks by conditions that are obtained by weakening the defining properties of a Galois connection. The *abstraction framework* (in which

the abstraction relation is a function) will return in the next chapter. A separate subsection was devoted to a situation that often occurs in practice, in which a set of concrete objects is lifted to its power set to embed it into one of the more common frameworks, usually the Galois-insertion framework (also this construction is used in the next chapter). We generalised this construction to weaker frameworks.

In the case of strong preservation, the hierarchy of abstraction frameworks collapses because the notion of approximation is dropped.

Section 3.3 gives a brief impression of the folklore of constructing abstractions. The "fundamental theorems" are reviewed that suggest under which conditions the abstraction of a least fixpoint of a function f can be approximated, or computed precisely, as the least fixpoint of an abstracted function ${}_{\alpha}f$. The fixpoint-based construction methods will not be used explicitly in the remainder. Constructing descriptions in the context of *strong* preservation is deferred until Chapter 5.

Chapter 4

Abstract Interpretation of Nondeterministic Systems

The theory of Abstract Interpretation is applied to construct descriptions, called Abstract Kripke structures, of reactive systems, which preserve properties expressible in the branching-time temporal logic CTL*. In contrast to many traditional applications of Abstract Interpretation, our framework suits the definition of descriptions displaying nondeterminism, maintaining not only universal, but also existential properties. Conditions are identified under which such descriptions are optimal with respect to the given abstract domain. Furthermore, we propose the notion of abstraction families providing a framework for the refinement of descriptions.

4.1 Introduction

This chapter presents an approach to the analysis of reactive programs, formalised in the framework of Abstract Interpretation. We model reactive programs by Kripke structures (Section 2.4), while their properties are specified in temporal logic (Section 2.3). The analysis is based on a notion of description of a Kripke structure that weakly preserves CTL*.

For a long time, applications of Abstract Interpretation have focussed on the analysis of *universal safety* properties, that hold in all states (safety) along all possible executions (universality) of the program.[1] With the advent of reactive systems, interest has broadened to a larger class of properties, also including properties concerning the existence of paths, liveness, timing and even probabilities. This is refected by a stream of publications[2] showing a revived interest in preservation results and Abstract-Interpretation techniques. The theory described in this chapter is based on [DGG94].

Several definitions of descriptions that weakly preserve reactive properties have been proposed in literature before the abovementioned revival, but these are limited to universal properties only, or they only concentrate on the definition of abstract models without indicating how such descriptions can be constructed. As argued in the previous chapter, Abstract Interpretation o ers a theoretical framework in which both aspects, preservation and construction, may conveniently be expressed. The large body of experience with the topic of constructing abstract models by the abstract interpretation of programs may turn out to be useful in coping with the state explosion problem as explained in Section 1.3.

4.1.1 Overview of the chapter

In the next section, we investigate a notion of abstraction of Kripke structures such that CTL* is weakly preserved. In other words, we define objects called *Abstract Kripke structures* and give a description relation ρ relating these to concrete Kripke structures, such that requirement 3.3 on page 36 holds (with CTL* substituted for L). Before embedding this setting in one of the abstraction frameworks discussed in the previous chapter (Section 3.2) by providing notions of approximation and optimality, as will be done in Section 4.4, we introduce a simple programming language and illustrate the process of constructing an abstract model through abstract interpretation of the program text (cf. Section 3.3). This is then formally justified

[1] The notions of universality and safety of a property are not always distinguished as explicitly as we do in this thesis. What we call "universal safety" is often just termed "safety" or "invariance" elsewhere.

[2] [Kur90, CGL92, BBLS92, Loi94, DGG94, CR94, CIY95, Kel95]; see Section 4.9 for a more complete overview and comparison.

in Section 4.4. Section 4.5 identifies conditions under which the abstract model that is constructed by abstractly interpreting a program coincides with the optimal abstractions of Section 4.2. Sections 4.6 and 4.7 deal with some practical aspects. Throughout, we illustrate the theory by means of an elaborate example.

In Section 4.8, we extend the framework of Abstract Interpretation so that constructed abstractions can be incrementally refined, which will be needed when a chosen abstraction turns out to be too gross to allow verification of the properties of interest. Section 4.9 contains an extensive comparison with related work, and Section 4.10 concludes.

4.1.1.1 A *In the rest of this chapter, we fix a Kripke structure* $\mathcal{C} = (\Sigma, \mathrm{R}, \mathrm{I}, \|\cdot\|_{\mathrm{Lit}})$.

4.2 Abstract Kripke Structures

We want to define a description relation $\xi \subseteq \mathcal{KS} \times {}_{\alpha}\mathcal{KS}$ between Kripke structures and Abstract Kripke structures (whose exact definition is yet to be determined) under which CTL* is weakly preserved[3]:

$$\forall_{\mathcal{C}\in\mathcal{KS},\mathcal{A}\in_{\alpha}\mathcal{KS}}\ [\xi(\mathcal{C},\mathcal{A})\ \Rightarrow\ \forall_{\varphi\in\mathrm{CTL}^*}\ [\mathcal{C}\models\varphi\ \Leftarrow\ \mathcal{A}\models\varphi]] \tag{4.1}$$

We choose to define a description relation between transition systems in terms of a more elementary description relation $\rho \subseteq \Sigma \times {}_{\alpha}\Sigma$ between the concrete states (of $\mathcal{C}$, see Assumption 4.1.1.1 above), and *abstract states* (of $\mathcal{A}$). The reason for doing so is that we want to construct abstract models by mimicking the (concrete) semantics of some programming language. Typically, the definition of such a semantic interpretation function involves tests and transformations that are evaluated over states, where a state specifies the valuation of all variables. To build an abstract model, the operations will be evaluated over abstract states that only contain partial information about the valuation of variables.

Thus, we formulate the preservation requirement on the level of individual states, which is a natural sharpening[4] of requirement 4.1:

$$\forall_{c\in\Sigma,a\in_{\alpha}\Sigma}\ [\rho(c,a)\ \Rightarrow\ \forall_{\varphi\in\mathrm{CTL}^*}\ [(\mathcal{C},c)\models\varphi\Leftarrow(\mathcal{A},a)\models\varphi]] \tag{4.2}$$

Thus, the concrete states in Σ take the role of concrete objects, while the abstract states in ${}_{\alpha}\Sigma$ act as descriptions. Neither Σ nor ${}_{\alpha}\Sigma$ come with an obvious approximation ordering. It will turn out to be convenient to assume that this situation is

[3] Henceforth, we use the symbol $\models$ for the satisfaction relations on both the concrete and abstract side.

[4] The term "sharpening" can only be taken formally after defining the set ${}_{\alpha}\mathrm{I}$ of abstract initial states; see Property 4.2.2.2 on page 62.

embedded into the Galois-insertion framework. Therefore, we assume that ${}_\alpha\Sigma$ is a complete lattice with approximation ordering $\preceq$ and that there exists a Galois insertion[5] (α, γ) from $(\mathcal{P}(\Sigma), \subseteq)$ to $({}_\alpha\Sigma, \preceq)$. Recall that then $a \preceq b$ i $\gamma(a) \subseteq \gamma(b)$.

We stress that this construction does not blow up the concrete or abstract state spaces[6]. It is meant to impose some structure on ${}_\alpha\Sigma$, about which we did not make any assumptions so far. On the concrete side, the state space is still Σ. The reason for considering $\mathcal{P}(\Sigma)$ is just that it turns out to be convenient to cast the relation between Σ and ${}_\alpha\Sigma$ in terms of a Galois insertion between $(\mathcal{P}(\Sigma), \subseteq)$ and $({}_\alpha\Sigma, \preceq)$ (see Section 3.2.1 and also Section 4.9.1 for a discussion of the advantages). However, the underlying description relation ρ between Σ and ${}_\alpha\Sigma$, related to γ by $\gamma = pre^\bullet_\rho$, will still be used. The rest of this section indeed shows how this description relation between states lifts to a description relation ξ between Kripke structures (cf. requirement 4.1).

So, we assume that a set ${}_\alpha\Sigma$ of abstract states is given together with the Galois insertion (α, γ) specifying its connection to the concrete states. In order to define an Abstract Kripke structure, we need three more ingredients:

1. A function ${}_\alpha\|\cdot\|_{\mathrm{Lit}}$ specifying the interpretation of literals over abstract states.
2. A set ${}_\alpha\mathrm{I}$ of abstract initial states.
3. An abstract transition relation ${}_\alpha\mathrm{R}$.

These points are considered in the following subsections.

4.2.0.1 D *For a (finite or infinite) sequence $\bar{a} = a_0 a_1 \cdots$ in ${}_\alpha\Sigma$, define* $\gamma(\bar{a}) = \{c_0 c_1 \cdots \mid \forall_i \mathrm{R}(c_i, c_{i+1}) \wedge c_i \in \gamma(a_i)\}$.

4.2.0.2 N *We will usually write $\alpha(c)$ for $\alpha(\{c\})$.*

4.2.1 Valuation of literals

In order to satisfy requirement 4.2 for the literals in CTL*, we must have for every literal p: $(\mathcal{A}, a) \models p \;\Rightarrow\; (\mathcal{C}, \gamma(a)) \models p$. As we intend to use the abstract model to infer properties of the concrete model, we would like as many literals as possible

[5] These assumptions may indeed be viewed as an application of the power construction of Section 3.2.1, be it that in the current case no set ${}_\alpha\Sigma$ is given in advance. Hence, this embedding into the Galois-insertion framework should be viewed as imposing conditions on the form of ${}_\alpha\Sigma$, rather than constructing a new set ${}_\alpha\Sigma'/\equiv$ of abstract states from some pre-existing ${}_\alpha\Sigma$.

[6] In particular, the set ${}_\alpha\Sigma$ consisting of a single top element describing all sets of concrete states would su ce, although it will not be very useful in practice.

to hold in each abstract state. Therefore, we let p be satisfied in a whenever it is satisfied in all concrete states described by a:

4.2.1.1 D *For $p \in \mathrm{Lit}$, define ${}_\alpha\|p\|_{\mathrm{Lit}} = \{a \in {}_\alpha\Sigma \mid \gamma(a) \subseteq \|p\|_{\mathrm{Lit}}\}$.*

This choice determines the valuation of literals in abstract states. The relation $\models \subseteq {}_\alpha\Sigma \times \mathrm{Lit}$ is defined as in clause 1 of Definition 2.4.1.1, where s now denotes an abstract state and $\|\cdot\|_{\mathrm{Lit}}$ has to be replaced by ${}_\alpha\|\cdot\|_{\mathrm{Lit}}$. By this choice, as many literals as possible hold in each abstract state. Namely, it can be shown directly from the definitions of $\models$, $\|\cdot\|_{\mathrm{Lit}}$ and ${}_\alpha\|\cdot\|_{\mathrm{Lit}}$ that for every $a \in {}_\alpha\Sigma$ and $p \in \mathrm{Lit}$, $(\mathcal{A}, a) \models p \Leftrightarrow (\mathcal{C}, \gamma(a)) \models p$.

Note that if $a \in {}_\alpha\Sigma$ is such that $\gamma(a)$ contains concrete states in which p holds *and* concrete states in which $\neg p$ holds, then $a \notin {}_\alpha\|p\|_{\mathrm{Lit}}$ but also $a \notin {}_\alpha\|\neg p\|_{\mathrm{Lit}}$. So, although it is always the case that either $a \models p$ holds, or its negation $a \not\models p$, and similarly for $\neg p$, it may occur that for some a we have neither $a \models p$ nor $a \models \neg p$. In particular, $a \not\models p$ does not necessarily imply that $a \models \neg p$.

Furthermore, less precise states satisfy fewer literals, as can easily be proven from the definitions of $\preceq$, $\models$ and ${}_\alpha\|\cdot\|_{\mathrm{Lit}}$:

4.2.1.2 L *Let $a, a' \in {}_\alpha\Sigma$. If $a' \succeq a$, then for all $p \in \mathrm{Lit}$ $a' \models p \Rightarrow a \models p$.*

4.2.2 Abstract initial states

Requirement 4.2 does not imply 4.1. The term $\mathcal{A} \models \varphi$ in 4.1 is an abbreviation for $\forall_{a \in {}_\alpha\mathrm{I}}\ (\mathcal{A}, a) \models \varphi$. By 4.2, this implies $\forall_{c \in \cup\{\gamma(a) \mid a \in {}_\alpha\mathrm{I}\}}\ (\mathcal{C}, c) \models \varphi$. A sufficient condition for this to imply $\mathcal{C} \models \varphi$ in 4.1, which is an abbreviation for $\forall_{c \in \mathrm{I}}\ (\mathcal{C}, c) \models \varphi$, is that $\cup\{\gamma(a) \mid a \in {}_\alpha\mathrm{I}\} \supseteq \mathrm{I}$. When we have preservation, 4.1, a property φ of the concrete model can be verified by checking it on the abstract model, i.e. by verifying that $\mathcal{A} \models \varphi$. Preferably, this condition is as weak as possible. That means that the set of abstract initial states has to be as small as possible. In general, it is not possible to choose ${}_\alpha\mathrm{I}$ such that $\cup\{\gamma(a) \mid a \in {}_\alpha\mathrm{I}\} = \mathrm{I}$. However, the following choice for ${}_\alpha\mathrm{I}$ yields the smallest set $\cup\{\gamma(a) \mid a \in {}_\alpha\mathrm{I}\}$ that still includes I.

4.2.2.1 D ${}_\alpha\mathrm{I} = \{\alpha(c) \mid c \in \mathrm{I}\}$.

One may wonder why we did not take $\alpha(\mathrm{I})$ as the (single) abstract initial state. The reason is that each element of $\{\alpha(c) \mid c \in \mathrm{I}\}$ is in general at least as precise as the singleton $\alpha(\mathrm{I})$: because α distributes over $\bigcup$ (Section 2.2.3), we have $\alpha(\mathrm{I}) = \bigvee\{\alpha(s) \mid s \in \mathrm{I}\}$ (where $\bigvee$ denotes the least upper bound in ${}_\alpha\Sigma$), so that for each $s \in \mathrm{I}$, $\alpha(s) \preceq \alpha(\mathrm{I})$. Therefore, the set $\bigcup\{\gamma(a) \mid a \in {}_\alpha\mathrm{I}\}$ of concrete states to which

${}_\alpha$I (as defined above) corresponds, is a subset of the concretisation $\gamma(\alpha(\mathrm{I}))$. That it can be a proper subset can be seen in Figure 3.1 on page 48. If we assume that $\mathrm{I} = \{d_3, d_2, d_1\}$, then we get ${}_\alpha\mathrm{I} = \{b_1, b_3\}$ (with "concretisation" $\{d_3, d_2, d_1\}$), while $\alpha(\mathrm{I}) = \{b_2\}$ (with "concretisation" $\{d_3, d_2, d_1, d_0\}$).

As argued above, statewise preservation, 4.2, now implies preservation, 4.1. The following property states something slightly more general, namely that if statewise preservation holds, and furthermore some property holds in a superset of the abstract initial states, then it holds in (the initial states of) the concrete system. The proof is straightforward.

4.2.2.2 P *Let* $\varphi \in \mathrm{CTL}^*$. *If* $\forall_{a \in_\alpha \Sigma}\ (\mathcal{A}, a) \models \varphi \ \Rightarrow\ (\mathcal{C}, \gamma(a)) \models \varphi$ *and* ${}_\alpha\mathrm{I}' \supseteq {}_\alpha\mathrm{I}$, *then* $(\mathcal{A}, {}_\alpha\mathrm{I}') \models \varphi \ \Rightarrow\ \mathcal{C} \models \varphi$.

4.2.3 Abstract transition relations

We now investigate the definition of the abstract transition relation: when is there a transition from abstract state a to b? When defining the transitions that an abstract state a can make, requirement 4.2 tells us that we should take into account the possible transitions of all concrete states described by a. In order for existential properties to be preserved, the abstract transition relation should be such that the existence of a successor of a satisfying certain properties, implies, for each $c \in \gamma(a)$, the existence of a successor of c satisfying the same properties. For preservation of universal properties, the fact that some property holds for all successors of a should imply that it holds for all successors of every $c \in \gamma(a)$.

If there has to be a single abstract transition relation ${}_\alpha\mathrm{R}$ satisfying both requirements, then ρ needs[7] to be a bisimulation between (Σ, R) and $({}_\alpha\Sigma, {}_\alpha\mathrm{R})$, which is a strong restriction. Also, it immediately puts us in a situation where CTL* is *strongly* preserved under ρ. Our solution is to define instead *two* transition relations on ${}_\alpha\Sigma$, one preserving existential properties, and the other universal properties. As a consequence, abstract transition systems will be essentially di erent from concrete systems. In particular, the valuation of CTL* formulae will be defined in such a way that existential formulae may only be evaluated over one type of transitions, and universal formulae only over the other type. It will turn out that because of this fact, abstractions are not necessarily strongly preserving: for $\varphi \in \mathrm{CTL}^*$, it may be the case that neither φ, nor $\neg\varphi$ holds on the abstract model.

[7] See Lemma 6.2.0.5 in Chapter 6.

The constrained abstract transition relation

Consider some $a \in {}_{\alpha}\Sigma$ that has a successor b for which some property $\varphi \in \mathrm{CTL}^*$ holds, i.e. $a \models \exists \mathsf{X}\varphi$. If requirement 4.2 is to hold, then it must be the case that every concrete state in $\gamma(a)$ has a successor in which φ holds. In other words, there exists a set Y of concrete states, each satisfying φ, such that for every state in $\gamma(a)$, there exists a successor in Y. We choose the following condition: b may only be a successor of a if $\mathrm{R}^{\forall\exists}(\gamma(a), Y)$ for some $Y \subseteq \Sigma$ that is described by b, i.e. $Y \subseteq \gamma(b)$. This condition guarantees safety in the sense that existential formulae are preserved, as will be shown further on. We also would like a to have as many successors as possible, and furthermore each of them should be a description of Y that is as precise as possible. The first requirement is satisfied by letting b be a successor of a *whenever* $\mathrm{R}^{\forall\exists}(\gamma(a), Y)$; the second by choosing Y to be minimal and b to be the best description of it, as specified by α (cf. Lemma 3.2.1.8). Formally:

4.2.3.1 D $$ {}_{\alpha}\mathrm{R}^{C}(a, b) \;\Leftrightarrow\; b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \mathrm{R}^{\forall\exists}(\gamma(a), Y')\}\}. $$

${}_{\alpha}\mathrm{R}^{C}$ is called the *constrained (abstract transition) relation*[8]. This definition is illustrated in Figure 4.1. Note that the lower part is similar to Figure 3.1 on page 48. The dashed

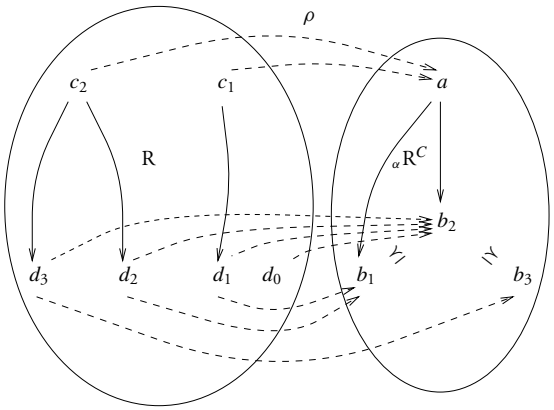


Figure 4.1: The constrained transition relation.

[8]The term "constrained" was chosen because, intuitively, the outgoing transitions from an abstract state a are being constrained by the transitions that are possible from the corresponding concrete states c in $\gamma(a)$: a may only make a transition to b if the cs "agree" in the sense that they can all make a transition into $\gamma(b)$.

arrows represent the description relation ρ, from which the corresponding concretisation and abstraction functions can be derived: e.g. $\gamma(a) = \{c_1, c_2\}$, $\alpha(\{d_3\}) = b_3$, $\alpha(\{d_2\}) = \alpha(\{d_1\}) = \alpha(\{d_2, d_1\}) = b_1$ and $\alpha(\{d_3, d_1\}) = \alpha(\{d_0\}) = b_2$. The constrained transitions from state a lead to both b_1 and b_2. From the point of preservation of properties, the transition to b_1 would be enough. Namely, it can be shown that, because b_1 is more precise than b_2 ($b_1 \preceq b_2$), not only the properties from Lit (see Lemma 4.2.1.2), but indeed all the $\exists$CTL*-properties that b_2 enjoys, are shared by b_1. Hence, any property of the form $\exists \mathsf{X}\varphi$ that holds in a because φ holds in successor b_2 would still hold in a if b_1 were the only successor. In a sense, the two transition systems (the one with and the one without the transition from a to b_2) are "equipotent". This will be formalised in Section 4.4.

Although we based this choice for ${}_\alpha\mathrm{R}^C$ on the preservation of the "single-step" modality $\exists \mathsf{X}$, we have the following preservation property for paths.

4.2.3.2 L *Let $a \in {}_\alpha\Sigma$, $c \in \gamma(a)$ and $\bar{a}$ be an $(({}_\alpha\Sigma, {}_\alpha\mathrm{R}^C), a)$-path. Then there exists a $((\Sigma, \mathrm{R}), c)$-path $\bar{c}$ in $\gamma(\bar{a})$.*

P Assume $\bar{a} = a_0 a_1 \cdots$ with $a_0 = a$. We show that there exists an infinite sequence $\bar{c} = c_0 c_1 \cdots$ of states in Σ such that for all $i \geq 0$, $c_i \in \gamma(a_i)$ and $\mathrm{R}(c_i, c_{i+1})$. $\bar{c}$ is constructed inductively, as follows. Let $c_0 = c$. Then by definition, $c_0 \in \gamma(a_0)$. Now suppose that for some $n \geq 0$, c_n is given such that $c_n \in \gamma(a_n)$. Because ${}_\alpha\mathrm{R}^C(a_n, a_{n+1})$, there must be (by Definition 4.2.3.1 of ${}_\alpha\mathrm{R}^C$) Y such that $\mathrm{R}^{\forall\exists}(\gamma(a_n), Y)$ and $\alpha(Y) = a_{n+1}$. By definition of $\mathrm{R}^{\forall\exists}$, there exists $c_{n+1} \in Y$ such that $\mathrm{R}(c_n, c_{n+1})$. Because (α, γ) is a Galois connection, we have by extensiveness of $\gamma \circ \alpha$ (property 2.5 on page 17) that $\gamma(\alpha(Y)) \supseteq Y$, so $c_{n+1} \in \gamma(a_{n+1})$. Thus, $\bar{c}$ is a $((\Sigma, \mathrm{R}), c)$-path and $\bar{c} \in \gamma(\bar{a})$. □

The free abstract transition relation

Now, consider some $a \in {}_\alpha\Sigma$ such that for every successor b, $\varphi \in$ CTL* holds, i.e. $a \models \forall \mathsf{X}\varphi$. By requirement 4.2, every successor of every concrete state in $\gamma(a)$ should satisfy φ. Because CTL* is (semantically) closed under negation (Definition 2.3.0.1), this may be restated conversely: if some $c \in \gamma(a)$ has a successor for which φ holds, then a must have a successor for which φ holds. We choose the following condition: b must be a successor of a if $\mathrm{R}^{\exists\exists}(\gamma(a), Y)$ and Y is described by b. Again, this condition guarantees safety in the sense that universal formulae are preserved. Furthermore, we also would like a to have as few successors as possible, and furthermore each of them should be a description of Y that is as precise as possible. The first requirement is satisfied by letting b be a successor of a *only* when $\mathrm{R}^{\exists\exists}(\gamma(a), Y)$; the second by choosing Y to be minimal and b to be the best description of it, as specified by α. We get to the following definition:

4.2.3.3 D $\quad {}_{\alpha}\mathrm{R}^{F}(a,b) \;\Leftrightarrow\; b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \mathrm{R}^{\exists\exists}(\gamma(a), Y')\}\}.$

${}_{\alpha}\mathrm{R}^{F}$ is called the *free (abstract transition) relation*[9]. It is illustrated in Figure 4.2.

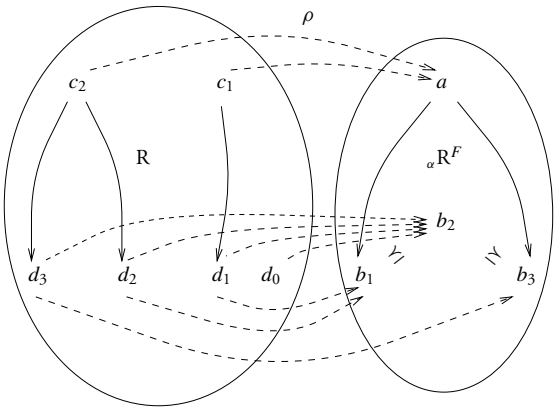


Figure 4.2: The free transition relation.

4.2.3.4 O *For any $a \in {}_{\alpha}\Sigma$, every minimal set Y' for which $\mathrm{R}^{\exists\exists}(\gamma(a), Y')$ holds, is a singleton. For example, in Figure 4.2 these Y' are $\{d_3\}$, $\{d_2\}$ and $\{d_1\}$, with $\alpha(Y')$ being b_3, b_1 and b_1 respectively.*

Again, the preservation of single steps ($\forall$X), on which the above definition is inspired, extends to paths:

4.2.3.5 L *Let $a \in {}_{\alpha}\Sigma$, $c \in \gamma(a)$ and $\bar{c}$ be a $((\Sigma, \mathrm{R}), c)$-path. Then there exists an $(({}_{\alpha}\Sigma, {}_{\alpha}\mathrm{R}^{F}), a)$-path $\bar{a}$ such that $\bar{c} \in \gamma(\bar{a})$.*

P Assume $\bar{c} = c_0 c_1 \cdots$ with $c_0 = c$. Define $\bar{a} = a_0 a_1 \cdots$ with $a_0 = a$ and $a_i = \alpha(c_i)$ for $i \geq 1$. Because (α, γ) is a Galois connection, we have $\gamma(\alpha(c_i)) \supseteq \{c_i\}$, so $c_i \in \gamma(a_i)$ for $i \geq 1$. Also, $c_0 \in \gamma(a_0)$. So $\bar{c} \in \gamma(\bar{a})$.

Because for all $i \geq 0$, $\mathrm{R}(c_i, c_{i+1})$, $c_i \in \gamma(a_i)$, and $a_i = \alpha(c_i)$, we have by Definition 4.2.3.3 of ${}_{\alpha}\mathrm{R}^{F}$: ${}_{\alpha}\mathrm{R}^{F}(a_i, a_{i+1})$. □

By the requirement of minimality of Y in Definitions 4.2.3.1 and 4.2.3.3, it is not in general the case that ${}_{\alpha}\mathrm{R}^{C} \subseteq {}_{\alpha}\mathrm{R}^{F}$, as can be seen by comparing Figures 4.1 and 4.2.

[9]The term "free" was chosen as the opposite of constrained.

In order to accommodate these two di erent transition relations in a single transition system, we give the following definition.

4.2.3.6 D *A* mixed transition system *is a triple* (S, F, C) *consisting of a set* S *of states, and two transition relations* F *and* C *called the free and constrained (transition) relations respectively. A* free path *is a path with all its transitions in* F*; a* constrained path *is a path with all its transitions in* C*. The interpretation of* CTL* *formulae over a mixed system is defined slightly di erent from Definition 2.4.1.1.* $\mathcal{K}$ *is now assumed to be the mixed system* $(S, F, C, \mathrm{I}, \|\cdot\|_{\mathrm{Lit}})$*. Clause 6 is replaced by*

6′. $s \models \forall\psi$ *i for every free* s*-path* π*,* $\pi \models \psi$*;* $s \models \exists\psi$ *i there exists a constrained* s*-path* π *such that* $\pi \models \psi$*.*

We can now define the abstraction of a Kripke structure.

4.2.3.7 D *An* Abstract Kripke structure *is a quintuple* $(S, F, C, J, \|\cdot\|)$ *representing a mixed transition system with initial states* J *and interpretation function* $\|\cdot\|$*. The notion of reachability in such an Abstract Kripke structure is taken relative to the union* $F \cup C$ *of both transition relations, unless explicitly specified otherwise.*

For the Kripke structure $\mathcal{C} = (\Sigma, \mathrm{R}, \mathrm{I}, \|\cdot\|_{\mathrm{Lit}})$ *and the set* ${}_{\alpha}\Sigma$ *of abstract states*[10]*, the abstraction function* $\alpha^M : \mathcal{KS} \to {}_{\alpha}\mathcal{KS}$ *maps* $\mathcal{C}$ *to the Abstract Kripke structure* $({}_{\alpha}\Sigma, {}_{\alpha}\mathrm{R}^F, {}_{\alpha}\mathrm{R}^C, {}_{\alpha}\mathrm{I}, {}_{\alpha}\|\cdot\|_{\mathrm{Lit}})$*, where* ${}_{\alpha}\mathrm{R}^F$*,* ${}_{\alpha}\mathrm{R}^C$*,* ${}_{\alpha}\mathrm{I}$ *and* ${}_{\alpha}\|\cdot\|_{\mathrm{Lit}}$ *are as defined above.*

We then have:

4.2.3.8 T *For every* $\varphi \in$ CTL*, $\alpha^M(\mathcal{C}) \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$.*

P We write $\mathcal{A}^M$ for $\alpha^M(\mathcal{C})$. By Lemma 4.2.2.2, it su ces to prove statewise preservation for every φ in CTL*. This is done by induction on the structure of φ, proving for state formulae that for every state $a \in {}_{\alpha}\Sigma$, $(\mathcal{A}^M, a) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(a)) \models \varphi$ and for path formulae that for every free or constrained path $\bar{a}$ in $\mathcal{A}^M$, $(\mathcal{A}^M, \bar{a}) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(\bar{a})) \models \varphi$.

The base case, $\varphi \in$ Lit, follows directly from Definition 4.2.1.1. The cases that φ is a conjunction or disjunction of state or path formulae, a state formula interpreted over a path, or a path formula with principal operator X, U or V, are straightforward. For φ of the form $\forall\psi$, let a be a state such that $(\mathcal{A}^M, a) \models \forall\psi$, let $c \in \gamma(a)$, and consider a $(\mathcal{C}, c)$-path $\bar{c}$. By Lemma 4.2.3.5, we know that there exists an $(({}_{\alpha}\Sigma, {}_{\alpha}\mathrm{R}^F), a)$-path $\bar{a}$ such that $\bar{c} \in \gamma(\bar{a})$, so, because $(\mathcal{A}^M, a) \models \forall\psi$, $(\mathcal{A}^M, \bar{a}) \models \psi$. By the ind. hyp. we have $(\mathcal{C}, \bar{c}) \models \psi$. So $(\mathcal{C}, \gamma(a)) \models \forall\psi$. The argument for φ of the form $\exists\psi$ is similar. □

[10] Although, for convenience, $\mathcal{C}$ and ${}_{\alpha}\Sigma$ are fixed throughout this chapter, this definition of α^M is understood to generalise for any given concrete Kripke structure and any set of abstract states.

So, Abstract Kripke structures allow verification of full CTL* while the degree of reduction is determined by the choice of the abstract domain and may hence be arbitrarily large. In contrast, reductions with respect to bisimulation equivalence (e.g. [BFH+92]) only allow a limited reduction, in the sense that only states that are bisimilar may be identified by the abstraction. These facts may seem contradictory, but the reader should note that by the definition of satisfaction of CTL* formulae over mixed systems, it is possible that neither φ, nor $\neg\varphi$ holds; this is not possible with bisimulation reduction.

Note that we have defined an abstraction function α^M mapping concrete to abstract systems, but so far we have not introduced an approximation ordering on Abstract Kripke structures. This will be done in Section 4.4, by lifting the approximation relation $\preceq$ on abstract states to Abstract Kripke structures. First, we take a brief look at the topic of constructing Abstract Kripke structures and introduce an elaborate example.

4.3 Abstract Interpretation of Programs

In the previous section we have defined abstract models and proven their preservation properties. The next topic is how to construct such an abstraction for a given program[11] $P \in \pounds$. We do not want to do this by first constructing the full concrete model $\mathcal{I}(P) \in \mathcal{KS}$ of P and then applying α^M to this structure. We would then still have to deal with the possibly unmanageable object $\mathcal{I}(P)$, which is precisely what we wish to avoid. Instead, as explained in the beginning of Section 3.3.1, we will construct an abstract model in a more direct fashion. The idea is to define a "non-standard" interpretation function ${}_\alpha\mathcal{I} : \pounds \to {}_\alpha\mathcal{KS}$ that "mimics" $\mathcal{I}$, in the sense that the operations that are performed in computing $\mathcal{I}$ are performed in a similar way to compute ${}_\alpha\mathcal{I}$, but applied to descriptions of objects rather than to the ojects themselves[12]. One may think of ${}_\alpha\mathcal{I}$ as the function that is obtained by pushing α^M into $\mathcal{I}$, distributing it as much over the operators occurring in $\mathcal{I}$ as possible, preferably until the level of the most elementary values, which usually correspond to the values that program variables can take. In that case, abstract interpretation is a non-standard interpretation over a domain of data descriptions.

To illustrate this, we need to fix a programming language $\pounds$. We use a language that is based on *action systems* [BKS83], that, although being very simple, will help

[11] $\pounds$ denotes the programming language; below, we will become more specific about the syntax of programs.

[12] Ideally, ${}_\alpha\mathcal{I}(P)$ equals $\alpha^M(\mathcal{I}(P))$; indeed, we will later identify conditions under which this is the case for any P.

to grasp the idea of how to abstractly interpret operations in "real" (imperative) programming languages, as it contains rudimentary forms of the common notions of assignment, test and loop. A program is a set of *actions* of the form $c_i(\bar{x}) \to t_i(\bar{x}, \bar{x}')$, where i ranges over some index set J, $\bar{x}$ represents the vector of program variables, c_i is a condition on their values and t_i specifies a transformation of their values into the new vector $\bar{x}'$. A program is run by repeatedly nondeterministically choosing an action whose condition c_i yields *true* and updating the program variables as specified by the associated transformation[13] t_i. We let Val be the set of values that the vector $\bar{x}$ may take, and IVal $\subseteq$ Val the set of values that it may have initially. Thus, each c_i is a predicate over Val and each t_i a relation over Val $\times$ Val.

4.3.0.1 D *The (concrete) interpretation function $\mathcal{I} : \pounds \to \mathcal{KS}$ is defined as follows. For $P = \{c_i(\bar{x}) \to t_i(\bar{x}, \bar{x}') \mid i \in J\}$ in $\pounds$, $\mathcal{I}(P)$ is the transition system* $(\Sigma, \mathrm{I}, \mathrm{R})$ *where:*

- $\Sigma = \mathrm{Val}$.
- $\mathrm{I} = \mathrm{IVal}$.
- $\mathrm{R} = \{(\bar{v}, \bar{v}') \in \mathrm{Val}^2 \mid \exists_{i \in J}\ c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{v}')\}$.

In the following, we fix such a program P and identify $\mathcal{I}(P)$ with the concrete Kripke structure $\mathcal{C}$ (see Assumption 4.1.1.1).

Next, we assume a set ${}_\alpha\mathrm{Val}$ of descriptions of values in Val, via a Galois insertion (α, γ) from $(\mathcal{P}(\mathrm{Val}), \subseteq)$ to $({}_\alpha\mathrm{Val}, \preceq)$, and define two types of non-standard, *abstract* interpretations of the c_i and t_i over ${}_\alpha\mathrm{Val}$, corresponding to the constrained and free transition relations. Note the similarity of these definitions with the Definitions 4.2.3.1 and 4.2.3.3 of ${}_\alpha\mathrm{R}^C$ and ${}_\alpha\mathrm{R}^F$ resp.

4.3.0.2 D *For $i \in J$, let ${c_i}^F$, ${c_i}^C$ be conditions on ${}_\alpha\mathrm{Val}$ and ${t_i}^F$, ${t_i}^C$ be transformations on ${}_\alpha\mathrm{Val} \times {}_\alpha\mathrm{Val}$.*

- ${c_i}^F$ *is a* free abstract interpretation of c_i *i for every* $a \in {}_\alpha\mathrm{Val}$,

$$ {c_i}^F(a) \Leftrightarrow \exists_{\bar{v} \in \gamma(a)}\ c_i(\bar{v}). $$

- ${t_i}^F$ *is a* free abstract interpretation of t_i *i for every* $a, b \in {}_\alpha\mathrm{Val}$,

$$ {t_i}^F(a, b) \ \Leftrightarrow\ b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid {t_i}^{\exists\exists}(\gamma(a), Y')\}\}. $$

[13] As t_i is a relation, there may be several di erent updated states $\bar{x}'$. In this case, one of these is selected nondeterministically.

- ${c_i}^C$ *is a* constrained abstract interpretation of c_i i *for every* $a \in {}_\alpha\mathrm{Val}$,

$$ {c_i}^C(a) \Leftrightarrow \forall_{\bar{v} \in \gamma(a)}\ c_i(\bar{v}). $$

- ${t_i}^C$ *is a* constrained abstract interpretation of t_i i *for every* $a, b \in {}_\alpha\mathrm{Val}$,

$$ {t_i}^C(a,b) \ \Leftrightarrow\ b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid {t_i}^{\forall\exists}(\gamma(a), Y')\}\}. $$

Furthermore, we define the abstract interpretation ${}_\alpha\mathcal{I}(P)$ *of* P *as the system* $\widehat{\mathcal{A}^M} = ({}_\alpha\Sigma, {}_\alpha\widehat{\mathrm{R}^F}, {}_\alpha\widehat{\mathrm{R}^C}, {}_\alpha\mathrm{I})$ *where:*

- ${}_\alpha\Sigma = {}_\alpha\mathrm{Val}$.
- ${}_\alpha\widehat{\mathrm{R}^F} = \{(a,b) \in {}_\alpha\mathrm{Val}^2 \mid \exists_{i \in J}\ {c_i}^F(a) \wedge {t_i}^F(a,b)\}$.
- ${}_\alpha\widehat{\mathrm{R}^C} = \{(a,b) \in {}_\alpha\mathrm{Val}^2 \mid \exists_{i \in J}\ {c_i}^C(a) \wedge {t_i}^C(a,b)\}$.
- ${}_\alpha\mathrm{I} = \{\alpha(\bar{v}) \mid \bar{v} \in \mathrm{IVal}\}$.

${}_\alpha\widehat{\mathrm{R}^F}$ *and* ${}_\alpha\widehat{\mathrm{R}^C}$ *are called the* computed *free and constrained transition relations respectively.*

Of course, the abstract interpretations ${c_i}^F$, ${t_i}^F$ and ${c_i}^C$, ${t_i}^C$ should be e ectively computable. The idea of abstract interpretation is that an analysis tool, when provided with the domain of abstract values and corresponding abstractions of the operators, should be able to automatically evaluate the abstract semantics ${}_\alpha\mathcal{I}(P)$ of any program P.

Before we proceed to formally justify this definition (by relating ${}_\alpha\widehat{\mathrm{R}^C}$ to ${}_\alpha\mathrm{R}^C$ and ${}_\alpha\widehat{\mathrm{R}^F}$ to ${}_\alpha\mathrm{R}^F$), we introduce an example.

4.3.1 Example: dining mathematicians

Consider the system consisting of the two concurrent processes depicted in Figure 4.3, which is a parallel variant of the Collatz ("$3n + 1$") program. We chose this example because it is small but nevertheless displays a non-trivial interplay between data and control. The properties that we will verify concern certain control aspects that depend on the values that the integer variable n takes under the various operations that are performed on it. Because the state space is infinite, data-abstraction will be necessary to verify aspects of the control-flow. As such, it is a simple illustration of the fact that abstraction techniques bring into reach the model checking of systems that cannot be verified through the standard approach.

The program may be viewed as a protocol controlling the mutually exclusive access to a common resource of two concurrent processes, modelling the behaviour

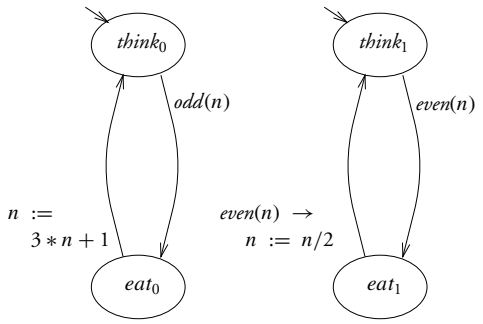


Figure 4.3: The dining mathematicians.

of two mathematicians, numbered 0 and 1. They both cycle through an infinite sequence of "think" and "eat" states. The right to enjoy a meal in strict solitude is regulated by having them inspect the value of n before eating, letting the one go ahead only if n has an odd value, and the other only if n is even. Upon exit from the dining room, each mathematician has her own procedure for assigning a new value to n. Transitions can only be taken when the enabling conditions are satisfied, e.g. mathematician 1 can only leave the dining room if n is divisible by 2. An execution is any infinite sequence of (arbitrarily) interleaved steps of both processes that starts in a state where both mathematicians are in their thinking state, and n is set to some arbitrary positive integer value. We want to verify that along every execution the following properties hold:

- The mathematicians have mutually exclusive access to the dining room.
- The mathematicians will not starve, i.e. when one mathematician is eating, then eventually the other will get access to the dining room.

In order to formalise this, we first express the program as an action system[14]: see Figure 4.4. As data and control are treated uniformly in such systems, we introduce variables ℓ_0 and ℓ_1, both ranging over $\{think, eat\}$, to encode the e ect of "being in a location" $think_i$ or eat_i. The state space Σ of this program is the set $\{think, eat\}^2 \times \mathbb{N}\backslash\{0\}$ of values that the vector $\langle \ell_0, \ell_1, n\rangle$ of program variables may assume. The initial states are $I = \{\langle think, think, n\rangle \mid n \in \mathbb{N}\backslash\{0\}\}$. The transitions are defined as in Definition 4.3.0.1, using the standard interpretations of the tests $=$, *even*, *odd*

[14]We use the more operational notion of assignment, :=, rather than the primed variables of Definition 4.3.0.1. Also note that, although we translate a concurrent into a sequential system, we do not have to "unfold" the inherent non-determinism: the two processes describing the mathematicians can be recognised in the first two lines and last two lines of the program in Figure 4.4.

$$\ell_0 = \mathit{think},\ \mathit{odd}(n) \longrightarrow \ell_0 := \mathit{eat} \tag{Act1}$$
$$\ell_0 = \mathit{eat} \longrightarrow \ell_0 := \mathit{think},\ n := 3 * n + 1 \tag{Act2}$$
$$\ell_1 = \mathit{think},\ \mathit{even}(n) \longrightarrow \ell_1 := \mathit{eat} \tag{Act3}$$
$$\ell_1 = \mathit{eat},\ \mathit{even}(n) \longrightarrow \ell_1 := \mathit{think},\ n := n/2 \tag{Act4}$$

Figure 4.4: The dining mathematicians: action system.

and operations $3*$, $+1$ and $/2$ (the latter three are considered as operations on one argument, i.e. functional binary relations), where $/2$ is assumed to be defined for even numbers only.

The properties to be verified are expressed in CTL* as follows.

$$\forall\mathsf{G}\neg(\ell_0 = \mathit{eat} \wedge \ell_1 = \mathit{eat}) \tag{4.3}$$
$$\forall\mathsf{G}((\ell_0 = \mathit{eat} \rightarrow \forall\mathsf{F}\ell_1 = \mathit{eat})) \tag{4.4}$$
$$\forall\mathsf{G}((\ell_1 = \mathit{eat} \rightarrow \forall\mathsf{F}\ell_0 = \mathit{eat})) \tag{4.5}$$

As these formulae are in $\forall$CTL*, we can verify them via an abstraction with free transitions only.

The abstract domain is defined by providing abstractions of the components that comprise the concrete domain. We choose to leave the components $\{\mathit{think}, \mathit{eat}\}$ the same. Formally, this means that we take an abstract domain containing elements think and eat whose concretisations are $\{\mathit{think}\}$ and $\{\mathit{eat}\}$ respectively. To abstract $\mathbb{N} \setminus \{0\}$, we choose an abstract domain in which n may take the values e and o, describing the even and odd positive integers respectively, i.e. $\gamma(\mathsf{e}) = \{2, 4, 6, \dots\}$ and $\gamma(\mathsf{o}) = \{1, 3, 5 \dots\}$. To both abstract domains, we add a top element $\top$. The set ${}_\alpha\Sigma$ of abstract states is now defined as follows.

$${}_\alpha\Sigma = \{\mathsf{think}, \mathsf{eat}, \top\}^2 \times \{\mathsf{e}, \mathsf{o}, \top\}$$

Its top element is $\langle\top, \top, \top\rangle$, while the approximation relation $\preceq$ is the obvious extension of the orderings on each of the three components. It is easily verified that the concretisation function thus defined determines a Galois insertion (α, γ) from $(\mathcal{P}(\Sigma), \subseteq)$ to $({}_\alpha\Sigma, \preceq)$. For the abstract initial states we have, according to Definition 4.2.2.1:

$${}_\alpha\mathrm{I} = \{\langle\mathsf{think}, \mathsf{think}, \mathsf{e}\rangle, \langle\mathsf{think}, \mathsf{think}, \mathsf{o}\rangle\}$$

Having chosen an abstract domain, we also have to provide abstract interpretations, over this domain, of the operations that appear in the program, along the lines of Definition 4.3.0.2. Tables a and b in Figure 4.5 give the definitions of the free abstract interpretations of the transformations and tests[15] on the abstract domain $\{\mathsf{e}, \mathsf{o}, \top\}$. The operations $3*$, $+1$ and $/2$ are considered single symbols. For completeness, Figure 4.5c gives the table with free abstract interpretations of the tests $=$ *think* and $=$ *eat* (to be considered single symbols) on the domain $\{\mathsf{think}, \mathsf{eat}, \top\}$. The tables have to be interpreted as indicated by the following examples. The entry *true* in Table b, row $even^F$, column e, indicates that $even^F(\mathsf{e})$ holds, i.e. (cf. Definition 4.3.0.2), $\exists_{n \in \gamma(\mathsf{e})}\ even(n)$. The entry *false* in Table a, row $+1^F$, column (e, e), means that $+1^F(\mathsf{e}, \mathsf{e})$ is false, i.e. for any minimal Y such that $+1^{\exists\exists}(\gamma(\mathsf{e}), Y)$, we have $\alpha(Y) \neq \mathsf{e}$ (see Definitions 4.3.0.2 and 2.1.0.1). From these diagrams we see

F :	(e, e)	(e, o)	(e, ⊤)	(o, e)	(o, o)
$3*^F$	*true*	*false*	*false*	*false*	*true*
$+1^F$	*false*	*true*	*false*	*true*	*false*
$/2^F$	*true*	*true*	*false*	*false*	*false*

F :	(o, ⊤)	(⊤, e)	(⊤, o)	(⊤, ⊤)
$3*^F$	*false*	*true*	*true*	*false*
$+1^F$	*false*	*true*	*true*	*false*
$/2^F$	*false*	*true*	*true*	*false*

(a)

F :	e	o	⊤
$even^F$	*true*	*false*	*true*
odd^F	*false*	*true*	*true*

(b)

F :	think	eat	⊤
$(= think)^F$	*true*	*false*	*true*
$(= eat)^F$	*false*	*true*	*true*

(c)

Figure 4.5: Free abstract interpretations of operations (a) and tests (b and c).

for example that $/2^F$ is not functional (Table a, row $/2^F$, first two columns, as well as the columns for $(\top, \mathsf{e})$ and $(\top, \mathsf{o})$), illustrating that a function may become a relation when being abstracted. The abstract interpretation of the composed operation $3 * ... + 1$ that occurs as a transformation in the program, is now obtained by composition of the abstract interpretations of the constituents.

[15] The abstract interpretations extend to ${}_\alpha\Sigma$ in the obvious way.

Observe that both $3*^F(\top, \mathsf{e})$ and $3*^F(\top, \mathsf{o})$ hold, but that $3*^F(\top, \top)$ is *false*. Although this is in correspondence with Definition 4.2.3.3, it may seem surprising as, intuitively, the result of multiplying an unknown number by 3 yields an unknown number again. However, it should be noted that we are dealing with the *free* interpretation of the operation $3*$ here, so that the transitions in the Abstract Kripke structure will be universally quantified. Hence, the intuition of the interpretations $3*^F(\top, \mathsf{e})$ and $3*^F(\top, \mathsf{o})$ should be as follows: starting from an unknown number, multiplication by 3 may yield an even number and may yield an odd number.

Now we can abstractly interpret the program over this abstract domain, using the interpretations given in the tables. We start in the two initial states $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$ and $\langle \mathsf{think}, \mathsf{think}, \mathsf{o} \rangle$. Consider for example $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$. According to Tables b and c, the only action from the program whose condition c_i evaluates to *true* is Act3 (see Figure 4.4). As a result of the corresponding transformation ($\ell_1 := \mathit{eat}$), the (only) successor of $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$ is $\langle \mathsf{think}, \mathsf{eat}, \mathsf{e} \rangle$. Continuing from this state, the only action that applies is Act4. From the entries for the operation $/2^F$ on the value e, we see that the result can be both e and o. Hence, we get free abstract transitions from $\langle \mathsf{think}, \mathsf{eat}, \mathsf{e} \rangle$ back to $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$, and also to $\langle \mathsf{think}, \mathsf{think}, \mathsf{o} \rangle$. Such an abstract execution yields the abstract model of Figure 4.6. In this model,

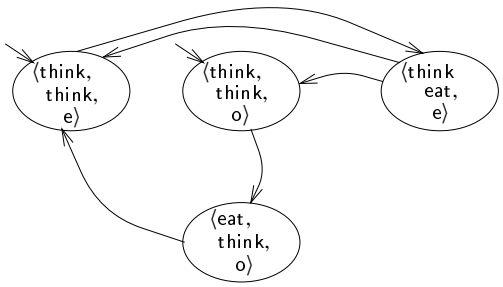


Figure 4.6: The free abstract model.

only those states are shown that are reachable along the computed free transition relation.

In order to illustrate the use of the constrained abstraction, we consider a small extension to the program: we add a third concurrent process that can "restart" the system by setting n to value 100. This may only be done when both mathematicians are thinking (otherwise there may be executions possible that violate the mutual exclusion property). To this e ect, the following fifth action is added to the program:

$$\ell_0 = \mathit{think},\ \ell_1 = \mathit{think} \longrightarrow n := 100 \qquad \text{(Act5)}$$

We want to check whether along every path, in every state there is a continuation of the path that reaches a "restart" state. Writing *restart* for $\ell_0 = \mathit{think} \wedge \ell_1 = \mathit{think} \wedge n = 100$, this property is expressed in CTL* by:

$$\forall \mathsf{G} \exists \mathsf{F}\, \mathit{restart} \tag{4.6}$$

We extend the abstract domain for n by the value 100, where $\gamma(100) = \{100\}$. Formula 4.6 being in full CTL*, we need a mixed transition system. Instead of providing the constrained abstract interpretations of all tests and operations over all abstract values, Figure 4.7 only provides those entries that are needed to construct the system of Figure 4.8 (in which only *part* of the reachable state space is shown — see below). Also, the tables of Figure 4.5 have to be extended to take into account the new abstract value 100. Being straightforward, these extensions are left to the reader.

C .:	(e, 100)	(e, e)	(e, o)	(e, ⊤)
$/2^C$	*false*	*false*	*false*	*true*

C .:	(100, 100)	(100, e)	(100, o)	(100, ⊤)
$/2^C$	*false*	*true*	*false*	*false*

(a)

C .:	(o, 100)	(o, e)	(o, o)	(o, ⊤)
$3*^C$	*false*	*false*	*true*	*false*
$+1^C$	*false*	*true*	*false*	*false*

(b)

C .:	100	e	o	⊤
$even^C$	*true*	*true*	*false*	*false*
odd^C	*false*	*false*	*true*	*false*

(d)

C .:	think	eat	⊤
$= think^C$	*true*	*false*	*false*
$= eat^C$	*false*	*true*	*false*

(e)

Figure 4.7: Constrained abstract interpretations of operations (a, b, c) and tests (d and e).

The relevant part of the constructed Abstract Kripke structure is depicted in Fig-

ure 4.8. Solid arrows denote free transitions, dashed arrows represent constrained transitions. Not all reachable states are shown, but only those that can be reached from the initial states by taking a (finite) number of free transitions followed by a number of constrained transitions. The paths in this model are the only ones that are needed for checking formula 4.6, because there are only two path quantifiers in that formula, namely a universal followed by an existential path quantifier (cf. Definitions 2.4.1.1 and 4.2.3.6 giving the interpretation of CTL* over mixed systems). The complete reachable state space includes 10 more abstract states, which are only reachable via free transitions starting from $\langle \mathsf{think}, \mathsf{think}, \top \rangle$. Again, we see that it need not be the case that ${}_\alpha\mathrm{R}^C \subseteq {}_\alpha\mathrm{R}^F$, as is illustrated by the arrow from $\langle \mathsf{think}, \mathsf{eat}, \mathsf{e} \rangle$ to $\langle \mathsf{think}, \mathsf{think}, \top \rangle$ for example. Before checking properties 4.3–4.6 on this ab-

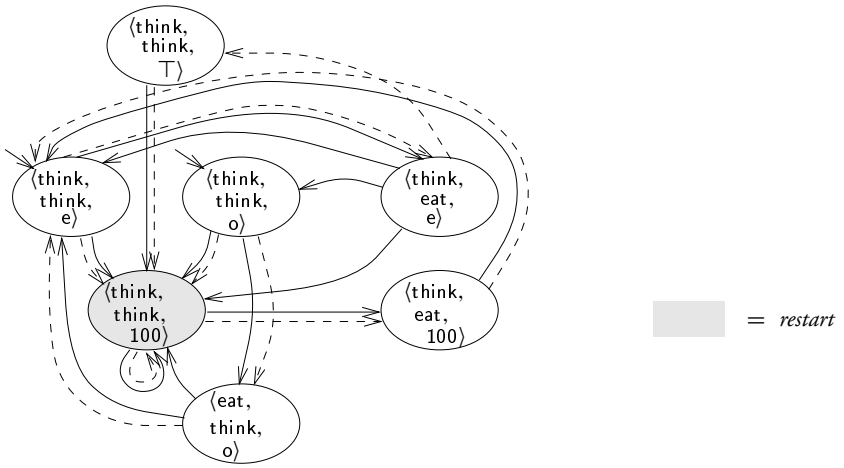


Figure 4.8: The mixed abstract model for the modified program.

straction, we have to establish the correctness of our approach by showing how the Abstract Kripke structures that result from the abstract interpretation of programs relate to the (optimal) abstractions that were defined (through α^M) in Section 4.2.

4.4 Approximations

We have defined in Section 4.2 a notion of abstraction of a Kripke structure, through a function $\alpha^M : \mathcal{KS} \to {}_\alpha\mathcal{KS}$. In the previous section, we have defined an abstract interpretation function ${}_\alpha\mathcal{I} : \pounds \to {}_\alpha\mathcal{KS}$ mapping programs directly to Abstract Kripke structures. This abstract interpretation function was defined in terms of abstract interpretations of the test and transformation operators, inspired by the definitions of

the abstract transition relations ${}_\alpha\mathrm{R}^C$ and ${}_\alpha\mathrm{R}^F$. We have expressed our hope that it would turn out that ${}_\alpha\mathcal{I}(P) = \alpha^M(\mathcal{I}(P))$ for every program P, however, we have postponed answering this question so far.

The answer turns out to be *no* — which can be seen if we extend Figure 4.8 by considering all the computed free abstract transitions starting from state $\langle$think, think, $\top\rangle$ (currently, only the transition to $\langle$think, think, 100$\rangle$ is shown). The successors are then $\langle$eat, think, $\top\rangle$, $\langle$think, eat, $\top\rangle$, and $\langle$think, think, 100$\rangle$, corresponding to choosing the first, third and fifth actions in the program respectively. On the other hand, the free successors of $\langle$think, think, $\top\rangle$ according to ${}_\alpha\mathrm{R}^F$ (Definition 4.2.3.3) are di erent, which can be seen as follows. Consider the concretisation, $\gamma(\langle$think, think, $\top\rangle) = \{\langle \mathit{think}, \mathit{think}, n\rangle \mid n \in \mathbb{N} \setminus \{0\}\}$, call this X. It is not di cult to see that under the concrete transition relation induced by the program, every state $\langle \mathit{think}, \mathit{think}, n\rangle$ in X has 2 successors: if n is odd, then these successors are $\langle \mathit{eat}, \mathit{think}, n\rangle$ and $\langle \mathit{think}, \mathit{think}, 100\rangle$; if n is even, they are $\langle \mathit{think}, \mathit{eat}, n\rangle$ and $\langle \mathit{think}, \mathit{think}, 100\rangle$. So, the minimal sets Y of states such that $\mathrm{R}^{\exists\exists}(X, Y)$ holds are all the singletons $\{\langle \mathit{eat}, \mathit{think}, n\rangle\}$ with n odd, $\{\langle \mathit{think}, \mathit{eat}, n\rangle\}$ with n even, and $\{\langle \mathit{think}, \mathit{think}, 100\rangle\}$. Taking the abstractions (α) of each of these, results in the abstract states $\langle$eat, think, o$\rangle$, $\langle$think, eat, e$\rangle$ and $\langle$think, think, 100$\rangle$ being the ${}_\alpha\mathrm{R}^F$-successors of $\langle$think, think, $\top\rangle$ — di erent from the computed free successors above.

Let α^F be the function mapping a concrete transition relation R to the free abstract transition relation ${}_\alpha\mathrm{R}^F$: $\alpha^F(\mathrm{R}) = {}_\alpha\mathrm{R}^F$. Then the reason that precision is lost in the example above is that we do not apply α^F to R as a whole, but rather we attempt to compute it compositionally, by breaking it down over the individual operations c_i and t_i that appear in R's definition, as specified by $\widehat{{}_\alpha\mathrm{R}^F}$. Consequently, the information about n being odd or even prior to the execution of the first resp. third action is forfeited because the interpretations ${c_i}^F$ of the conditions are separated from those (${t_i}^F$) of the transformations.

On the other hand, although the computed abstraction ${}_\alpha\mathcal{I}(P)$ is not the same as $\alpha^M(\mathcal{I}(P))$, it is intuitively "safe". What we mean is that for every (${}_\alpha\mathrm{R}^F$-)successor of $\langle$think, think, $\top\rangle$ in $\alpha^M(\mathcal{I}(P))$, there is a less (or equally) precise (i.e. $\preceq$-greater) successor in the system ${}_\alpha\mathcal{I}(P)$, namely the state $\langle$eat, think, o$\rangle$ is "covered" by $\langle$eat, think, $\top\rangle$, state $\langle$think, eat, e$\rangle$ by $\langle$think, eat, $\top\rangle$, and $\langle$think, think, 100$\rangle$ is covered by $\langle$think, think, 100$\rangle$. In the rest of this section, we formalise this intuition by introducing a notion of approximation between Abstract Kripke structures. This is done in Definition 4.4.0.1. In Theorem 4.4.0.2 we show that approximations are safe with respect to the properties we are interested in: any CTL* property that holds in an approximation $\mathcal{A}''$ of $\mathcal{A}'$ will hold in $\mathcal{A}'$ as well[16]. Finally, Lemma 4.4.1.1

[16]Note that this notion is similar to the notion of safety of a description with regard to a concrete

and Theorem 4.4.1.2 show that the computed abstract model of Section 4.3 is indeed an approximation to the one as defined in Section 4.2. In particular, the proof of Lemma 4.4.1.1 shows what happens when α^M is pushed down the expression $\mathcal{I}(P)$. In Section 4.5 we return to that point when we want to identify conditions under which no loss of information[17] occurs when pushing down α^M. The dining mathematicians example is continued in Subsection 4.4.2 below.

For the approximation relation between abstract transition systems we use the same symbol $\preceq$ that already denotes approximation between individual abstract states — we will see to it that this overloading will not lead to confusion.

4.4.0.1 D *Let $\mathcal{A}' = ({}_\alpha\Sigma, F', C', I')$ and $\mathcal{A}'' = ({}_\alpha\Sigma, F'', C'', I'')$ be abstract transition systems. $\mathcal{A}' \preceq \mathcal{A}''$ i each of the following holds.*

1. *$F' \preceq$-pseudo-simulates[18] F''.*
2. *$C'' \succeq$-pseudo-simulates C'.*
3. *For every $a \in I'$, there exists $a'' \in I''$ such that $a' \preceq a''$.*

For paths $\bar{a} = a_0 a_1 \cdots$ and $\bar{a}' = a'_0 a'_1 \cdots$, $\bar{a} \preceq \bar{a}'$ is defined as $\forall_{i \geq 0}\ a_i \preceq a'_i$.

Note that $\mathcal{A}'$ and $\mathcal{A}''$ have the same state sets ${}_\alpha\Sigma$, over which the approximation relation $\preceq$ is fixed. The condition that $F' \preceq$-pseudo-simulates F'' means that for any two states $a', a'' \in {}_\alpha\Sigma$ with $a' \preceq a''$, if a' can make an F'-transition to some b', then a'' should be able to make an F''-transition to some b'' with $b'' \succeq b'$. On the other hand, $C'' \succeq$-pseudo-simulates C' means that for any two states $a', a'' \in {}_\alpha\Sigma$ with $a'' \succeq a'$, if a'' can make a C''-transition to some b'', then a' should be able to make a C'-transition to some b' with $b' \preceq b''$. Hence, Definition 4.4.0.1 imposes conditions on the transition relations that capture the intuition of "covering" explained above.

The relation $\preceq$ over abstract systems is a pre-order but not a partial order. We turn it into a partial order by identifying systems $\mathcal{A}'$ and $\mathcal{A}''$ whenever both $\mathcal{A}' \preceq \mathcal{A}''$ and $\mathcal{A}'' \preceq \mathcal{A}'$ hold. For example, the Abstract Kripke structure of Figure 4.1 (which only contains constrained transitions) is thus identified with the Abstract Kripke structure obtained from it by removing the transition from a to b_2. To improve readability, we do not explicitly distinguish between equivalence classes and representants.

---

object as defined in Section 3.2.1, page 35.

[17] We mean loss relative to $\alpha^M(\mathcal{I}(P))$. Of course, the fact that we apply abstraction unavoidably brings about a loss of information.

[18] Pseudo-simulation, defined in Definition 2.4.2.1, does not take into account the labelling of states with literals.

4.4.0.2 T *If $\mathcal{A}' \preceq \mathcal{A}''$, then for every $\varphi \in \mathrm{CTL}^*$, $\mathcal{A}'' \models \varphi \;\Rightarrow\; \mathcal{A}' \models \varphi$.*

P It is easily seen that if $\mathcal{A}' \preceq \mathcal{A}''$, then for every free $(\mathcal{A}', a)$-path $\bar{a}'$ there is a free $(\mathcal{A}'', a)$-path $\bar{a}'' \succeq \bar{a}'$, and for every constrained $(\mathcal{A}'', a)$-path $\bar{a}''$ there is a constrained $(\mathcal{A}', a)$-path $\bar{a}' \preceq \bar{a}''$ (for every $a \in {}_{\alpha}\Sigma$). Using these observations instead of Lemmata 4.2.3.2 and 4.2.3.5, and in addition Lemma 4.2.1.2, the rest of the proof is similar to that of Theorem 4.2.3.8. □

As an immediate corollary of this theorem and Theorem 4.2.3.8, we have that if $\mathcal{A}'' \succeq \alpha^M(\mathcal{C})$, then CTL* is preserved from $\mathcal{A}''$ to $\mathcal{C}$. Conversely, it is not true that any abstract system that preserves CTL* is an approximation to $\alpha^M(\mathcal{C})$. The conditions under which CTL* is preserved may alternatively be based solely on the existence of simulation relations, without requiring the existence of a Galois insertion between (sets of) concrete states and (single) abstract states. We defer a comparison of our framework with such *simulation-based* approaches, as we call them, to Section 4.9.1.

We should now briefly pause to explain how the concepts that we have defined so far, in particular the function α^M and the order $\preceq$ between Abstract Kripke structures, fit into the Abstract Interpretation frameworks discussed in Chapter 3. In that chapter, a central role is played by the notion of a description relation ρ. Whenever a concrete object c is described by an abstract object a, expressed by the fact $\rho(c, a)$, then a may safely be used to derive information about c (see condition 3.3 on page 36). An approximation ordering $\preceq$ between abstract objects is assumed such that $\preceq$-larger objects satisfy fewer properties and are hence less precise. For a given c, the $\preceq$-minimal descriptions of c are called *optimal* (Definition 3.2.1.1). In the case that for every c, an $\preceq$-*least* description exists, ρ may alternatively be represented by an abstraction function α that maps every c to its most precise description.

In the current chapter, we have first defined (in Section 4.2) an abstraction function α^M, and only in the current section have we defined an approximation order between abstract objects (being Abstract Kripke structures). Together, these induce a description relation[19] $\xi \subseteq \mathcal{KS} \times {}_{\alpha}\mathcal{KS}$, namely $\xi(\mathcal{C}, \mathcal{A}) \;\Leftrightarrow\; \alpha^M(\mathcal{C}) \preceq \mathcal{A}$. α^M is then precisely the abstraction function induced by ξ, and $\alpha^M(\mathcal{C})$ is by definition optimal[20].

One may wonder whether, among the Abstract Kripke structures with state set ${}_{\alpha}\Sigma$, $\alpha^M(\mathcal{C})$ is indeed the structure that enjoys the greatest number of properties,

[19] This description relation on the level of Kripke structures should not be confused with the description relation ρ between states (see Section 4.2).

[20] Because of the implicit quotient construction that is needed to turn $\preceq$ as defined above into a partial order, abstract systems $\mathcal{A}'$ for which both $\mathcal{A}' \preceq \alpha^M(\mathcal{C})$ and $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$ hold, are also optimal.

relative to all such Abstract Kripke structures that are safe for $\mathcal{C}$. Although answering this question is outside the scope of this thesis, we conjecture that this does hold under additional conditions, namely that (1) the description relation $\rho \subseteq \Sigma \times {}_{\alpha}\Sigma$ between states (see Section 4.2) coincides with the safety relation for literals, defined by $\forall_{p \in \mathrm{Lit}}\ c \models p \iff a \models p$ (cf. the footnote on page 36), and (2) Kripke structures are image-finite (page 21). A similar result is established in a slightly di erent framework in [CIY95], based on a result from [Lar89]. The first condition above is similar to the requirement of *precision* of [CIY95] (Definition 20). For a further discussion see Section 4.9 (page 102).

The following lemma, which is easily proven, gives su cient conditions for $\preceq$- and $\succeq$-pseudo-simulation. It will be used in the following sections.

4.4.0.3 L *Let R' and R'' be transition relations over ${}_{\alpha}\Sigma$.*

1. *If for all $a, b \in {}_{\alpha}\Sigma$,*

$$R'(a,b) \Rightarrow \exists_{b' \succeq b}\ R''(a,b') \tag{4.7}$$

and $R'' \preceq$-pseudo-simulates[21] R'', then $R' \preceq$-pseudo-simulates R''.

2. *If for all $a, b' \in {}_{\alpha}\Sigma$*

$$R''(a,b') \Rightarrow \exists_{b \preceq b'}\ R'(a,b) \tag{4.8}$$

and $R' \succeq$-pseudo-simulates R', then $R'' \succeq$-pseudo-simulates R'.

The intuition of condition 4.7 is that an approximation R'' of R' may relate an element a to a less precise state (namely b') than R' does. In the case of condition 4.8, on the other hand, R'' may only do so if there is a "reason". Note that 4.7 is satisfied if $\mathrm{R}' \subseteq \mathrm{R}''$ and 4.8 if $\mathrm{R}'' \subseteq \mathrm{R}'$.

4.4.1 Abstract interpretation gives approximations

The following lemma and theorem express that the abstract interpretations defined in Section 4.3 can be used to compute approximations to $\alpha^M(\mathcal{C})$.

4.4.1.1 L

1. ${}_{\alpha}\widehat{\mathrm{R}^F} \supseteq {}_{\alpha}\mathrm{R}^F$.

[21] Refer to Definition 2.4.2.1 and note that this is not a trivial requirement.

2. *For all* $a, b \in {}_\alpha\Sigma$, ${}_\alpha\widehat{\mathrm{R}^C}(a,b) \Rightarrow \exists_{b'' \preceq b}\ {}_\alpha\mathrm{R}^C(a,b'')$.

While the difference between the free and constrained cases in the lemmata and definitions so far has been based on exchanging (some) $\forall$s and $\exists$s, this symmetry is disturbed in the above lemma. The transition relation of a program is a disjunction of conjunctions: $\exists_{i\in J}\ c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{v}')$ (see Definition 4.3.0.1). Distribution of quantifier pairs $\exists\exists$ and $\forall\exists$ over such an expression cannot be performed without loosing symmetry, as can be seen in the proof below. However, after applying Lemma 4.4.0.3 to the statements of the above lemma, as is done in the proof of Theorem 4.4.1.2 below, symmetry is restored: it then turns out that ${}_\alpha\mathrm{R}^F$ $\preceq$-pseudo-simulates ${}_\alpha\widehat{\mathrm{R}^F}$ and ${}_\alpha\widehat{\mathrm{R}^C}$ $\succeq$-pseudo-simulates ${}_\alpha\mathrm{R}^C$.

P L 4.4.1.1.

1. Let $a, b \in {}_\alpha\mathrm{Val}$ and suppose $(a,b) \in {}_\alpha\mathrm{R}^F$. By Definition 4.2.3.3 of ${}_\alpha\mathrm{R}^F$, Definition 2.1.0.1 of $R^{\exists\exists}$ and Definition 4.3.0.1 of R, this is equivalent to

$$b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [\exists_{i\in J}\ [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}\}. \tag{4.9}$$

Exchanging existential quantifiers yields the equivalent

$$b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{i\in J}\ [\exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}\}. \tag{4.10}$$

Consider the subterm

$$Y \in \min\{Y' \mid \exists_{i\in J}\ [\exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}. \tag{4.11}$$

Because the elements of the set $\min\{Y' \mid \dots\}$ are singletons (see Observation 4.2.3.4), this subterm can be rewritten to the equivalent

$$\exists_{i\in J}\ [Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}]. \tag{4.12}$$

After replacing 4.11 by 4.12 in 4.10, we can bring the $\exists_{i\in J}$ outside, resulting in the equivalent formula

$$\exists_{i\in J}\ [b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}\}]. \tag{4.13}$$

Now this is *weakened* by distributing the innermost existential quantifier over the $\wedge$:

$$\exists_{i\in J}\ [b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a)}\ [c_i(\bar{v})] \wedge \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [t_i(\bar{v},\bar{w})]]\}\}]. \tag{4.14}$$

Because both the innermost and outermost sets do not depend on $\exists_{\bar{v}\in\gamma(a)}\ [c_i(\bar{v})]$, this conjunct may be taken out of all the set brackets. Using Definition 2.1.0.1 of ${t_i}^{\exists\exists}$ and Definition 4.3.0.2 of ${c_i}^F$, ${t_i}^F$, and ${}_\alpha\widehat{\mathrm{R}^F}$, the resulting equivalent term can then be rewritten to $(a,b) \in {}_\alpha\widehat{\mathrm{R}^F}$.

2. Let $a, b \in {}_\alpha\mathrm{Val}$ and suppose $(a, b) \in \widehat{{}_\alpha\mathrm{R}^C}$. By Definition 4.3.0.2 of $\widehat{{}_\alpha\mathrm{R}^C}$, ${c_i}^C$ and ${t_i}^C$, and Definition 2.1.0.1 of ${t_i}^{\forall\exists}$, this is equivalent to

$$\exists_{i\in J}\,[\forall_{\bar{v}\in\gamma(a)}\,[c_i(\bar{v})]\ \wedge\ b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\,\exists_{\bar{w}\in Y'}\,[t_i(\bar{v},\bar{w})]\}\}]. \tag{4.15}$$

This expression can be rewritten to the equivalent:

$$b \in \{\alpha(Y) \mid \exists_{i\in J}\,[Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\,\exists_{\bar{w}\in Y'}\,[c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}]\} \tag{4.16}$$

(define $\min\emptyset = \emptyset$ in this proof). Now consider the subexpression

$$\exists_{i\in J}\,[Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\,\exists_{\bar{w}\in Y'}\,[c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}]. \tag{4.17}$$

Compare this to the expression that is obtained by pushing the $\exists_{i\in J}$ inside:

$$Y \in \min\{Y' \mid \exists_{i\in J}\,[\forall_{\bar{v}\in\gamma(a)}\,\exists_{\bar{w}\in Y'}\,[c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}. \tag{4.18}$$

If Y satisfies 4.17, then there exists an $i \in J$ such that Y is minimal among all "$\forall\exists$-successors" of $\gamma(a)$ that correspond to action i. On the other hand, if Y satisfies 4.18, then Y is minimal among *all* the $\forall\exists$-successors of $\gamma(a)$, regardless of the specific i. Hence, this latter Y will be a subset of (or possibly equal to) the Y that satisfies 4.17. So, for each set that satisfies 4.17, there exists a subset of it that satisfies 4.18, so that if b satisfies 4.16, there exists $b' \preceq b$ that satisfies:

$$b' \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{i\in J}\,[\forall_{\bar{v}\in\gamma(a)}\,\exists_{\bar{w}\in Y'}\,[c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}\}. \tag{4.19}$$

A similar step can be made again: if b' satisfies 4.19, then there exists $b'' \preceq b'$ satisfying:

$$b'' \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\,\exists_{\bar{w}\in Y'}\,\exists_{i\in J}\,[c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}\} \tag{4.20}$$

which is, by Definition 4.3.0.1 of R, Definition 2.1.0.1 of $R^{\forall\exists}$ and Definition 4.2.3.1 of ${}_\alpha\mathrm{R}^C$, equivalent to ${}_\alpha\mathrm{R}^C(a, b'')$. □

4.4.1.2 T *For every $P \in \pounds$, ${}_\alpha\mathcal{I}(P) \succeq \alpha(\mathcal{I}(P))$.*

P We write $\widehat{\mathcal{A}^M}$ for ${}_\alpha\mathcal{I}(P)$ and $\mathcal{A}^M$ for $\alpha(\mathcal{I}(P)) = \alpha(\mathcal{C})$. We have to show points 1, 2 and 3 as in Definition 4.4.0.1. Point 3 is immediate as the initial states of $\widehat{\mathcal{A}^M}$ and $\mathcal{A}^M$ are identical. As to points 1 and 2, it easily follows from the definitions of $\widehat{{}_\alpha\mathrm{R}^F}$ and ${}_\alpha\mathrm{R}^C$ that $\widehat{{}_\alpha\mathrm{R}^F}$ $\preceq$-pseudo-simulates $\widehat{{}_\alpha\mathrm{R}^F}$ and that ${}_\alpha\mathrm{R}^C$ $\succeq$-pseudo-simulates ${}_\alpha\mathrm{R}^C$. Hence, by Lemmata 4.4.1.1 and 4.4.0.3, ${}_\alpha\mathrm{R}^F$ $\preceq$-pseudo-simulates $\widehat{{}_\alpha\mathrm{R}^F}$ and $\widehat{{}_\alpha\mathrm{R}^C}$ $\succeq$-pseudo-simulates ${}_\alpha\mathrm{R}^C$. □

4.4.1.3 C *For every $P \in \pounds$ and $\varphi \in$ CTL*, ${}_\alpha\mathcal{I}(P) \models \varphi \Rightarrow \mathcal{I}(P) \models \varphi$.*

P From Theorems 4.4.1.2, 4.4.0.2 and 4.2.3.8. □

4.4.2 Dining mathematicians continued (I)

As was summarised in Corollary 4.4.1.3 above, we have formally established all the machinery that justifies the use of the Abstract Kripke structures that were constructed in Section 4.3.1 to infer properties of the original, concrete system.

We see in Figure 4.6 that in no reachable state the property $\ell_0 = \mathsf{eat} \wedge \ell_1 = \mathsf{eat}$ holds. Hence we have established property 4.3 (page 71). Furthermore, every path from the state where $\ell_0 = \mathsf{eat}$, reaches $\ell_1 = \mathsf{eat}$ within 2 steps, so we have also verified property 4.4.

However, the abstraction does not allow verification of the other non-starvation property, 4.5: a counter-example in the abstract model is the path cycling infinitely between $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$ and $\langle \mathsf{think}, \mathsf{eat}, \mathsf{e} \rangle$. It turns out that the negation of property 4.5 can also not be established via the constrained transition relation. So, only refinement of the abstract domain may bring the answer. In this case, the abstract states where $n = \mathsf{e}$ would have to be unraveled into infinitely many states representing the cases where n is divisible by 4, by 8, by 16, ... Hence, with our methodology, it is impossible to verify property 4.5 through a finite abstraction.

In Section 4.8 as well as in Chapter 5 we further investigate the question how the refinement of an abstract model, in order to decide indeterminate results, can be computed. The solution proposed in Section 4.8 is based on the idea of a "tunable" abstract domain, in which the refinement is determined by a parameter of the domain. In Chapter 5 we identify conditions under which a strongly preserving abstraction may be computed by successive refinement that is guided by the form of the formula to be checked. It should be said that in their present form, neither of these approaches indeed help in verifying the specific property 4.5, the main objection being that the underlying concrete system is infinite. However, on-going work concentrates on including fairness constraints in the abstract models. This research, which is further discussed in Section 7.3, has meanwhile yielded results that enable the verification of property 4.5 via a finite abstraction.

Property 4.6 is verified on the model of Figure 4.8, interpreting the universal quantification over the free paths, and the existential quantification over the constrained paths. It can easily be seen that 4.6 holds, hence, we have established its validity in the concrete program. Note that the presence of the state $\langle \mathsf{think}, \mathsf{think}, \top \rangle$, though it is not reachable via free transitions alone, is essential in proving the property.

4.5 Optimal Abstract Interpretations

Construction of an abstract model by abstract interpretation of the "elementary" operations (the c_i and t_i) occurring in a program is a natural thing to do — it resembles the way abstractions are computed in traditional applications of Abstract Interpretation. However, as we have seen, the computed abstract models (Definition 4.3.0.2) may be strictly less precise than the optimal abstractions of Definitions 4.2.3.1 and 4.2.3.3. How much precision is lost exactly, depends on the program to be analysed and the choice of the abstract domain. In order to get some insight, we discuss two approaches to obtain optimality. First, we derive su cient conditions on abstract states (and on the program) for the computed abstract transitions of Definition 4.3.0.2 to be optimal. Second, we briefly sketch how, alternatively, the abstract interpretation of programs may be adapted in such a way that computed models are optimal.

4.5.1 Conditions on the abstract domain

In order to pinpoint the reasons why the computed abstractions are not optimal, we analyse the proof of Lemma 4.4.1.1. In part 1, concerning the free abstraction, the only place where the formula being manipulated is (strictly) weakened, is when the term $\exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]$ (T1) is replaced by $\exists_{\bar{v}\in\gamma(a)}\ [c_i(\bar{v})] \wedge \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [t_i(\bar{v},\bar{w})]$ (T2). The following small example illustrates what happens. Suppose that the concrete state space consists of a single integer variable v, and that the abstract domain contains values e and o, being descriptions of the even and the odd numbers respectively. Assume that P contains as action i: $v = 4 \rightarrow v := v/4$ (specifying $c_i(v)$ to be $v = 4$ and $t_i(v,w)$ to be $w = v/4$). Then T1, with e for a and $\gamma(\mathsf{e})$ for Y', does not hold. On the other hand, T2 *does* hold: there exists an even number that is equal to 4 and there exists a (di erent) even number that, when divided by 4, yields an even number. In order to avoid this situation and enforce equivalence of T1 and T2, we can impose conditions on the abstract states. For an abstract state a, this condition intuitively requires that the concrete states in $\gamma(a)$ behave uniformly with respect to every condition c_i.

4.5.1.1 L *Let $\widehat{\mathcal{A}^M} = {}_\alpha\mathcal{I}(P)$ be the abstract model computed according to Definition 4.3.0.2, and let $a \in {}_\alpha\Sigma$. If for every $i \in J$, we have either $\forall_{\bar{v}\in\gamma(a)}\ c_i(\bar{v})$ or $\forall_{\bar{v}\in\gamma(a)}\ \neg c_i(\bar{v})$, then every outgoing $\widehat{{}_\alpha\mathrm{R}^F}$-transition of a is in ${}_\alpha\mathrm{R}^F$.*

P The precondition of the lemma is easily seen to imply equivalence of the terms T1 and T2 and hence of the formulae 4.13 and 4.14 in the proof of Lemma 4.4.1.1. The conclusion then follows directly. □

So, under the given condition, all outgoing free transitions of state a are optimal. Clearly, the condition is very strong for "$\preceq$-large" states. E.g. if $a = {}_{\alpha}\top$, then it is only satisfied if all program conditions c_i are either tautologies or unsatisfiable. However, as far as properties $\varphi \in \forall\text{CTL}^*$ are concerned, it is su cient to require the condition to hold only for those states on which φ depends. In that case, the result of model checking φ over the computed model $\widehat{\mathcal{A}^M}$ will be the same as when checking it over the optimal model $\mathcal{A}^M$. As a consequence, unreachable states may be ignored alltogether. As to the reachable states, observe that only the *atoms* of the abstract domain, i.e. the elements $\{\alpha(\{c\}) \mid c \in \Sigma\}$, can be reachable via a free transition. This follows from Observation 4.2.3.4. Hence, we should preferably choose the abstract domain in such a way that these atoms are $\preceq$-small[22]. However, if φ is a subformula of a formula that contains existential path quantifiers, then also certain states that are reachable via constrained transitions may have to satisfy the condition of Lemma 4.5.1.1.

Although su cient, the condition required in Lemma 4.5.1.1 is not necessary. However, it is a reasonable condition that can be checked rather easily: one has to check that for each atomic abstract state a and each condition c_i of the program, either "$a \Rightarrow c_i$" or "$a \cap c_i = \emptyset$". For instance, in the example above, "being even" neither implies nor excludes "being equal to 4", so the condition is not met. The condition also gives a deeper insight in how to design "good" abstract domains given a program(ming language).

For the constrained relation, we analyse part 2 of the proof of Lemma 4.4.1.1. The last two steps in this proof introduce the di erences between $\widehat{{}_{\alpha}\mathrm{R}^C}$ and ${}_{\alpha}\mathrm{R}^C$. We consider these steps in reverse direction, going from ${}_{\alpha}\mathrm{R}^C$ to $\widehat{{}_{\alpha}\mathrm{R}^C}$. While in formula 4.20 the "$\forall\exists$-successors" Y' of $\gamma(a)$ are taken relative to transitions via *any action* (i.e. all states in $\gamma(a)$ must be able to make a transition to some state in Y' via no matter which action i), the $\forall\exists$-successors Y' of $\gamma(a)$ in 4.19 are taken "per action", i.e. for a single action $i \in J$, all states in $\gamma(a)$ must be able to make a transition to some state in Y' *via action* i. This means that in the latter case, certain $\forall\exists$-successors Y' may be "missed" and consequently, $\widehat{{}_{\alpha}\mathrm{R}^C}$ may contain fewer transitions than ${}_{\alpha}\mathrm{R}^C$. However, note that if for such a transition, say from a to b, which is in ${}_{\alpha}\mathrm{R}^C$ but not in $\widehat{{}_{\alpha}\mathrm{R}^C}$, there exists another transition in $\widehat{{}_{\alpha}\mathrm{R}^C}$ from a to a more precise state $b' \preceq b$, this loss does not matter: $\widehat{{}_{\alpha}\mathrm{R}^C}$ will not be less precise (in the sense of Definition 4.4.0.1) than ${}_{\alpha}\mathrm{R}^C$ because of this. It is this observation on which the condition in Lemma 4.5.1.2 below is based.

Now consider the step from 4.19 to 4.16 — more precisely, the replacement of subformula 4.18 by 4.17. In 4.18, the minimality of Y is determined globally over

[22] See [CC79] for a variety of techniques for the construction of suitable abstract domains.

all actions, while in 4.17 all Ys that are minimal relative to a single action $i \in J$ are taken. As a result, the set of successors under $\widehat{{}_{\alpha}\mathrm{R}^{C}}$ of some abstract state a may be a superset of its successors under ${}_{\alpha}\mathrm{R}^{C}$ (the fact that $\widehat{{}_{\alpha}\mathrm{R}^{C}}$ is not strictly $\preceq$-below ${}_{\alpha}\mathrm{R}^{C}$ is explained by observing that for each such extra successor b' under $\widehat{{}_{\alpha}\mathrm{R}^{C}}$ there will be a more precise successor $b \preceq b'$ under ${}_{\alpha}\mathrm{R}^{C}$). Hence, this e ect does not negatively a ect the precision of $\widehat{{}_{\alpha}\mathrm{R}^{C}}$ with respect to ${}_{\alpha}\mathrm{R}^{C}$.

4.5.1.2 L *Let $a \in {}_{\alpha}\Sigma$ and suppose that both of the following conditions hold:*

1. *For every $i \in J$, $\forall_{\bar{v}\in\gamma(a)}\ c_i(\bar{v})$ or $\forall_{\bar{v}\in\gamma(a)}\ \neg c_i(\bar{v})$.*
2. *For all $i, j \in J$ with $i \neq j$ and $b_i, b_j \in {}_{\alpha}\Sigma$ with ${c_i}^{C}(a) \wedge {t_i}^{C}(a, b_i)$ and ${c_j}^{C}(a) \wedge {t_j}^{C}(a, b_j)$: if there exists $b \in {}_{\alpha}\Sigma$ with $\gamma(b_i) \cap \gamma(b) \neq \emptyset$ and $\gamma(b_j) \cap \gamma(b) \neq \emptyset$, then there exist $k \in J$ and $b_k \in {}_{\alpha}\Sigma$ with ${c_k}^{C}(a) \wedge {t_k}^{C}(a, b_k)$ such that $\gamma(b_k) \subseteq \gamma(b)$.*

Then for every $b' \in {}_{\alpha}\Sigma$, ${}_{\alpha}\mathrm{R}^{C}(a, b') \Rightarrow \exists_{b\preceq b'}\ \widehat{{}_{\alpha}\mathrm{R}^{C}}(a, b)$.

Note that condition 1 is similar to the condition in Lemma 4.5.1.1 above. Condition 2 specifies that two abstract successors b_i and b_j of a corresponding to di erent actions ($i \neq j$) may only both be (partially or completely) overlapped by a third state b if b completely overlaps some successor b_k (possibly $k = i$ or $k = j$) of a.

P It may be helpful to realise that we are, roughly speaking, trying to reverse the direction of the argument in point 2 of the proof of Lemma 4.4.1.1. Let $b' \in {}_{\alpha}\Sigma$ and assume ${}_{\alpha}\mathrm{R}^{C}(a, b')$. By Definition 4.2.3.1 of ${}_{\alpha}\mathrm{R}^{C}$, Definition 2.1.0.1 of $R^{\forall\exists}$ and Definition 4.3.0.1 of R, this is equivalent to saying that b' is an element of

$$\{\alpha(Y) \mid Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\ \exists_{\bar{w}\in Y'}\ \exists_{i\in J}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]\}\}. \tag{4.21}$$

Next, consider the set that is obtained by taking the $\exists_{i\in J}$ outside of the $\forall_{\bar{v}\in\gamma(a)}\exists_{\bar{w}\in Y'}$:

$$\{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{i\in J}\ [\forall_{\bar{v}\in\gamma(a)}\ \exists_{\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]]\}\}. \tag{4.22}$$

We consider two cases. If b' is an element of 4.22, then we proceed as follows. The subexpression

$$Y \in \min\{Y' \mid \exists_{i\in J}\ [\forall_{\bar{v}\in\gamma(a)}\ \exists_{\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]]\} \tag{4.23}$$

of 4.22 is *weakened* by bringing the $\exists_{i\in J}$ outside:

$$\exists_{i\in J}\ [Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\ \exists_{\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]\}]. \tag{4.24}$$

Therefore, b' is also an element of the set obtained by replacing subexpression 4.23 of 4.22 by 4.24, resulting in the set

$$\{\alpha(Y) \mid \exists_{i\in J}\ [Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\ \exists_{\bar{w}\in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]]\}\}. \tag{4.25}$$

Similar to the first two steps in point 2 of the proof of Lemma 4.4.1.1, but in reverse order, this implies that $(a, b') \in {}_{\alpha}\widehat{\mathrm{R}^C}$.

The other case is that b' is not in 4.22. Let $Z \subseteq \Sigma$ be such that $b' = \alpha(Z)$. Because b' is in 4.21 but not in 4.22, we can choose $i, j \in J$ with $i \neq j$, $\bar{v}_i, \bar{v}_j \in \gamma(a)$ and $\bar{w}_i, \bar{w}_j \in Z$ such that $c_i(\bar{v}_i) \wedge t_i(\bar{v}_i, \bar{w}_i)$ and $c_j(\bar{v}_j) \wedge t_j(\bar{v}_j, \bar{w}_j)$. Because, by condition 1 of the lemma, $\forall_{\bar{v} \in \gamma(a)}\ c_i(\bar{v})$ and $\forall_{\bar{v} \in \gamma(a)}\ c_j(\bar{v})$, we can also choose $Z_i, Z_j \subseteq \Sigma$ such that $\bar{w}_i \in Z_i$ and $\bar{w}_j \in Z_j$. Hence, $\alpha(Z)$ must have a non-empty intersection with both $\alpha(Z_i)$ and with $\alpha(Z_j)$. The precondition of the lemma then requires that there exist $k \in J$ and $b_k \in {}_{\alpha}\Sigma$ with ${c_k}^C(a) \wedge {t_k}^C(a, b_k)$ (and therefore $(a, b_k) \in {}_{\alpha}\widehat{\mathrm{R}^C}$) such that $b_k \preceq \alpha(Z)$, i.e. $b_k \preceq b'$. □

Again, for a formula $\varphi \in \exists\mathrm{CTL}^*$ being checked, it su ces to impose the conditions of this lemma only on those states on which the interpretation of φ depends. If φ is in full CTL*, then the condition of Lemma 4.5.1.1 should hold in those states on whose outgoing free transitions φ depends and the conditions of Lemma 4.5.1.2 should hold in those states on whose outgoing constrained transitions φ depends. In that case, model checking φ over the computed mixed abstraction $\widehat{\mathcal{A}^M}$ of Definition 4.3.0.2 gives the same result as checking it over the optimal $\mathcal{A}^M$.

Dining mathematicians continued (II)

We illustrate the application of the above lemmata to the abstract models that were constructed in Section 4.3.1. It is easy to check that for each condition c_i of the action system in Figure 4.4 and each abstract state a of the system of Figure 4.6, either c_i evaluates to true in all concrete states in $\gamma(a)$, or it evaluates to false in all concrete states in $\gamma(a)$. This implies by Lemma 4.5.1.1 that (the reachable part of) the computed free abstraction coincides with (the reachable part of) the optimal free abstraction as defined in Definition 4.2.3.3.

In order to check optimality of transitions in the model of Figure 4.8, we verify the preconditions of Lemma 4.5.1.2 for its states. Condition 1 holds for all states but $\langle \mathsf{think}, \mathsf{think}, \top \rangle$. As to condition 2, the only states that have two different (constrained) successors (corresponding to the b_i and b_j in Lemma 4.5.1.2) are $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$, $\langle \mathsf{think}, \mathsf{think}, \mathsf{o} \rangle$ and $\langle \mathsf{think}, \mathsf{think}, 100 \rangle$. The successors of $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$ are $\langle \mathsf{think}, \mathsf{eat}, \mathsf{e} \rangle$ and $\langle \mathsf{think}, \mathsf{think}, 100 \rangle$. The states in ${}_{\alpha}\Sigma$ (corresponding to the b in Lemma 4.5.1.2) that overlap both of them are $\langle \mathsf{think}, \top, 100 \rangle$ and all states that are $\preceq$-greater than it. Because $\langle \mathsf{think}, \top, 100 \rangle$ completely overlaps a successor of $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$, namely $\langle \mathsf{think}, \mathsf{think}, 100 \rangle$ (b_k in Lemma 4.5.1.2), all $\preceq$-greater states also overlap this successor. In a similar way, condition 2 of Lemma 4.5.1.2 can be shown to hold for $\langle \mathsf{think}, \mathsf{think}, \mathsf{o} \rangle$ and $\langle \mathsf{think}, \mathsf{think}, 100 \rangle$ too. The conclusion is that only the constrained transition that starts in the state

$\langle \mathsf{think}, \mathsf{think}, \top \rangle$ may be non-optimal[23].

4.5.2 Adapting the abstract interpretation

Instead of imposing conditions guaranteeing optimality of abstract models computed as specified by Definition 4.3.0.2, we may change the definition of these abstract interpretations themselves in such a way that the "loss" of Lemma 4.4.1.1 does not occur. For the free abstract interpretation, this means that it cannot be distributed over the individual condition and transformation parts of an action. In the case of the example given above, this would mean that an abstract interpretation ${act_i}^F$ has to be provided for the action $act_i(v, w) \Leftrightarrow v = 4 \wedge w = v/4$ as a whole, satisfying ${act_i}^F(a, b) \Leftrightarrow \exists_{v \in \gamma(a)} [c_i(v) \wedge b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid {t_i}^{\exists\exists}(\{v\}, Y')\}\}]$.

In the case of the constrained abstract interpretations, loss of optimality already occurs at the point where they are distributed over the individual actions of a program. Here, the adaptation would require the generalisation of the abstract interpretation of actions by taking into account the e ect of executing an arbitrary number of actions "at the same time" by defining $act_{\{i_1,\ldots,i_k\}}(a, b) \Leftrightarrow \exists_{a_1,\ldots,a_k,b_1,\ldots,b_k \in {}_\alpha\Sigma}$ $[a_1 \vee \cdots \vee a_k = a, b_1 \vee \cdots \vee b_k = b, \forall_{j \in \{1,\cdots,k\}} {c_{i_j}}^C(a_j) \wedge {t_{i_j}}^C(a_j, b_j)]$ for subsets $\{i_1, \ldots, i_k\}$ of J ($\vee$ denotes the least upper bound on ${}_\alpha\Sigma$). This approach corresponds to the *merge over all paths analysis* of [CC79].

4.6 Computing Approximations

One may deliberately choose to compute non-optimal abstractions by specifying approximations to the abstract interpretations of the c_i and t_i. A reason for doing so may be that the computation of optimal abstract interpretations is too complex, when the c_i and t_i involve intricate operations for example. In that case, even if the abstract interpretations are optimal, it may be cumbersome to actually prove so, and one may settle for proving approximation without bothering about optimality.

4.6.0.1 D *The definition of approximation is extended to abstract interpretations of the transformation operators, as follows. For abstract operations*[24] $t, \overline{t} \in {}_\alpha\mathrm{Val} \times {}_\alpha\mathrm{Val}$,

$$\overline{t} \succeq t \;\Leftrightarrow\; \forall_{a,b,\bar{b} \in {}_\alpha\mathrm{Val}} \left[t(a, b) \Rightarrow \exists_{\bar{b} \succeq b}\, \overline{t}(a, \bar{b})\right] \wedge \left[\overline{t}(a, \bar{b}) \Rightarrow \exists_{b \preceq \bar{b}}\, t(a, b)\right].$$

Approximations $\overline{{t_i}^F} \succeq {t_i}^F$ *and* $\overline{{t_i}^C} \succeq {t_i}^C$ *(for every* $i \in J$*) to the free and con-*

[23] In fact, it *is* optimal, as is easily seen. This indicates that Lemma 4.5.1.2 only gives a *su cient* condition.

[24] Remember that such "operations" are binary relations.

strained interpretations (see Definition 4.3.0.2) induce the abstract model $\overline{\mathcal{A}^M} = ({}_\alpha\Sigma, \overline{{}_\alpha\mathrm{R}^F}, \overline{{}_\alpha\mathrm{R}^C}, {}_\alpha\mathrm{I})$*, where:*

- ${}_\alpha\Sigma = {}_\alpha\mathrm{Val}$.
- $\overline{{}_\alpha\mathrm{R}^F} = \{(a,b) \in {}_\alpha\mathrm{Val}^2 \mid \exists_{i\in J}\ {c_i}^F(a) \wedge \overline{{t_i}^F}(a,b)\}$.
- $\overline{{}_\alpha\mathrm{R}^C} = \{(a,b) \in {}_\alpha\mathrm{Val}^2 \mid \exists_{i\in J}\ {c_i}^C(a) \wedge \overline{{t_i}^C}(a,b)\}$.
- ${}_\alpha\mathrm{I} = \{\alpha(\bar{v}) \mid \bar{v} \in \mathrm{IVal}\}$.

4.6.0.2 L $\overline{\mathcal{A}^M} \succeq {}_\alpha\mathcal{I}(P)$.

P . Write $\widehat{\mathcal{A}^M}$ for ${}_\alpha\mathcal{I}(P)$. We show that (1) $\widehat{{}_\alpha\mathrm{R}^F}$ $\preceq$-pseudo-simulates $\overline{{}_\alpha\mathrm{R}^F}$ and (2) $\overline{{}_\alpha\mathrm{R}^C}$ $\succeq$-pseudo-simulates $\widehat{{}_\alpha\mathrm{R}^C}$.

1. Let $a, a_1, a' \in {}_\alpha\Sigma$ with $\widehat{{}_\alpha\mathrm{R}^F}(a,a_1)$ and $a' \succeq a$. We show that there exists $a'_1 \succeq a_1$ such that $\overline{{}_\alpha\mathrm{R}^F}(a',a'_1)$. By Definition 4.3.0.2 of $\widehat{{}_\alpha\mathrm{R}^F}$, $\widehat{{}_\alpha\mathrm{R}^F}(a,a_1)$ is equivalent to $\exists_{i\in J}[{c_i}^F(a) \wedge {t_i}^F(a,a_1)]$. Because $a' \succeq a$, we have ${c_i}^F(a) \Rightarrow {c_i}^F(a')$ and also ${t_i}^F(a,a_1) \Rightarrow {t_i}^F(a',a_1)$ (see Definition 4.3.0.2 of ${c_i}^F$ and ${t_i}^F$ and Definition 2.1.0.1 of $\cdot^{\exists\exists}$), so $\exists_{i\in J}\ [{c_i}^F(a') \wedge {t_i}^F(a',a_1)]$. By Definition 4.6.0.1 of $\overline{{t_i}^F}$, there exists $a'_1 \succeq a_1$ such that $\exists_{i\in J}\ [{c_i}^F(a') \wedge \overline{{t_i}^F}(a',a'_1)]$, i.e. $\overline{{}_\alpha\mathrm{R}^F}(a',a'_1)$.

2. Let $a, a_1, a' \in {}_\alpha\Sigma$ with $\overline{{}_\alpha\mathrm{R}^C}(a,a_1)$ and $a' \preceq a$. We show that there exists $a'_1 \preceq a_1$ such that $\widehat{{}_\alpha\mathrm{R}^C}(a',a'_1)$. We have $\overline{{}_\alpha\mathrm{R}^C}(a,a_1)$. By Definition 4.6.0.1 of $\overline{{}_\alpha\mathrm{R}^C}$ and $\overline{{t_i}^C}$, there exists $a''_1 \preceq a_1$ such that $\exists_{i\in J}\ [{c_i}^C(a) \wedge {t_i}^C(a,a''_1)]$. Because $a' \preceq a$, we have ${c_i}^C(a) \Rightarrow {c_i}^C(a')$, and also we can choose $a'_1 \preceq a''_1$ such that ${t_i}^C(a',a'_1)$ (see Definition 4.3.0.2 of ${c_i}^C$ and ${t_i}^C$ and Definition 2.1.0.1 of $\cdot^{\forall\exists}$). So $\exists_{i\in J}\ [{c_i}^C(a') \wedge {t_i}^C(a',a'_1)]$, i.e. $\widehat{{}_\alpha\mathrm{R}^C}(a',a'_1)$, and, by transitivity of $\preceq$, $a'_1 \preceq a_1$. □

As an example of the computation of approximations by choosing non-optimal abstract interpretations $\overline{t_i}$ of operations in the program, consider the dining mathematicians without the "restart" extension. Take optimal free abstract interpretations of all operations except $3*$, for which we take the following approximation: $\overline{3*^F}(\mathsf{o},\top) = \mathit{true}$ and $\overline{3*^F}(\mathsf{o},\mathsf{e}) = \overline{3*^F}(\mathsf{o},\mathsf{o}) = \mathit{false}$. Furthermore, we take $\langle \mathsf{think}, \mathsf{think}, \top\rangle$ as the abstract initial state. This gives the free abstraction of Figure 4.9, from which still various properties may be deduced, such as the fact that at least one mathematician will keep engaged in a cycle of thinking and eating.

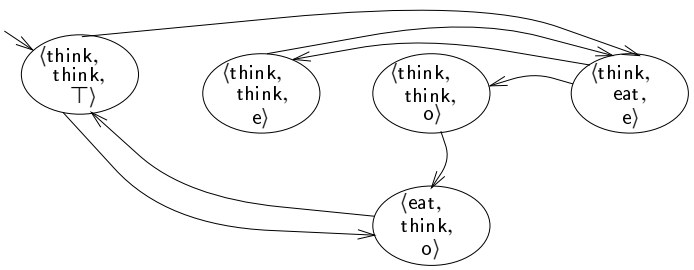


Figure 4.9: An approximation to the free abstraction.

4.7 Practical Application

This section discusses a few points relating to the practical application of the methodology developed so far.

The use of abstract interpretation to model check a property φ for a program P is characterised by the following phases. First, an abstract domain ${}_{\alpha}\mathrm{Val}$ has to be chosen and for all operation symbols occurring in P, abstract interpretations have to be provided. Typically, the tests and transformations are defined in terms of more elementary operations, in which case abstract interpretations may be provided for these. Depending on the property φ to be checked, free and/or constrained interpretations should be given; these have to satisfy Definition 4.3.0.2. Then, the abstract model can be constructed by a symbolic evaluation of the program over the abstract domain, interpreting the operations according to their abstract interpretations. Finally, φ is model checked over the abstract model. It is important to notice that only *positive* results of this model checking carry over to the concrete model: a negative result $\mathcal{A} \not\models \varphi$ does *not* imply that $\mathcal{A} \models \neg\varphi$ and hence does not justify the conclusion that $\mathcal{C} \models \neg\varphi$, in spite of the fact that $\neg\varphi$ is (an abbreviation of) a CTL* formula. However, it may be possible to resolve such a negative answer for φ by checking the negation $\neg\varphi$. If *true* is returned, then we know that $\mathcal{C} \models \neg\varphi$, i.e. $\mathcal{C} \not\models \varphi$. As $\neg\varphi$ contains the dual[25] path quantifiers of those in φ, its satisfaction by the Abstract Kripke structure $\mathcal{A}$ may depend on di erent paths — in particular, it may not hold either. So, whether this "trick" to resolve negative answers is successful, depends on how the dual abstractions (in the sense of free vs. constrained) are chosen. We do not further investigate this point here; the interested reader is referred to [KDG95].

The same idea of constructing an abstract model by abstract interpretation of program operations, although based on a di erent theoretical framework ([LGS$^+$95,

[25] Recall (Definition 2.3.0.1) that $\neg\varphi$ is the abbreviation of a CTL* formula in negation-normal form.

Loi94], see Section 4.9 for a comparison), is applied to a "real-life" example in [Gra94]. Graf shows in that paper how a distributed cache memory, which is in principle an infinite state system because request queues are unbounded, can be verified by providing a finite abstract domain and corresponding abstract operations.

As a consequence of the interpretation of CTL*, which is defined over paths (which are infinite sequences), no properties about finite computations of a program can be expressed. One could argue that our interest is in verifying *reactive* systems, which by definition only exhibit infinite behaviour. But a verification methodology should not only enable the formal a rmation of properties of a correct reactive system, but also detect mistakes in incorrect systems. If a reactive system contains an unintentional deadlock, i.e. a reachable state without outgoing transitions, this will not be caught by our methodology: because of the interpretation over infinite computations only, it is not possible to express deadlock freedom. For example, the formula $\forall \mathsf{G} \exists \mathsf{X} \mathit{true}$ is valid, meaning it is equivalent to *true* in any Kripke structure.

This deficiency can be repaired by assuring that all computations of the concrete system are infinite, before checking any other properties. On the level of action systems, deadlock freedom may be ensured by extending the set Val with a "special" value *stop* and adding an extra action of the form $\left(\bigwedge_{i \in J} \neg c_i(\bar{x})\right) \vee \bar{x} = \mathit{stop} \rightarrow \bar{x} := \mathit{stop}$. On a more theoretical level this corresponds to adding a state *stop* to the Kripke structure and extending the transition relation by adding a transition leading to *stop* from every state that has no outgoing transitions, including *stop* itself. New literals *is_stop* and $\neg$*is_stop* are added to Lit and the valuation of other literals is extended to *stop*, in such a way that *is_stop* is the only literal that holds in *stop*, and $\neg$*is_stop* holds everywhere but in *stop*. Note that for the construction of abstract models, abstract interpretations for the extra condition $\left(\bigwedge_{i \in J} \neg c_i(\bar{x})\right) \vee \bar{x} = \mathit{stop}$ and transformation $\bar{x} := \mathit{stop}$ now have to be provided. Some care has to be taken when specifying safety properties with regard to the extended system. For example, the property that ψ holds along all states of the (finite or infinite) computation π in the original concrete system should be adapted for the *stop* extension and is expressed in CTL* as $\pi \models \mathsf{W}(\psi, \mathit{is_stop})$ (the Weak-Until operator W is defined on page 24).

If the existence of stopping states is not intended, in which case they are called deadlock states, one usually wants to verify that the program is "deadlock free". This means that no deadlock state is reachable from any of the initial states. Such an analysis may be performed within the framework, by checking the formula $\forall \mathsf{G} \neg \mathit{is_stop}$ over the program obtained by applying the transformation described above. If this succeeds, this implies that the original program is deadlock free. It is not di cult to see that the extra action may then be removed again from the program when verifying other properties via abstract interpretations, as long as these properties are evaluated

in the initial states of the abstract system.

One may wonder whether it is not possible instead to extend the definition of CTL*'s interpretation to finite computations, by interpreting the path quantifiers over *maximal* prefixes (cf. [DNV90b] for example), i.e. prefixes that cannot be extended because either they are infinite, or their last state has no outgoing transitions. The answer is that this would require a revision of the results developed in this chapter. The example in Figure 4.10 illustrates that the preservation results do not hold anymore in this case. Although $\forall \mathsf{X} p$ clearly holds in abstract state a, it does not hold in the concrete state c in $\gamma(a)$: the set of all maximal c-prefixes only contains the empty prefix ϵ. However, we have $\epsilon \not\models \mathsf{X} p$, because the next-state operator X requires the existence of a next state — one might say that it has an "existential character". A similar problem occurs with the Until operator U, which is not surprising once we realise that U may be viewed as being defined in terms of X: $\mathsf{U}(\psi_1, \psi_2)$ is equivalent to the weakest predicate z for which $z \equiv \psi_2 \vee (\psi_1 \wedge \mathsf{X} z)$. This

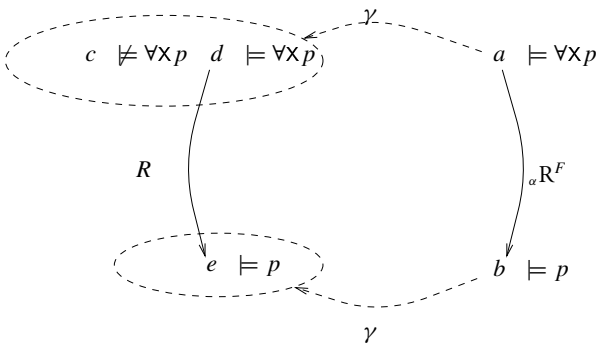


Figure 4.10: When CTL* is interpreted over maximal prefixes.

counter-example indeed suggests that for the interpretation over maximal prefixes, the distinction between the universal and existential fragments of CTL* has to be refined. Technically, the point is that the next-state operator X is not its own dual when it is interpreted over possibly finite prefixes, i.e. $\neg \mathsf{X} \varphi$ is not equivalent to $\mathsf{X} \neg \varphi$. A solution would be to introduce the dual operator, say X', of X; this dual operator would then have a "universal character". However, such a solution would complicate the definition of CTL*. These complications are avoided when we take the μ-calculus (L_μ) as starting point. Because it is defined (when negation-normal form is considered) in terms of the "existential" next-operator $\Diamond$, its dual $\Box$, and fixpoints over these, it allows a concise identification of the universal and existential fragments even when interpreted over non-total transition relations. This approach is followed

in [DGG].

Although the model checking procedure itself is an automated process, it is not obvious how the choice of an appropriate abstract domain with corresponding abstract operations, as well as the proofs that these operations satisfy the conditions of Definition 4.3.0.2, can be performed in an automated fashion. So far we have assumed that the abstract domain is provided by the user of the method; an example of this may be found in [Gra94]. The proofs for the abstract operators may form a difficult step in the method. In [KDG95], approximations to the transition relation of StateCharts ([Har87]) are used to verify μ-calculus properties of a production cell ([DHKS95]) in a compositional fashion. In [DGG93a] and [DGD+94], a method is developed that aims at full automation of these steps. We return to this method in the next chapter. The following section suggests another possibility.

4.8 Refinement of Abstractions

In the previous sections we have shown how various properties of nondeterministic systems can be checked by constructing suitable abstractions. Such abstractions can be constructed by abstract interpretation of programs over appropriate abstractions of elementary values. Who or what provided such abstractions was, until now, no matter of our concern. An obvious question that is raised is: What to do when neither the formula nor its negation can be established over the abstraction? Clearly, the reason that this occurs is that the abstract model does not contain sufficient detail to establish or refute the property. Consequently, a general answer to the above question is: "Refine the abstraction". But how?

One possibility is to let the user of the verification system be responsible for indicating how to refine the current abstraction. This answer is in line with the Abstract Interpretation approach that we have followed so far. In this case, the user should have information to make a reasonable "guess". A good way to gain insight is to analyse why the formula (or its negation) cannot be established. In the case of a universal formula, useful information may be provided by counter-examples. Several model checkers exist that offer good facilities to analyse diagnostic information. The reasons for failure in proving an existential formula are harder to pinpoint. In that case, it seems important to have powerful tools to "browse" the abstract model. As this thesis is oriented on automatic methods, we do not pursue this approach.

In this thesis, we discuss two approaches to automatic refinement. The first is still in line with the idea of Abstract Interpretation. Instead of providing a single abstract domain at a time, the user is required to supply, in advance, a *sequence* of

domains, with corresponding abstract operators, having the property that each next domain is more refined than the previous. Another way to view this is as an abstract domain in which the granularity of abstract values is "tunable" through a parameter. The idea is that when verification yields an indefinite answer, the system moves on to a more refined domain. If the domain can be refined su ently, it will eventually be possible to verify the property of interest. The way in which the abstract model is refined does not depend on the form of the property, but is only driven by the shape of the parametrised domain. In this sense, this method is not goal-oriented. In particular this means that the constructed abstract model may be far larger than the minimal model strongly preserving the property. This approach is further discussed in Section 4.8.1 below.

Raising the issue of su cient conditions for strong preservation and minimality of models leads to more theoretical, model-theoretic considerations. We return to these in the next chapter.

4.8.1 Abstraction families

We assume a sequence $\{_\alpha\Sigma_i\}_{i\geq 0}$ of abstractions (with associated approximation orders $\preceq_i$) of the concrete domain $(\Sigma, \sqsubseteq)$ such that each $_\alpha\Sigma_i$ is an abstraction of $_\alpha\Sigma_{i+1}$ (see Figure 4.11). As before, the notion of abstraction is formalised through a Galois con-

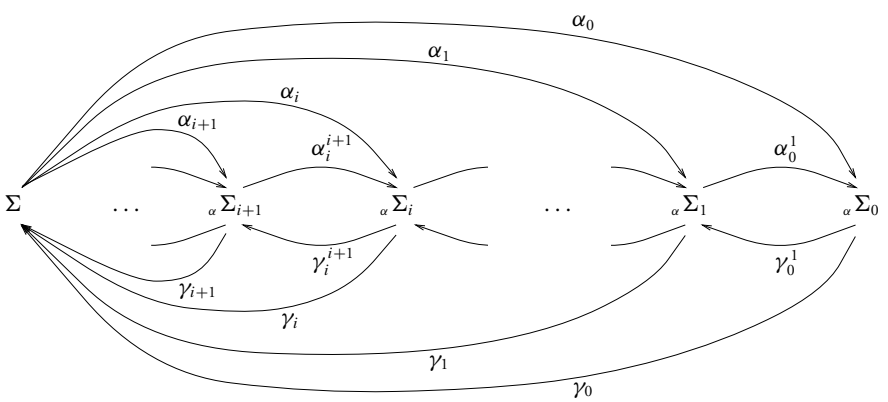


Figure 4.11: Abstraction family.

nection. For every i, the relation between Σ and $_\alpha\Sigma_i$ is given by the Galois connection (α_i, γ_i), while the relation between $_\alpha\Sigma_i$ and $_\alpha\Sigma_{i+1}$ is captured by the abstraction function $\alpha_i^{i+1} : {}_\alpha\Sigma_{i+1} \to {}_\alpha\Sigma_i$ and concretisation function $\gamma_i^{i+1} : {}_\alpha\Sigma_i \to {}_\alpha\Sigma_{i+1}$. As

a "sanity condition" we require that

$$\alpha_i = \alpha_i^{i+1} \circ \alpha_{i+1} \tag{4.26}$$

$$\gamma_i = \gamma_{i+1} \circ \gamma_i^{i+1} \tag{4.27}$$

It is easy to show that, given Galois connections $(\alpha_{i+1}, \gamma_{i+1})$ and $(\alpha_i^{i+1}, \gamma_i^{i+1})$, their composition $(\alpha_i^{i+1} \circ \alpha_{i+1}, \gamma_{i+1} \circ \gamma_i^{i+1})$ is a Galois connection as well. Therefore, the above conditions are satisfiable. On the other hand, it may be possible to devise a Galois connection (α_i, γ_i) that is di erent from this composition — but in that case the intuitive "meaning" of elements of ${}_\alpha\Sigma_i$ (i.e. what they describe) under (α_i, γ_i) will be di erent from their meaning under $(\alpha_i^{i+1} \circ \alpha_{i+1}, \gamma_{i+1} \circ \gamma_i^{i+1})$, which we consider an undesirable situation. Note that 4.26 and 4.27 can be viewed as defining (α_i, γ_i) in terms of $(\alpha_{i+1}, \gamma_{i+1})$ and $(\alpha_i^{i+1}, \gamma_i^{i+1})$, implying that when for some k the definitions of (α_k, γ_k) are provided as well as the definitions of the connections $(\alpha_i^{i+1}, \gamma_i^{i+1})$ (for every $i \leq k-1$) between subsequent abstractions[26], then this determines (α_i, γ_i) for every $i \leq k-1$. On the other hand, given (α_i, γ_i) and $(\alpha_i^{i+1}, \gamma_i^{i+1})$, it is not possible to determine $(\alpha_{i+1}, \gamma_{i+1})$. Instead, we have

$$\gamma_i^{i+1} \circ \alpha_i = \gamma_i^{i+1} \circ \alpha_i^{i+1} \circ \alpha_{i+1} \succeq \alpha_{i+1} \tag{4.28}$$

$$\gamma_i \circ \alpha_i^{i+1} = \gamma_{i+1} \circ \gamma_i^{i+1} \circ \alpha_i^{i+1} \succeq \gamma_{i+1} \tag{4.29}$$

$$\alpha_{i+1} \circ \gamma_i = \alpha_{i+1} \circ \gamma_{i+1} \circ \gamma_i^{i+1} \preceq \gamma_i^{i+1} \tag{4.30}$$

$$\alpha_i \circ \gamma_{i+1} = \alpha_i^{i+1} \circ \alpha_{i+1} \circ \gamma_{i+1} \preceq \alpha_i^{i+1} \tag{4.31}$$

(using the equalities 4.26 and 4.27, and $\gamma_i^{i+1} \circ \alpha_i^{i+1} \succeq id$ in the first two, and $\alpha_{i+1} \circ \gamma_{i+1} \preceq id$ in the last two inequalities).

One property that such a sequence of abstractions should have is that it allows verification of any property (from some fixed property set L), i.e. for every property $\varphi \in \mathrm{L}$ to be verified, there exists i such that $\mathcal{A}_i$ (the abstract model induced by ${}_\alpha\Sigma_i$) strongly preserves φ. This guarantees that the successive refinements will eventually allow us to verify the property. This requirement is clearly satisfied if some ${}_\alpha\Sigma_i$ is identical to the concrete domain[27] Σ. However, this is not a necessary condition. Depending on the form of the property set L, certain aspects of $\mathcal{C}$ need not be exposed in any $\mathcal{A}_i$, yet all L-properties are strongly preserved. We do not investigate this question at this point. In the following chapters, minimal conditions on the

[26] As an abstraction function determines the corresponding concretisation, and reversely (see Section 2.2.3), indeed only one of these functions has to be provided for each connection.

[27] This will then be the largest i that will be needed for the purpose of verification. Of course, it is desirable that verification succeeds at some smaller i, before having to resort to this domain.

abstract system for strong preservation are identified for the logic CTL* as well as for a number of fragments.

Of course, it should be possible to define the abstractions e ectively, i.e. there should be a finite and preferably compact specification that allows for the automatic construction of ever finer abstract models. Furthermore, together with the abstract-state sets ${}_\alpha\Sigma_i$, abstract interpretations of the operations from the programming language should be provided. Again, it should be possible to e ectively define these interpretations for all i. We summarise by the following definition. For simplicity, we only deal with a single function f on Σ giving the interpretation of some program operator. So, we abandon the more specific conditions and transformations of the action systems introduced before. We expect that it is not a problem to generalise this to more functions, possibly being defined over di erent concrete domains.

4.8.1.1 D *Let $(\Sigma, \sqsubseteq)$ be a poset and $f : \Sigma \to \Sigma$ a function. An* abstraction family (for Σ) *consists of an e ective definition of the following:*

1. *A (possibly infinite) sequence $\{({}_\alpha\Sigma_i, \preceq_i)\}_i$ of posets such that there exist Galois connections satisfying the sanity conditions 4.26 and 4.27 on page 94.*
2. *Corresponding safe*[28] *abstract interpretations ${}_\alpha f_i : {}_\alpha\Sigma_i \to {}_\alpha\Sigma_i$.*

In practice, the fact that these abstractions must all be e ectively defined probably means that there will be a strong uniformity in the ${}_\alpha\Sigma_i$.

We consider an example. Let $\mathbb{B} = \{0, 1\}$ and consider the sets $\mathbb{B}^k$ of bitstrings of length $k \geq 1$ and $\mathbb{B}^*$ of bitstrings of arbitrary finite length, interpreted as natural numbers in binary notation, as usual. For a bitstring $\bar{b}$ and $n \in \mathbb{N}$, $\bar{b}$ div n and $\bar{b}$ mod n are assumed to return bitstrings representing the results of integer division and remainder. We use regular-expression notation (in particular, $\cdot$ for concatenation and * for finite iteration) to define sets of bitstrings. For example, $\{0, 1\}^* \cdot 1 \cdot \{0, 1\}^*$ is the set of all bitstrings containing at least one 1, and $0 \cdot (\bar{b} \bmod 2^k)$ is the singleton containing the string obtained by concatenating 0 and the result of $\bar{b}$ mod 2^k. When we consider pairs of bitstrings, we write (B_1, B_2) (where the B_i are sets represented by regular expressions) to denote $\{(b_1, b_2) \mid b_1 \in B_1, b_2 \in B_2\}$. The function *cut_zeros* : $\mathbb{B}^* \to \mathbb{B}^*$ removes all leading zeros from a bitstring while *fill_zeros*$_k$: $(\bigcup_{i=0}^{k} \mathbb{B}^k) \to \mathbb{B}^k$ prepends a string of zeros so that the length of the resulting bitstring is k. Finally, the function *bit_or* : $\mathbb{B}^* \to \mathbb{B}$ returns the bitwise "or" of the bits of a string; $\vee$ is used for the binary "or". Functions are extended pointwise to (regular) sets.

[28] Safety of an abstract function ${}_\alpha f$ with regard to f is defined as condition 3.17 on page 52.

We define the concrete domain $\Sigma = \mathcal{P}(\mathbb{B}^*)$ and for every $k \geq 1$, the abstract domains ${}_\alpha\Sigma_k = \mathcal{P}(\mathbb{B} \times \mathbb{B}^k)$. Intuitively, we abstract numbers (represented as bit-strings) by chopping o all bits with the exception of the k lowest (i.e. rightmost — so we keep the least significant bits, possibly filled up with zeros if there are fewer than k). Besides these k bits, the abstracted number contains one additional bit that indicates whether the resulting number is still precise (i.e. the original number can be represented in k bits). Such a concrete domain may occur in the context of binary decision diagrams. The elements of the abstract domain $\mathcal{P}(\mathbb{B} \times \mathbb{B}^k)$ can then be used as approximations that can be represented by BDDs of limited height, see e.g. [DGD+94]. An element of $\mathbb{B} \times \mathbb{B}^k$ is represented as $(e, \bar{b})$ where $e \in \mathbb{B}$ is called the *overflow* and $\bar{b}$ is in $\mathbb{B}^k$. The abstraction is formally captured by the following abstraction and concretisation functions. For $B \in \mathcal{P}(\mathbb{B}^*)$ and $A \in \mathcal{P}(\mathbb{B} \times \mathbb{B}^k)$:

$$\begin{aligned}\alpha_k(B) &= \{(\mathit{bit_or}(\bar{b} \operatorname{div} 2^k), \mathit{fill_zeros}_k(\bar{b} \bmod 2^k)) \mid \bar{b} \in B\} \\ \gamma_k(A) &= \bigcup\{0^* \cdot \mathit{cut_zeros}(\bar{a}) \mid (0, \bar{a}) \in A\} \\ &\quad \cup \bigcup\{\{0,1\}^* \cdot 1 \cdot \{0,1\}^* \cdot \bar{a} \mid (1, \bar{a}) \in A\}\end{aligned}$$

It is easy to verify that for every $k \geq 1$, (α_k, γ_k) forms a Galois insertion from $(\Sigma, \subseteq)$ to $({}_\alpha\Sigma_k, \subseteq)$.

Next, we give the relation between successive ${}_\alpha\Sigma_k$.

$$\begin{aligned}\alpha_k^{k+1}(B) &= \{(e \vee (\bar{b} \operatorname{div} 2^k), \mathit{fill_zeros}_k(\bar{b} \bmod 2^k)) \mid (e, \bar{b}) \in B\} \\ \gamma_k^{k+1}(A) &= \bigcup\{(\{\{0\}, 0 \cdot \bar{a}) \mid (0, \bar{a}) \in A\} \\ &\quad \cup \bigcup\{(\{\{1\}, \{0,1\} \cdot \bar{a}) \cup (\{0\}, 1 \cdot \bar{a}) \mid (1, \bar{a}) \in A\}\end{aligned}$$

Again, it is straightforward to show that $(\alpha_k^{k+1}, \gamma_k^{k+1})$ forms a Galois insertion for every k and that the sanity conditions 4.26 and 4.27 on page 94 are satisfied (with $\subseteq$ for $\preceq$).

If the values in the concrete domain are bounded, then clearly some ${}_\alpha\Sigma_i$ will induce a strongly-preserving abstract model.

Incrementally computing abstract functions

It would be desirable if the abstract interpretations can be computed incrementally, i.e. in such a way that the results from the previous abstraction can be re-used. This allows for a progressive computation of the transition relation ${}_\alpha\mathrm{R}_i$ from the previously computed ${}_\alpha\mathrm{R}_{i-1}$ and hence minimises the e ort involved in a refinement step. The rest of this section documents some initial investigation of this condition.

Consider the function f giving the interpretation of some programming operator, and let ${}_\alpha f_i : {}_\alpha\Sigma_i \to {}_\alpha\Sigma_i$ be its interpretation over ${}_\alpha\Sigma_i$. In defining the

interpretation of f over the next refined abstract domain ${}_\alpha\Sigma_{i+1}$, we could simply use the Galois connection $(\alpha_i^{i+1}, \gamma_i^{i+1})$ between ${}_\alpha\Sigma_i$ and ${}_\alpha\Sigma_{i+1}$ and define: ${}_\alpha f_{i+1} = \gamma_i^{i+1} \circ {}_\alpha f_i \circ \alpha_i^{i+1}$. This yields a safe function: as ${}_\alpha f_i$ is safe, i.e. $\alpha_i \circ f \preceq {}_\alpha f_i \circ \alpha_i$, we have ${}_\alpha f_i \succeq \alpha_i \circ f \circ \gamma_i$ and hence ${}_\alpha f_{i+1} \succeq \gamma_i^{i+1} \circ \alpha_i \circ f \circ \gamma_i \circ \alpha_i^{i+1}$, from which by the inequalities 4.28 and 4.29 it follows that ${}_\alpha f_{i+1} \succeq \alpha_{i+1} \circ f \circ \gamma_{i+1}$. However, such a definition obviously ignores the extra precision that ${}_\alpha\Sigma_{i+1}$ o ers over ${}_\alpha\Sigma_i$, because the element that ${}_\alpha f_{i+1}$ has to be applied to is abstracted to the domain ${}_\alpha\Sigma_i$. (To formalise this observation, we have to compare ${}_\alpha f_{i+1}$ and ${}_\alpha f_i$ by applying them to some concrete object via α_{i+1} and α_i resp.: $\gamma_{i+1} \circ {}_\alpha f_{i+1} \circ \alpha_{i+1}$ vs. $\gamma_i \circ {}_\alpha f_i \circ \alpha_i$. By substituting the above definition of ${}_\alpha f_{i+1}$ in the first and using the inequalities 4.30 and 4.31, we see that these are equal functions.) We continue our example to illustrate this.

As function f, we consider multiplication by 2; on bitstrings this has the e ect of appending a zero to the right. Formally, we define $f : \mathcal{P}(\mathbb{B}^*) \to \mathcal{P}(\mathbb{B}^*)$ by $f(B) = \bigcup\{\bar{b}\cdot 0 \mid \bar{b} \in B\}$. On abstract values, this operation may cause the overflow to be set. We define ${}_\alpha f_k(A) = \bigcup\{(\mathit{bit_or}(e\cdot(\bar{a}\cdot 0)\mathrm{div}2^k), \mathit{fill_zeros}_k((\bar{a}\cdot 0)\mathrm{mod}2^k)) \mid (e, \bar{a}) \in A\}$. It is not di cult to see that for the functions ${}_\alpha f_k$ thus defined we have ${}_\alpha f_k = \alpha_k \circ f \circ \gamma_k$ for every k.

Now, suppose that for a certain k we compute $\gamma_k^{k+1} \circ {}_\alpha f_k \circ \alpha_k^{k+1}$ instead of ${}_\alpha f_{k+1}$. The following example illustrates how precision is lost with respect to ${}_\alpha f_{k+1}$, for the case $k = 2$.

$$
\begin{aligned}
& \gamma_2^3 \circ {}_\alpha f_2 \circ \alpha_2^3(\{(0, 101)\}) \\
= \quad & \\
& \gamma_2^3 \circ {}_\alpha f_2(\{(1, 01)\}) \\
= \quad & \\
& \gamma_2^3(\{(1, 10)\}) \\
= \quad & \\
& \{(1, 010), (1, 110), (0, 110)\} \\
\supseteq \quad & \\
& \{(1, 010)\} \\
= \quad & \\
& {}_\alpha f_3(\{(0, 101)\})
\end{aligned}
$$

This di erence in precision is clearly caused by the loss of information about the position of the leftmost bit in 101, when abstracting from ${}_\alpha\Sigma_3$ to ${}_\alpha\Sigma_2$ in the first step. On the other hand, for the part of 101 that remains intact when going from ${}_\alpha\Sigma_3$ to ${}_\alpha\Sigma_2$, viz. the last two bits, we could "re-use" the result of applying ${}_\alpha f_2$ to this

part: the same last two bits of the result, 10, also occur in the desired more precise value. And to find the rest of this more precise value, we do not need to compute the function ${}_\alpha f_3$ on the value (0, 101), but we need its e ect on that part of (0, 101) which was lost when going from ${}_\alpha\Sigma_3$ to ${}_\alpha\Sigma_2$.

Let us reformulate this more generally, using Figure 4.12. Given are a con-

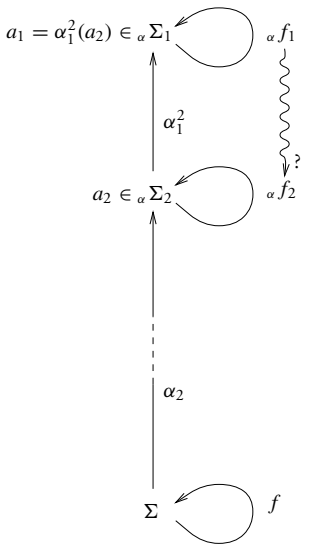


Figure 4.12: Reuse of the more abstract ${}_\alpha f_1$.

crete domain Σ, an operation f on it, an abstraction ${}_\alpha\Sigma_1$ of Σ with the optimal abstraction ${}_\alpha f_1$ of f, and a more concrete abstraction ${}_\alpha\Sigma_2$, i.e. ${}_\alpha\Sigma_1$ is an abstraction of ${}_\alpha\Sigma_2$ that in turn abstracts Σ. We want (this intention is indicated by the squiggly arrow) to define the optimal abstraction ${}_\alpha f_2$ of f on ${}_\alpha\Sigma_2$, by reusing ${}_\alpha f_1$ as much as possible. When we abstract a given object $a_2 \in {}_\alpha\Sigma_2$ to $a_1 \in {}_\alpha\Sigma_1$ and then apply ${}_\alpha f_1$ to it, certain information is lost. We would like to have some way to extract this information, say r ("residue") from a_2. Then, it should be possible to compose r with ${}_\alpha f_1(a_1)$ in such a way that the result is the same as ${}_\alpha f_2(a_2)$.

Consider the example again. When $\{(0, 101)\} \in {}_\alpha\Sigma_3$ is abstracted to $\{(1, 01)\} \in {}_\alpha\Sigma_2$, information about the leftmost bit of 101 is lost. Furthermore, when ${}_\alpha f_2$ is applied, resulting in $\{(1, 10)\}$, also the middle bit of 101 is lost. Hence, the residue of $\{(0, 101)\}$, relative to the operation ${}_\alpha f_2 \circ \alpha_2^3$, should at least give the first two bits of 101 as well as the overflow bit 0.

We formalise this by another abstraction of ${}_\alpha\Sigma_2$, say α_r, into the domain ${}_\alpha\Sigma_r$.

See Figure 4.13. The condition formulated above can now be formalised by requiring

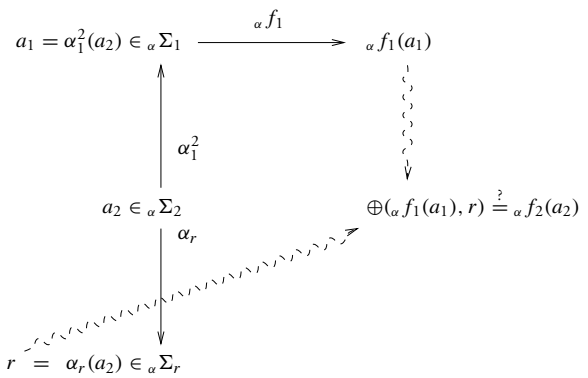


Figure 4.13: Residue abstraction.

the existence of a "composition function" $\oplus : {}_\alpha\Sigma_1 \times {}_\alpha\Sigma_r \to {}_\alpha\Sigma_2$ such that for every $a_2 \in {}_\alpha\Sigma_2$, $\oplus({}_\alpha f_1 \circ \alpha_1^2(a_2), \alpha_r(a_2)) = {}_\alpha f_2(a_2)$.

There is another requirement that we would like to impose, namely that the residue abstraction provides the *least* amount of information needed for the existence of such a compostion function. This should ensure that the composition of $\alpha_r(a_2)$ with ${}_\alpha f_1(\alpha_1^2(a_2))$, indicated by the two dashed squiggly arrows, which is supposed to yield the same result as if we had applied the optimal ${}_\alpha f_2$ to a_2, does not "redo" anything that ${}_\alpha f_1$ already does. In terms of the example given above: The residue abstraction, relative to ${}_\alpha f_2 \circ \alpha_2^3$, of (0, 101) should *not* provide the lower bit, 1, of 101, as this information is already provided by ${}_\alpha f_2 \circ \alpha_2^3$. However, we do not yet see an appropriate way to formalise this[29]. Furthermore, we expect that in practical situations it will sometimes be more convenient if the residue abstraction does contain some redundancy, as that may allow for simpler composition functions.

We continue our example about bitstrings. It turns out that the residue abstraction suggested before, which selects the overflow bit as well as the first two bits of the bitstring, is not yet su cient to allow the definition of a composition function. The problem is that the elements of ${}_\alpha\Sigma_3$ are (unordered) *sets*. When we take such a set A

[29]Also, we could continue by defining a notion of "incremental abstraction family", in which for every $i \geq 1$ there exists a residue abstraction α_r^i of ${}_\alpha\Sigma_i$ with respect to ${}_\alpha f_{i-1} \circ \alpha_{i-1}^i$ such that ${}_\alpha f_i$ can be defined by ${}_\alpha f_i(a_i) = \oplus^i({}_\alpha f_{i-1}(\alpha_{i-1}^i(a_i)), \alpha_r^i(a_i))$ where $\oplus^i$ is the composition function on ${}_\alpha\Sigma_i$ with respect to f. However, because we do not require residue abstractions to give minimal information, every abstraction family would be incremental. The reason is that the identity function on ${}_\alpha\Sigma_i$ is a valid residue abstraction.

and select, of each of the pairs $(e, \bar{a}) \in A$, the overflow bit e and the first two bits of $\bar{a}$, we get a set again, say $\alpha_r^3(A)$, and the information about which element of $\alpha_r^3(A)$ belongs to which elements of A, is lost. Consider $A = \{(0, 000), (0, 101)\} \in {}_\alpha\Sigma_3$. ${}_\alpha f_2(\alpha_2^3(A)) = \{(0, 00), (1, 10)\}$. The residue would be $\alpha_r^3(A) = \{(0, 00), (0, 10)\}$ (where the first element of a pair is the overflow bit of the abstracted value and the second element gives the two leftmost bits). A composition function would combine these two sets into a set with four elements, while the required result should equal ${}_\alpha f_3(A) = \{(0, 000), (1, 010)\}$, which has only two elements.

A natural solution is to change the abstract domains so that the elements of sets become ordered, for example by using lists instead of sets. Then, for every k, a composition function can be defined as the pointwise extension over these lists of the following function $\oplus' : (\mathbb{B} \times \mathbb{B}^k) \times (\mathbb{B} \times \mathbb{B}^2) \to (\mathbb{B} \times \mathbb{B}^{k+1})$:

$$\oplus'((e_k, \bar{a}_k), (e_2, \bar{a}_2)) = (e_2', \bar{a}_2' \cdot \bar{a}_k) \text{ where } (e_2', \bar{a}_2') = {}_\alpha f_2((e_2, \bar{a}_2))$$

This is as much as we would like to say about abstraction families. Their usefulness can only be judged by gaining practical experience from diverse applications — which is an activity outside the scope of this thesis.

4.9 Related Work

Property-preserving abstractions of reactive systems have been the topic of intensive research lately. Most of these e orts are based on the notion of simulation (Definition 2.4.2.1). *Homomorphisms* (see e.g. [Gin68]), used in automata theory to construct language preserving reductions of automata, can be viewed as a precursor of this. Adapted to our notion of transition system, $h : \Sigma \to {}_\alpha\Sigma$ is a homomorphism i $c \in \mathrm{I}$ implies $h(c) \in {}_\alpha\mathrm{I}$ and $\mathrm{R}(c, d)$ implies ${}_\alpha\mathrm{R}(h(c), h(d))$, where ${}_\alpha\mathrm{I}$ is the set of initial abstract states and ${}_\alpha\mathrm{R}$ the abstract transition relation. In [Mil71], Milner introduced the term simulation to denote a homomorphism between deterministic systems. Since then, it has been re-adapted to nondeterministic transition systems and has become popular in the areas of program refinement and verification; [Sif82], [Sif83] and [HM80] are some early papers on this topic. [Dil89] and [Kur90] focus on trace (linear time) semantics and universal safety and liveness properties. Some of the first papers that consider the (strong) preservation of full CTL* and L_μ are [CGL92] and [BBLS92].

Following [Kur90], [CGL92] defines the relation between the concrete and abstract model by means of a homomorphism h, which induces an equivalence relation $\sim$ on the concrete states, defined by $c \sim d \;\Leftrightarrow\; h(c) = h(d)$. The abstract states are then representations of the equivalence classes of $\sim$. It is shown that properties

expressed in ∀CTL* are preserved from the abstract to the concrete model. Preservation of full CTL* is shown to hold when h is *exact*, which boils down to requiring the concrete and abstract Kripke structures to be bisimilar (Definition 2.4.2.7). Consequently, CTL* is *strongly* preserved, thus only allowing for relatively small reductions in the size of model. A notion of approximation between abstract systems is given based on the subset ordering on transition relations. As a result, an abstraction that is based on an exact h cannot be approximated, except by itself. Our approach to defining approximations in Section 4.4 is a generalisation of this — see Lemma 4.4.0.3 and the last sentence of that subsection. Furthermore, our notion of abstract Kripke structure *does* allow proper approximations in the context of weak preservation of full CTL*. [CGL92] also explains the construction of abstract models and approximations thereof, by abstract interpretation of elementary operations (called *abstract compilation*), and illustrates this with a number of examples. A journal version appeared as [CGL94]. [Lon93] also contains these results, presented in a slightly more general setting.

[BBLS92] presents similar ideas in a more general setting by considering simulation relations to connect the concrete and abstract transition systems. Also, properties are preserved in the μ-calculus, L_μ, which is a more powerful logic than CTL*. Preservation of both the existential ($\Diamond L_\mu$) and the universal ($\Box L_\mu$) fragments of L_μ is dealt with in the setting of weak preservation. It is shown that if there exists a simulation from $\mathcal{C}$ to $\mathcal{A}$ that is total on Σ, then properties expressed in $\Box L_\mu$ are preserved from $\mathcal{A}$ to $\mathcal{C}$, while existential properties $\Diamond L_\mu$ are preserved from $\mathcal{C}$ to $\mathcal{A}$. Again, preservation of the full μ-calculus is only shown for abstractions that are bisimilar to $\mathcal{C}$. The construction of abstract models, which is only briefly touched upon in [BBLS92], is worked out further in the journal version, [LGS+95], where it is shown in addition how the abstraction of a concurrent system can be constructed compositionally from the abstractions of the individual components. In [Loi94], this theory is not only worked out in full detail, but the implementation of a tool based on it is described and analysed too. A closely related paper is [GL93]. The approach is similar to that taken in Section 4.3, although the results deviate because the underlying frameworks are slightly di erent.

In [LGS+95], Loiseaux *et al.* also use Galois connections to relate concrete and abstract states spaces, but in a di erent way than we do. It is shown that in their case, this is equivalent to using simulation relations. However, being between $\mathcal{P}(\Sigma)$ and $\mathcal{P}({}_\alpha\Sigma)$, these connections do not impose structure on the set ${}_\alpha\Sigma$ of abstract states (cf. the remarks below equation 4.2 on page 59). In particular, no approximation ordering $\preceq$ to relate the precision of abstract states is defined. As a result, that approach is more general, but fails to capture the notion of optimality, both on the level of states and on the level of complete transition systems. On the other hand, our approach is

a proper instance of the simulation-based framework, and does distinguish between optimal abstractions of transition systems (as captured by the abstraction function α^M) and approximations (expressed by the relation $\preceq$ on abstract transition systems, see Definition 4.4.0.1). This is further discussed in Section 4.9.1 below.

[Kel95] also discusses the preservation of universal and existential μ-calculus properties within the framework of Abstract Interpretation. As in [BBLS92], the relation between abstract and concrete systems is defined through simulations cast in the form of Galois connections. The interpretation of a μ-calculus formula, which is a set of states, is approximated from below and above. By combining these dual approximate interpretations, using one for the $\Box$-operator and the other for the $\Diamond$, weak preservation of arbitrary μ-calculus formulae is obtained. This technique is similar to the mixed abstractions presented in this chapter. A strong point of Kelb's thesis is the integration of these theoretical results with symbolic (BDD-based) representations. Indeed, [Kel95] proceeds by decribing practical experiments on the symbolic verification of StateChart programs, including part of the material from [KDG95], which was discussed in Section 4.9.2.

[CIY94] is based on an early version, [DGG93b], of [DGG94] and independently develops the idea of mixing both free and constrained abstractions in a single abstract system to attain preservation of full CTL*. More recently, [CIY95] focusses on the issue of optimality. An approximation ordering $\propto_h$ (relative to a homomorphism h) on abstract transition systems is defined and shown to coincide with the CTL*-property ordering, i.e. $\mathcal{T}_1 \propto_h \mathcal{T}_2$ ($\mathcal{T}_1$ is an approximation of $\mathcal{T}_2$) if and only if any CTL*-property satisfied by $\mathcal{T}_1$ is satisfied by $\mathcal{T}_2$ as well. This should be compared to our approach in which the notion of approximation on Abstract Kripke structures, $\preceq$, is more loosely related to the CTL*-property ordering: see Theorem 4.4.0.2 and also the footnote on page 39. [CIY95] defines an abstraction function (cf. our α^M) that maps transition systems to "$\propto_h$-optimal"[30] abstractions. This approach ensures that the formal notion of optimality corresponds more to the intuition of usefulness. It would be interesting to investigate how similar results can be obtained in our case. As shown in [CIY95], it requires certain restrictions on the class of transition systems. Furthermore, the framework of [CIY95], being based on homomorphic functions h, is slightly less general than ours so that the results do not translate immediately.

[CR94] presents a framework for the abstract interpretation of processes that pass values. Application of Abstract Interpretation to verify properties of CCS is described in [DFFGI95].

While developed independently, and from a di erent perspective, Abstract Kripke structures bear some resemblance to the *modal transition systems* of [LT88], which also combine two types of transition relations ("may" and "must"-transitions) in one

[30] In [CIY95], a di erent notation is used for this.

system. Modal transition systems have been developed in the area of specification. Must-transitions specify what is required while may-transitions specify what is admissible. In [LT88], a notion of refinement is defined such that the must-transitions in the specification simulate those in the refined system, while the may-transitions in the refined system simulate those in the specification. This is similar to our definition of approximation between Abstract Kripke structures (Definition 4.2.3.7). On the other hand, in modal transition systems, the must-relation is required to be a subset of the may-relation. Also, there is no notion of approximation ordering between states.

A recent paper, [KDG95], reports on an application of abstract interpretation techniques to the verification of properties of a production cell. Section 4.9.2 below contains an account on the findings. First, we discuss in some more depth the relation between our Galois-insertion approach and the simulation-based approach taken in [LGS$^+$95] for example.

4.9.1 Comparing the simulation-based and Galois-insertion approach

The conditions under which CTL* is preserved from a mixed transition system $\mathcal{A}' = (\Sigma', F', C', I')$ to the concrete system $\mathcal{C} = (\Sigma, \mathrm{R}, \mathrm{I})$ may be formulated entirely in terms of pseudo-simulations (Definition 2.4.2.1) as follows. There should exist a relation $\sigma \subseteq \Sigma \times \Sigma'$ such that:

1. For every $c \in \Sigma$ and $a \in \Sigma'$, if $\sigma(c, a)$ then $\forall_{p \in \mathrm{Lit}}\, a \in \|p\|'_{\mathrm{Lit}} \Rightarrow c \in \|p\|_{\mathrm{Lit}}$ (where $\|\cdot\|'_{\mathrm{Lit}}$ specifies the valuation of literals over states in Σ').
2. R σ-pseudo-simulates F'.
3. C' σ^{-1}-pseudo-simulates R.
4. For every $c \in \mathrm{I}$ there exists $a \in I'$ such that $\sigma(c, a)$.

Such a simulation-based approach, as we call it, is a generalisation of our approach using Galois insertions, as expressed by the following lemma.

4.9.1.1 L *Let $\mathcal{A}' = (\Sigma', F', C', I')$ such that $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$. Then there is a relation $\sigma \subseteq \Sigma \times \Sigma'$ such that the conditions 1–4 above are satisfied.*

P . Note that $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$ implies, by Definition 4.4.0.1, that Σ' is equal to the set ${}_\alpha\Sigma$ of abstract states, which is connected to the concrete states via (α, γ) and on which the relation $\preceq$ and the valuation ${}_\alpha\|\cdot\|_{\mathrm{Lit}}$ of literals have been defined (so $\|\cdot\|'_{\mathrm{Lit}} = {}_\alpha\|\cdot\|_{\mathrm{Lit}}$). Consider the underlying description relation $\rho \subseteq \Sigma \times \Sigma'$ defined by[31] $\rho(c, a) \Leftrightarrow c \in \gamma(a)$. We show that ρ satisfies the conditions 1–4 above.

[31] Note that this is equivalent to $\gamma = pre^{\bullet}_{\rho}$.

1. Suppose that $\rho(c, a)$. Let $p \in \text{Lit}$ such that $a \in {}_{\alpha}\|p\|_{\text{Lit}}$. Because (α, γ) is a Galois connection, $\rho(c, a)$ implies that $\alpha(c) \preceq a$, so, by Lemma 4.2.1.2, $\alpha(c) \in {}_{\alpha}\|p\|_{\text{Lit}}$. By Definition 4.2.1.1 and the fact that $\gamma(\alpha(c)) \supseteq \{c\}$, it now follows that $c \in \|p\|_{\text{Lit}}$.

2,3. For the optimal abstraction $\alpha^M(\mathcal{C})$, it is easily shown that R ρ-pseudo-simulates ${}_{\alpha}\text{R}^F$ and ${}_{\alpha}\text{R}^C$ ρ^{-1}-pseudo-simulates R. From the fact that $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$ it follows by Definition 4.4.0.1 that ${}_{\alpha}\text{R}^F \preceq$-pseudo-simulates F' and $C' \succeq$-pseudo-simulates ${}_{\alpha}\text{R}^C$. Because the composition of ρ with $\preceq$ is ρ again, it now follows by transitivity of simulation that R ρ-pseudo-simulates F' and C' ρ^{-1}-pseudo-simulates R.

4. Let $c \in \text{I}$. Then by Definition 4.2.2.1 of abstract initial states, $\alpha(c) \in {}_{\alpha}\text{I}$ (the initial states of $\alpha^M(\mathcal{C})$). From point 3 in Definition 4.4.0.1 of $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$ it now follows that there exists $a' \in I'$ with $\alpha(c) \preceq a'$. So $\rho(c, a')$. □

Conversely, if for a mixed system $\mathcal{A}' = (\Sigma', F', C', I')$, there exists a relation $\sigma \subseteq \Sigma \times \Sigma'$ such that conditions 1–4 above hold, then it need not be the case that $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$: although every concretisation function γ induces an underlying description relation ρ by $\rho(c, a) \Leftrightarrow c \in \gamma(a)$ that satisfies conditions 1–4 (see the proof above), an arbitrary σ satisfying these conditions does not necessarily induce the concretisation function of a Galois insertion. So why use Galois insertions when the simulation-based approach is more general? We see two main advantages of our framework.

Useful constrained abstractions First, to define useful abstractions that preserve existential properties, where with useful we mean that a fair amount of existential properties indeed hold in the abstract system, the abstract states need to be partially ordered anyway. We illustrate this by an example. Consider Figure 4.14. c_1, c_2, d_1 and d_2 are concrete states while a, b_1, and b_2 are abstract. The relation ρ, indicated by dashed arrows, gives the relation between them, so one could say that c_1 is described by a, or that c_1 is in the concretisation of a (although a Galois insertion does not necessarily exist). In order for *universal* properties to be preserved from a to its concretisation, R has to ρ-pseudo-simulate the abstract transition relation. This implies that there have to be abstract transitions from a to both b_1 and to b_2. It is not hard to see that as long as ρ is total on the concrete states, it is always possible to find an abstract transition relation F such that R ρ-pseudo-simulates F. In order for *existential* properties to be preserved, the abstract transition relation C should ρ^{-1}-pseudo-simulate R. For the situation of Figure 4.14, no C-transition from a is possible under this requirement. This shows that an abstract domain that is suitable for defining useful abstractions that preserve universal properties is not necessarily suitable for defining useful abstractions that preserve existential properties too. In the case of this example, if we want to have an outgoing C-transition from a, we need to extend the abstract domain with a state that describes (at least) both d_1 and

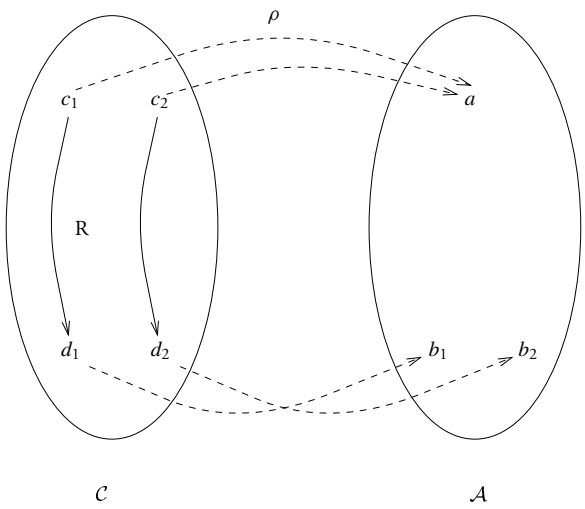


Figure 4.14: Finding simulations in both ways.

d_2. In general, to define useful abstractions preserving existential properties, the abstract domain should contain states describing subsets of concrete states of various sizes. In particular, if it should always be possible for C to be total, the abstract domain should have a "top" element describing all concrete states. The abstract state $\langle \mathsf{think}, \mathsf{think}, \top \rangle$ in Figure 4.8 is a good example of this. Without it, property 4.6 (page 74) could not have been verified.

Optimal abstractions vs. approximations Second, the mere requirement that a (pseudo-)simulation must exist in order for preservation to hold has the drawback that it leaves too much freedom in the choice of "good" abstractions. The Galois insertion framework in which we developed our results may be viewed as a (successful) attempt to try and quantify the notion of *precision* of abstractions by distinguishing between the notions of *(optimal) abstraction* and *approximation*. This point is discussed in the following comparison of our work with that of [LGS+95, Loi94].

We focus on the free abstract transition relation. Given a concrete transition system and a set of abstract states that is related to the concrete states by a description relation $\rho \subseteq \Sigma \times \Sigma'$ (total on Σ), the requirement $\rho^{-1}\mathrm{R} \subseteq {}_{\alpha}\mathrm{R}\,\rho^{-1}$ (R pseudo-simulates ${}_{\alpha}\mathrm{R}$, see Lemma 2.4.2.2), viewed as an inequality over ${}_{\alpha}\mathrm{R}$, has many solutions. From the point of view of property preservation, the $\subseteq$-minimal solutions are interesting. Namely, the $\subseteq$-smaller the abstract transition relation ${}_{\alpha}\mathrm{R}$, the greater

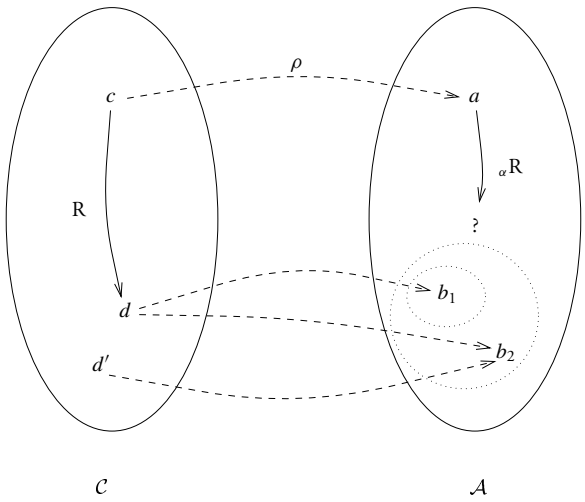


Figure 4.15: Abstraction with states of comparable precision.

the number of universal properties that hold in $\mathcal{A}$. However, even such minimal solutions may have comparable quality, as illustrated by the example in Figure 4.15, where the problem is to choose an ${}_\alpha$R-successor of a such that $\rho^{-1}\mathrm{R} \subseteq {}_\alpha\mathrm{R}\,\rho^{-1}$ is satisfied. The $\subseteq$-minimal solutions are obtained by taking either b_1 or b_2 as successor (but not both). However, choosing b_1 will generally give better property preservation, as it describes fewer concrete states.

Instead of exploring this freedom to refine the notion of quality of abstract transition relations, [LGS$^+$95] proposes a condition under which all minimal solutions are bisimilar to each other. This condition is

$$\rho\rho^{-1}\rho = \rho \tag{4.32}$$

i.e. ρ should be a difunctional (Definition 2.1.0.2). Expressed in words, it says that if two concrete states share a description (abstract state), then they share all descriptions. For example, in Figure 4.15 also the states d' and b_1 would have to be related by ρ. It is easy to see that the generality of simulations over Galois insertions is eliminated by this condition. In fact, requirement 4.32 implies that it is useless to have a ρ that is not functional. This is expressed in the following lemma (which can be found in [LGS$^+$95]). It implies that whenever $\rho(c,a)$ and $\rho(c,a')$ $(a \neq a')$ for some c — i.e, ρ is not functional — then a and a' are bisimilar. If the goal of abstraction is to produce abstract systems with a minimal number of states[32], then one of a and a'

[32]In general, the goal is to have minimal *representations* of the system. When states are not represented

should be removed from $\mathcal{A}$.

4.9.1.2 L *If ρ is total on Σ, ${}_{\alpha}\mathrm{R}$ is a $\subseteq$-minimal relation such that* R *ρ-simulates ${}_{\alpha}\mathrm{R}$, and $\rho\rho^{-1}\rho = \rho$, then $\rho\rho^{-1}$ is a bisimulation on $\mathcal{A}$.*

P In order to show that $\rho\rho^{-1}$ is a bisimulation (Definition 2.4.2.7) on $\mathcal{A}$, we have to show that $\rho\rho^{-1}$ and $(\rho\rho^{-1})^{-1}$ are simulations on $\mathcal{A}$. Because $(\rho\rho^{-1})^{-1} = \rho\rho^{-1}$, it su ces to show that $\rho\rho^{-1}$ is a simulation, i.e. (by Lemma 2.4.2.2) that $(\rho\rho^{-1})^{-1}\,{}_{\alpha}\mathrm{R} \subseteq {}_{\alpha}\mathrm{R}(\rho\rho^{-1})^{-1}$, i.e. $\rho^{-1}\rho\ {}_{\alpha}\mathrm{R} \subseteq {}_{\alpha}\mathrm{R}\ \rho^{-1}\rho$ (*). Because any minimal solution ${}_{\alpha}\mathrm{R}$ satisfies ${}_{\alpha}\mathrm{R} = \rho^{-1}\mathrm{R}\ \rho$ (see [LGS+95]), (*) is equivalent to $\rho^{-1}\rho\rho^{-1}\mathrm{R}\ \rho \subseteq \rho^{-1}\mathrm{R}\ \rho\rho^{-1}\rho$. Because $\rho\rho^{-1}\rho = \rho$ and therefore also $\rho^{-1}\rho\rho^{-1} = \rho^{-1}$ (apply the inverse, $(\cdot)^{-1}$, to both sides), this is equivalent to $\rho^{-1}\mathrm{R}\ \rho \subseteq \rho^{-1}\mathrm{R}\ \rho$, which is obviously true. □

So, to distinguish optimal abstractions from approximations, [LGS+95] make assumption 4.32, which renders their framework less general than the Galois-insertion approach, because, under the reasonable assumption that the abstract system does not contain bisimilar states, it forces ρ to be functional.

Consider Figure 4.15 again. In our framework, the simulation relation ρ induces the following Galois insertion on sets of states: for any $a \in \Sigma'$, $\gamma(a) = pre^{\bullet}_{\rho}(a)$ and for any $C \subseteq \Sigma$, $\alpha(C) = \bigwedge\{a \mid \gamma(a) \supseteq C\}$, where $\bigwedge$ denotes the glb corresponding to the ordering $\preceq$ defined by $a \preceq a' \Leftrightarrow \gamma(a) \subseteq \gamma(a')$. Taking ${}_{\alpha}\mathrm{R}$ to be ${}_{\alpha}\mathrm{R}^{F}$ as specified by Definition 4.2.3.3 yields b_1 as the only successor of a, as desired.

4.9.2 [KDG95]: An application

In a recent paper, [KDG95], we report on an application of abstract-interpretation techniques to the verification of universal and existential properties of an industrial production cell. This section gives a summary. The cell, depicted in Figure 4.16, has served as a case study for the evaluation of a number of formal methods, see [LL95]. It consists of two conveyor belts (the *feed belt* in the front and the *deposit belt* in the back of the picture), a rotary table (right to the feed belt), a press (to the right of the picture), a robot with two independent movable arms (between rotary table and press) and a crane (under the top of the picture). In addition, we assume that there is a user who puts a metal blank on the feed belt. When the user puts a new metal blank on the feed belt, it will be transported to the rotary table. The table will rotate and lift the blank up, so it can be picked up by the first robot arm. The robot puts the blank into the press. After pressing the blank, the second robot arm fetches the pressed blank and puts it on the deposit belt, which will transport the blank to the

explicitly, but by BDDs for example, the size of the representation may actually shrink as the number of states grows. In such cases, it may indeed be useful to have some "redundant" states around.

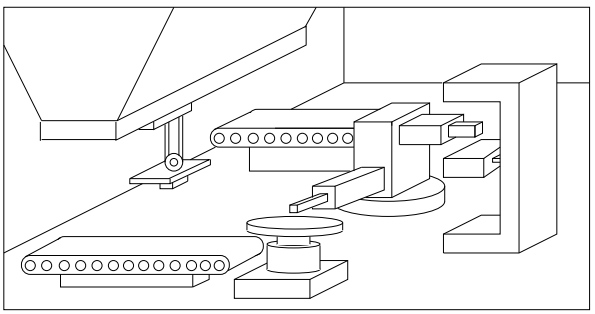


Figure 4.16: Production cell.

crane. The crane closes the cycle and moves the blank from the deposit belt to the feed belt to press it again.

Typical properties to be verified involve various combinations of universal/existential path quantification and safety/liveness properties. Some examples are (informally):

1. It is always the case that the vertical position of the rotary table is within the specified limits.

2. It is always the case that if the vertical position of the rotary table is at its lowest point and its horizontal position is such that it can accept a metal blank from the belt and the belt is running while there is a metal blank on its rightmost end, then there exists a continuation such that eventually there will be a metal blank on the rotary table.

This system is naturally modelled as a number of individual components that are composed in parallel. The transition system capturing its behaviours is obtained as the product of the transition systems modelling the individual components. Such products tend to be too large for practical purposes. Below, we sketch an application of the techniques developed in this chapter that alleviates this state explosion.

We model the components of the system as edge-labelled transition systems, called *processes* henceforth. The entire system is then represented as the program formed by the parallel composition of these. For each process, its set of local states is disjoint from the state set of any other process, while the labels along its edges, called *conditions*, are boolean combinations of the states of the other processes. Intuitively, an edge may be taken (resulting in a transition) if the current states of the other processes satify the condition along the edge. Formally, each process P_i ($0 \leq i \leq n$) induces a Kripke structure $\mathcal{K}_i$ over the global state space that consists

of tuples $\langle \ell_0, \ldots, \ell_n \rangle$ where for each $0 \leq j \leq n$, ℓ_j is a local state of process j. There is a transition from $\langle \ell_0, \ldots, \ell_n \rangle$ to $\langle \ell'_0, \ldots, \ell'_n \rangle$ in $\mathcal{K}_i$ if there is an edge from ℓ_i to ℓ'_i in P_i that is labelled by a condition that evaluates to *true* over the global state $\langle \ell_0, \ldots, \ell_n \rangle$. Note that there are no restrictions on the local target states $\ell'_0, \ldots, \ell'_{i-1}, \ell'_{i+1}, \ldots, \ell'_n$ of the other processes — such synchronisation only takes place when the parallel composition is "expanded". Such an expansion results in a "global" Kripke structure $\mathcal{K}$, capturing the behaviour of the parallel composition as a whole. $\mathcal{K}$ is the synchronous product of the $\mathcal{K}_i$: its transition relation is the intersection of the transition relations of the individual $\mathcal{K}_i$. The size of this global Kripke structure may be prohibitively large for model checking.

In [KDG95], a symbolic model checking algorithm (Section 2.4.3) is used, in which sets of states and transition relations are represented by BDDs. The global transition relation R of $\mathcal{K}$ is not represented by a single BDD. Instead, a list of BDDs each representing the local transition relation R_i of the individual process P_i is kept. When pre-image sets (Definition 2.1.0.1) $pre_{R_0 \cap \cdots \cap R_n}(S)$ and $\widetilde{pre}_{R_0 \cap \cdots \cap R_n}(S)$ of some set S of states over the global transition relation have to be computed[33], which is a key operation in symbolic model checking, the intersection of the R_i needs to be computed. It is this intermediate result that turns out to be the bottle neck in the model-checking process. Hence, the aim is to reduce the size of these representations. A notorious property of BDDs is that their size does not correlate directly to the size of the set being represented. In this particular case, it turns out that the size of the intermediate "product BDD" blows up as a result of the *interaction* between the various processes (this observation can be made more precise by looking at the particular way states are encoded). Accordingly, the idea behind the abstractions used in [KDG95] is to reduce such interactions. This is achieved by maintaining both overapproximations $R_i^{\Box} \supseteq R_i$ and underapproximations $R_i^{\Diamond} \subseteq R_i$ to the transition relations R_i. Clearly, the intersection $R_0^{\Box} \cap \cdots \cap R_n^{\Box}$ ($R_0^{\Diamond} \cap \cdots \cap R_n^{\Diamond}$) of over-(under-)approximations yields an over-(under-)approximation again. Furthermore, it is not difficult to see that for every state set S, $\widetilde{pre}_{R_i^{\Box}}(S) \subseteq \widetilde{pre}_{R_i}(S)$ and that $pre_{R_i^{\Diamond}}(S) \subseteq pre_{R_i}(S)$ for every i. Hence, when $\widetilde{pre}$ is computed on overapproximations and pre on underapproximations, the result is always an underapproximation of the intended set of states. In the symbolic model-checking approach, this set of states is the characteristic set[34] of the formula being checked. Hence, this approximation is "safe" in the sense that any state in the set that is obtained, is guaranteed to satisfy the formula being checked.

[33] The results of [KDG95] are presented in the setting of the μ-calculus. $\widetilde{pre}$ is needed to evaluate universal ($\Box$-)properties, while pre is needed for existential ($\Diamond$-)properties.

[34] The characteristic set of a formula φ is the set of all states satifying φ; cf. Definition 5.2.0.2 on page 121.

The approach can be formalised in the framework presented in this chapter. Let $\mathcal{K}_i = (\Sigma_i, R_i, \|\cdot\|_i)$ be the Kripke structure corresponding to process P_i. The abstraction function $\alpha_i : \Sigma_i \to \mathcal{P}(\Sigma_i)$ on the level of states maps each concrete state c to the singleton $\{c\}$. We ignore the di erence between single elements and singletons, so we may say that a concrete state is mapped to itself; hence no real abstraction takes place. The Abstract Kripke structure $\alpha^M(\mathcal{K}_i)$ is then $(\Sigma_i, R_i, R_i, \|\cdot\|_i)$. The abstract transition system $(\Sigma_i, R_i^{\Box}, R_i^{\Diamond}, \|\cdot\|_i)$ over which the properties are interpreted now is an approximation, in the sense of Definition 4.4.0.1 in Section 4.4, to $\alpha^M(\mathcal{K}_i)$. This follows from Lemma 4.4.0.3.

Above, we have described a theoretical solution to the state-explosion problem. It is di cult to assess how good this solution is in practice. In other words, how often can we find approximations that both o er substantial reductions and allow many properties to be verified? Heuristically, the overapproximations $R_i^{\Box}$ are often more useable than the underapproximations $R_i^{\Diamond}$. This is because in the lattice of relations, program relations tend to be closer to the empty relation than they are to the full one. Indeed, a program describes the allowed behaviour rather than those behaviours that are disallowed. E.g. a function for computing, say, the square root of its input, is usually not implemented by a program that admits arbitrary behaviour only constrained by the requirement that it computes the square root. Quite the opposite: a square root program should only perform those actions that are necessary for its correct functioning. This means that there tends to be ample room to add transitions to a relation — as an overapproximation does — without hitting the full relation, but less so to remove them. The e ect of this is that it tends to be easier to find an overapproximation that preserves the truth of a universal property than it is to find a usable underapproximation that preserves an existential property.

This problem is overcome in [KDG95] by using the overapproximations $R_i^{\Box}$ in computing an underapproximation to $pre_{R_i}(S)$. The key observation is that under a certain condition on a set S of global states, called *independence* (see Definition 6.6 in [KDG95]), we have $pre_{R_i^{\Box}}(S) \subseteq pre_{R_i}(S)$ (even though $R_i^{\Box} \supseteq R_i$). A closure operator Γ is defined that maps each set of global states to its largest independent subset; we then have $pre_{R_i^{\Box}}(\Gamma(S)) \subseteq pre_{R_i}(S)$ for every S. This pre-image set is then used to improve the quality of $pre_{R_i^{\Diamond}}(S)$ by taking the union $pre_{R_i^{\Diamond}}(S) \cup pre_{R_i^{\Box}}(\Gamma(S))$, which is clearly also an underapproximation to $pre_{R_i}(S)$.

This idea is applied to the verification of various properties of the production cell. To this purpose, the system is modelled in StateCharts ([Har87]), a graphical specification language for reactive systems. It is based on automata theory and allows the specification of automata that are composed in parallel. It permits a straightforward translation into (lists of implicitly intersected) BDDs; see [HK94]. The general

structure of the StateChart specification of the production cell is a parallel composition, where each process corresponds to one of the components mentioned above. The lock-step composition of processes is such that a process "stutters" (i.e. makes a transition leading back to its current local state) if no other transitions are enabled. As a result, the transition relation of the global Kripke structure is total.

The properties are formalised in CTL, which can be considered as an abbreviation of L_μ. Some of the resulting formulae are in either of the fragments $\forall$CTL or $\exists$CTL, others contain both types of path quantifiers. None of the formulae containing existential quantifiers could be verified over the unabstracted system. The verification of strictly universal properties requires intermediate BDDs consisting of approximately 36.000 up to 100.000 nodes, depending on the property.

These numbers are then compared with the results of applying the approximation techniques explained above. For all properties, the approximations that are applied are of the same kind: subsets of the edges of processes are selected and all conditions along these edges are abstracted. Choosing such approximations for the various properties does not require a lot of ingenuity. Using a few heuristics, approximations can be determined by looking at the specific form of the property to be checked. Indeed, all of the verifications performed succeed at the first try.

Having chosen a subset of processes whose interaction can be abstracted, the actual construction of the (BDD representation of the) abstract Kripke structure is performed by abstractly interpreting the StateChart specification. Basically, this boils down to re-interpreting the conditions along edges in the following way. For the $R^\Box$ approximation, if a condition c occurs along an abstracted edge, and c is not equivalent to *false* (i.e. if $\neg c$ is not a tautology), then c is interpreted as *true*; otherwise, c is interpreted as usual. For the the $R^\Diamond$ approximation, if a condition c occurs along an abstracted edge, and c is not equivalent to *true* (i.e. if c is not a tautology), then interpret c as *false*; otherwise, interpret c as usual. These abstract interpretations are easily seen to yield over- and underapproximations respectively. Note that conditions along *un*abstracted edges can still be evaluated as usual. The reason is that abstractions only a ect the *interactions* as expressed by the conditions, and not the states.

The resulting reductions range from 6- to 34-fold, depending on the formula, with an average of 17. All properties can be checked, including the existential ones that could not be verified without abstraction. While these initial results are certainly encouraging, further experiments are needed. In particular, we should compare this method to an approach where not only the interaction, but *all* information about subsets of process is being abstracted. Such a comparison would lead to a better understanding of the contribution of these approximations over compositional methods.

4.10 Concluding Remarks

The results of this chapter may be considered from two points of view. From the position of Abstract Interpretation, we have presented a generalisation of the framework extending it to the analysis of reactive properties. This generalisation consists in allowing the next-state relation of a non-deterministic transition system to be abstracted to a relation. This allows the analysis, via the abstraction, not only of universal properties — expressing that something holds along all possible executions —, but also existential properties — expressing the *existence* of paths satisfying some property. Furthermore, both safety as well as liveness properties are preserved. We have proven that the truth of every property expressible in CTL* is preserved from an abstract to a concrete model. As is common in Abstract Interpretation, the attained reduction solely depends on the choice of the abstraction function, thus allowing better reductions than is the case with minimisation based on bisimulation. This was possible by considering *mixed* abstract transition systems, which have two di erent transition relations, each preserving a separate fragment of CTL*. The use of a Galois insertion to relate concrete and abstract states allowed the definition of both types of transitions over the same set of abstract states, resulting in the preservation of full CTL*.

From the viewpoint of property-preserving characteristics of simulation relations, we have managed to define a notion of precision that allows us to "separate the wheat from the cha ". An abstraction function α specifies the optimal abstract model for a given concrete system, while an approximation order $\preceq$ distinguishes the relative precision between abstract models. The embedding of the property-preservation results for simulations in the framework of Abstract Interpretation opens up the possibility of constructing abstract models directly from the text of a program, thereby avoiding the intermediate construction of the full concrete model. This construction is possible by associating non-standard, abstract interpretations with the operators in a programming language that allow their evaluation over descriptions of data. To this purpose, we chose a simple programming language and defined abstract interpretations of its tests and operations. Conditions were given under which the free and constrained abstract transition relations thus computed coincide with the optimal relations as specified by α. Furthermore, a notion of approximation on the level of operations was given by which the user may simplify the task without losing the preservation results. Such approximations can accelerate the computation of abstract models, be it at the risk of obtaining a model that does not contain enough information to verify the property. It was illustrated by an example that these techniques can be applied to verify properties of systems with an infinite state space.

The price to be paid when using these abstract interpretation techniques is that

there are formulae — and increasingly so when the abstraction becomes coarser — that do not hold in the abstraction, and neither do their negations. In Section 4.8 we have proposed abstraction families as an approach to deal with such indefinite answers[35] produced by abstractions that turn out to be too coarse. By fixing a sequence of ever-finer abstract domains in advance, together with the corresponding abstract interpretations of program operations, automatic refinement is possible in the case this is needed. Furthermore, we have attempted to formalise the notion of incremental computation of the abstract transition relation. Using the new notion of a residue abstraction we have motivated a condition that should ensure reusability of the results computed in the previous step.

While we think that the presented framework o ers a stable theoretical foundation for the application of abstract interpretation in model checking, further research has to be performed to confirm this belief. Section 4.9.2 on [KDG95] already mentioned one line of on-going research. Also, the experiments described in e.g. [Gra94] show that the marriage between Abstract Interpretation and model checking is a fruitful one.

[35] Also called "false negatives" sometimes.

Chapter 5

Logical Partition Refinement

We now turn to strong preservation and investigate the consequences for the relation between Kripke structures and their descriptions. The key notion is that of consistency of the description relation with respect to a set of formulae called the companion of the property set. Consistency may be established by a partition refinement algorithm. Specific algorithms are presented for properties in ∀CTL.

5.1 Introduction

In this chapter we turn to strong preservation of temporal properties from abstract to concrete transition systems. The previous chapter presented a theory for the weak preservation of CTL*-properties, formalised in the framework of Abstract Interpretation. The use of abstraction families, introduced towards the end of that chapter to improve the precision of the abstraction by successive refinements of the abstract domain, may be viewed as a tentative approach to getting a decisive answer about truth or falsehood of a property — the underlying framework was still that of weak preservation. In the current chapter, we re-investigate the relation between the concrete and abstract model, this time starting from the requirement of strong preservation of properties.

Weak preservation of CTL* does not impose any restrictions on the minimal size of Abstract Kripke structures. No matter how many aspects are abstracted, it is always possible to obtain weak preservation from the abstract to concrete model. The only price that we pay for abstracting a lot is that few CTL*-properties will hold in the abstraction. On the other hand, the requirement of strong preservation puts a strict lower bound on how much we can abstract. For example, strong preservation of full CTL* can only be obtained if concrete and abstract transition systems are properly bisimilar. In practice, it may be the case that only part of CTL* needs to be strongly preserved. For example, the set of properties of interest is known beforehand, or is known to be contained in a proper fragment of the logic.

Therefore, we choose a set-up where weak preservation holds for all properties in the logic, while the conditions for strong preservation are investigated for specific properties only. The approach towards achieving strong preservation of certain properties that was taken by the method of abstraction families can be viewed as *domain driven*: there it was the form of the abstract domain that determined how the abstract model was going to be refined. Whether the refinement was in the "right direction" remained to be seen — the method was not goal oriented. On the other hand, the approach to refinement of models that is presented in this chapter will be *property driven*: it is the form of the specific property (or set of properties) of interest that determines how to adapt the abstract system.

Like in the previous chapter, we require *statewise* (strong) preservation: the description relation ρ is on the level of individual states, so that the properties of some abstract state give information about all concrete states being described.

We start with a small example that gives an impression of the topics of the current and next chapters. Central to these chapters is the notion of *partition refinement*. Given some set U of states equipped with an equivalence relation, a partition refinement algorithm (PRA) computes the equivalence classes by start-

ing from some initial partitioning of U and repeatedly splitting classes, until the coarsest partitioning into classes of equivalent states has been reached. Depending on the nature of the equivalence relation, di erent criteria for splitting classes may be used. One instance of the approach is obtained by considering bisimulation equivalence. On image-finite transition systems, this equivalence coincides with the equivalences induced (Section 2.4.1) by CTL as well as CTL*; see e.g. [BCG88] as well as the next chapter. In this context, a range of algorithms has been proposed [Hop71, CC82, PT87, KS90, BFH$^+$92, LY92, ACH$^+$92, YL93], each improving upon former ones in some way. Di erent instances of the idea consider variations like weak and branching bisimulation (the latter equivalence coincides with the equivalences induced by CTL*(U) and CTL(U); also see the next chapter), see [BCG88, GV90, FKM93]. Such algorithms may be used to reduce the size of a transition system while strongly preserving the corresponding logics. Below, we first briefly recall a simple version of the PRA for bisimulation, which underlies most of the mentioned algorithms. Then, we argue how we can find a criterion for splitting classes in the case that the considered equivalence is induced by a *fragment* of CTL*. To this means, we introduce the di erence between *logical* and *behavioural* equivalences.

Bisimulation equivalence of states can be defined in a concise way as follows. Let $\equiv$ be an equivalence relation on states, and consider its equivalence classes. Define the *successor classes* of a state to be the classes of its successors. Then $\equiv$ is a bisimulation if every class S is *stable*, meaning that all states in S agree on the truth of all literals and have the same successor classes. This simple characterisation forms the basis of the elegant PRA that computes the coarsest partitioning into classes of bisimilar states. The algorithm starts from the coarsest partitioning such that in every class, all states agree on the truth of all literals. Then, every class S that is not stable is split into maximal stable subclasses. Note that, as a result of this, a class containing predecessors of S may become unstable. This splitting is repeated until a fixpoint is reached, which can be shown to be the coarsest partitioning for which the classes are bisimulation equivalence classes (see Section 6.4).

As an example, consider the transition system in Figure 5.1. The s_i and t_i are names of states; p is a proposition. The splitting algorithm described above will start from the initial partitioning with classes $S_1 = \{s_1, s_2, s_3, t_1, t_2\}$ and $S_2 = \{t_3\}$ — the states where p holds and the states where $\neg p$ holds respectively. S_1 is not stable because t_2 has both S_1 and S_2 as successor classes, whereas the other elements of S_1 only have S_1 as successor class. So, in the first step, $\{t_2\}$ is split o S_1, resulting in the subclasses $S_{11} = \{s_1, s_2, s_3, t_1\}$ and $S_{12} = \{t_2\}$. Next, because S_{11} is not stable, it is split into $S_{111} = \{s_1\}$, $S_{112} = \{s_2, s_3\}$ and $S_{113} = \{t_1\}$. These subclasses

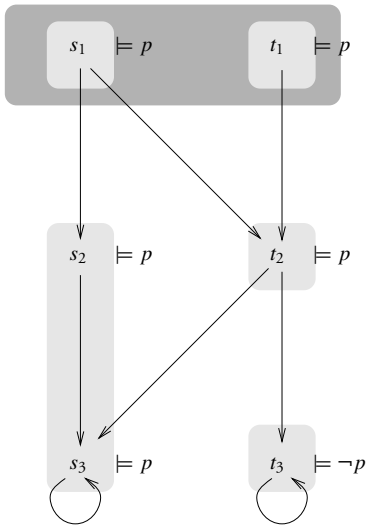


Figure 5.1: Concrete transition system.

can be computed by intersecting S_{11} with the preimage (*pre*, Definition 2.1.0.1) sets (and their complements) of all its successor classes. For example, S_{11} is intersected with $pre(S_{11}) = \{s_1, s_2, s_3, t_2\}$, $pre(S_{12}) = \{s_1, t_1\}$, and $pre(S_2) = \{t_2, t_3\}$. The first two intersections actually split S_{11} into the three parts mentioned above, the last intersection has no e ect. In the resulting partitioning, indicated by the light grey areas, all classes are stable, hence, the coarsest bisimulation classes have been identified.

Thus, the definition of bisimulation, which is in terms of the states and transitions of the underlying model, suggests a PRA for CTL-equivalence. Now suppose that we want to develop a PRA for the equivalence that is induced by some given *fragment* of CTL. Such an equivalence relation is defined as follows (cf. page 25):

> Two states s and t are equivalent i for every formula φ in the fragment,
> $s \models \varphi \Leftrightarrow t \models \varphi$.

Unlike the case for bisimulation, this definition is not directly in terms of the underlying model. In particular, it is not clear which criterion for splitting classes can be used. We call a definition in terms of the model, like the above definition of bisimulation, a *behavioural definition*, whereas a definition in terms of the formulae of a logic (fragment) is called a *logical definition*. Abusively, we will also refer to the equivalences being defined as behavioural and logical equivalences.

We now suggest how a logical definition may induce a criterion for splitting states. First, consider bisimulation (Definition 2.4.2.7) again. An alternative, logical definition of bisimulation is the following. Over image-finite transition systems it coincides with the original definition; see e.g. Lemmata 6.2.0.2 and 6.2.0.5.

> Two states s and t are bisimilar i for every formula φ in CTL, $s \models \varphi \Leftrightarrow t \models \varphi$.

The PRA for bisimulation presented above may alternatively be viewed as a process that *splits classes with respect to* CTL *formulae*: as long as there is a formula that can distinguish among the states within a class, the class should be split accordingly. From this viewpoint, the first splitting is based on the formula p — which holds in S_1 but not in S_2 — and the second splitting on $\forall \mathsf{X} p$ — which holds in S_{11} but not in S_{12}. The last splitting is actually based on two formulae: $\forall \mathsf{X} \neg \forall \mathsf{X} p$ and $\exists \mathsf{X} \neg \forall \mathsf{X} p$.

Our point is that the same method may be used in a PRA for a fragment of CTL: classes are split for each formula in the fragment. We illustrate this idea in Figure 5.1, considering the logical equivalence induced by ∀CTL. The criterion for splitting a state is now a ∀CTL formula. Thus, state t_3 is distinguished from the other states by formula p, resulting in a split into the same classes S_1 and S_2 as above. $\forall \mathsf{X} p$ distinguishes $\{s_1, s_2, s_3, t_1\}$ from $\{t_2\}$, e ecting a split of S_1 into S_{11} and S_{12}, again as above. Another ∀CTL formula is $\forall \mathsf{X} \forall \mathsf{X} p$; splitting for it causes S_{11} to split into $S'_{111} = \{s_1, t_1\}$ and $S'_{112} = \{s_2, s_3\}$, this time di erent from above. Now, there is no ∀CTL formula φ such that splitting for φ will cause any more changes. In particular, class S'_{111} (dark grey) is stable; indeed s_1 and t_1, although not bisimilar, cannot be distinguished by any ∀CTL formula. This algorithm will be worked out in Section 5.5.

The fragments of CTL* that we consider in this chapter, will indeed be fragments of ∀CTL. However, there are no theoretical obstacles in extending the results to the existential fragment (∃CTL), to CTL, or to the "starred" variants ∀CTL*, ∃CTL* or CTL* — see Section 5.8.

5.1.1 Overview of the chapter

After some preliminaries in the next section, we present in Section 5.3 the notion of a *companion* of a property set. In Chapter 3, Section 3.2.2, we have identified *consistency* of the description relation ρ with some property set L as a *necessary* condition for strong preservation. The notion of companion of a property set allows us to identify a *su cient* condition for strong preservation, namely, ρ has to be consistent with the companion of L. Consistency can be established by *splitting* the states of an abstract Kripke structure. A generic splitting algorithm is presented in Section 5.4, and is specialised to specific cases of property sets in Sections 5.5 and 5.6. The latter

section also contains an example. Section 5.7 discusses related work while Section 5.8 concludes.

5.2 Preliminaries

This section provides some preliminary material that is specific to this chapter.

5.2.0.1 C *We assume that a* finite *Kripke structure* $\mathcal{C} = (\Sigma, \mathrm{R}, \mathrm{I}, \|\cdot\|_{\mathrm{Lit}})$ *is given such that* R *is* total *(i.e. for every* $c \in \Sigma$ *there exists* $d \in \Sigma$ *such that* $\mathrm{R}(c, d)$*) and the set* Lit *(Section 2.3, page 18) is finite. Furthermore, we consider Abstract Kripke structures* $\mathcal{A}$ *of which only the (optimal) free transition relation* ${}_{\alpha}\mathrm{R}^{F}$ *(Definition 4.2.3.3) will be of interest; therefore we denote such structures by* $({}_{\alpha}\Sigma, {}_{\alpha}\mathrm{R}, {}_{\alpha}\mathrm{I}, {}_{\alpha}\|\cdot\|_{\mathrm{Lit}})$ *where* ${}_{\alpha}\mathrm{R}$ *denotes* ${}_{\alpha}\mathrm{R}^{F}$ *and* ${}_{\alpha}\mathrm{I}$ *and* ${}_{\alpha}\|\cdot\|_{\mathrm{Lit}}$ *are defined by Definitions 4.2.2.1 and 4.2.1.1 respectively.* ${}_{\alpha}\Sigma$ *and* Σ *are related by the Galois insertion* (α, γ) *from* $(\mathcal{P}(\Sigma), \subseteq)$ *to* $({}_{\alpha}\Sigma, \preceq)$*;* $\rho \subseteq \Sigma \times {}_{\alpha}\Sigma$ *is the corresponding description relation, i.e.* $\gamma = \mathit{pre}^{\bullet}_{\rho}$*.*

Pred is a set of predicates on Σ. We sometimes identify $p \in \mathit{Pred}$ with its characteristic set $\{c \in \Sigma \mid p(c)\}$, for example when writing $\mathit{pre}_{\mathrm{R}}(p)$. An alternative way to view this is by thinking of $\mathit{pre}_{\mathrm{R}}$ as a predicate transformer giving the weakest precondition such that it is possible to make a transition and get to a p-state (a state where p holds)[1]. We assume that *Pred* includes Lit (the set of propositions and their negations, page 18) and that it is closed under finite and infinite[2] conjunctions and disjunctions, and under $\widetilde{\mathit{pre}}$.

The *characteristic predicate* of a formula φ interpreted over $\mathcal{C}$ characterises the set of states in which φ holds. We can extend the definition of the interpretation function $\|\cdot\|_{\mathrm{Lit}}$ (Section 2.4, page 21) to ∀CTL, as follows.

5.2.0.2 D *The function* $\|\cdot\| : \forall\mathrm{CTL} \to \mathit{Pred}$*, which maps every formula to its* characteristic predicate*, is defined as follows. Let* $p \in \mathrm{Lit}$ *and* $\varphi_1, \varphi_2 \in \forall\mathrm{CTL}$*.*

$$\|p\| = \|p\|_{\mathrm{Lit}} \qquad \|\forall\mathsf{X}\varphi_1\| = \widetilde{\mathit{pre}}_{\mathrm{R}}(\|\varphi_1\|)$$

$$\|\varphi_1 \wedge \varphi_2\| = \|\varphi_1\| \wedge \|\varphi_2\| \qquad \|\forall\mathsf{U}(\varphi_1, \varphi_2)\| = \bigvee_{i\in\mathbb{N}} \|\forall\mathsf{U}_i(\varphi_1, \varphi_2)\|$$

$$\|\varphi_1 \vee \varphi_2\| = \|\varphi_1\| \vee \|\varphi_2\| \qquad \|\forall\mathsf{V}(\varphi_1, \varphi_2)\| = \bigwedge_{i\in\mathbb{N}} \|\forall\mathsf{V}_i(\varphi_1, \varphi_2)\|$$

[1] This is not the same predicate transformer as Dijkstra's weakest precondition wp or weakest liberal precondition wlp. For example, wp(R, p) characterises the set of states from which *any* R-transition (terminates and) leads to a p-state. When R is total, as we assume, then wp corresponds to $\widetilde{\mathit{pre}}$.

[2] Closure under infinite conjunctions and disjunctions is used in Definition 5.2.0.2.

The first point of the following lemma implies that on $\mathcal{C}$, every $\forall\mathsf{U}/\forall\mathsf{V}$-formula is equivalent to some finite approximant, meaning that every such formula can be rewritten into an equivalent form without $\forall\mathsf{U}/\forall\mathsf{V}$-operators. The second point states that the characteristic predicates of $\forall$CTL-formulae as defined in Definition 5.2.0.2 above agree with the interpretation of formulae as captured by $\models$. Finally, point 3 expresses that, on $\mathcal{C}$, the computation of the sequence $\{\bigvee_i \|\forall\mathsf{U}_i(\varphi_1, \varphi_2)\|\}_i$ stabilises once two subsequent values are the same, and that this value equals $\|\forall\mathsf{U}(\varphi_1, \varphi_2)\|$.

5.2.0.3 L *Let $\varphi, \varphi_1, \varphi_2 \in \forall$CTL, and recall Convention 5.2.0.1.*

1. *There exists $k \in \mathbb{N}$ such that $\mathcal{C} \models \forall\mathsf{U}(\varphi_1, \varphi_2) \equiv \forall\mathsf{U}_k(\varphi_1, \varphi_2)$, and similarly for $\forall\mathsf{V}$.*

2. *For every $c \in \Sigma$, $c \models \varphi$ if and only if $c \in \|\varphi\|$.*

3. *Let $i \in \mathbb{N}$. If $\mathcal{C} \models \forall\mathsf{U}_i(\varphi_1, \varphi_2) \equiv \forall\mathsf{U}_{i+1}(\varphi_1, \varphi_2)$, then $\forall_{j\geq 0}\, \mathcal{C} \models \forall\mathsf{U}_i(\varphi_1, \varphi_2) \equiv \forall\mathsf{U}_{i+j}(\varphi_1, \varphi_2) \equiv \forall\mathsf{U}(\varphi_1, \varphi_2)$.*

P .

1. Let $k = |\Sigma|$. We show that for every $c \in \Sigma$, (*) $c \models \forall\mathsf{U}(\varphi_1, \varphi_2) \Leftrightarrow c \models \forall\mathsf{U}_k(\varphi_1, \varphi_2)$, which implies the first point of the lemma (the $\forall\mathsf{V}$-case is similar). By induction on k it is easily shown that the $\Leftarrow$-direction in (*) holds for every k. Therefore, we only have to show that the $\Rightarrow$-direction holds. Assume that $c \models \forall\mathsf{U}(\varphi_1, \varphi_2)$. By Definition 2.4.1.1 of $\forall\mathsf{U}(\varphi_1, \varphi_2)$ this means that for every c-path π, we can choose $n_\pi \in \mathbb{N}$ as the smallest number such that $\pi^{n_\pi} \models \varphi_2$ and for all $i < n_\pi$, $\pi^i \models \varphi_1$. Let π be a c-path and consider the prefix $\pi_{[0,k]}$. Because, by our choice of k, the number of occurrences of states on this prefix is one larger than the total number of states of $\mathcal{C}$, there must be a cycle in this prefix. Therefore, the point n_π at which φ_2 is first fulfilled must be smaller than k, otherwise there would be a c-path π' for which $\pi'(n) \not\models \varphi_2$ for every $n \in \mathbb{N}$ and hence $\pi' \not\models \forall\mathsf{U}(\varphi_1, \varphi_2)$, namely the path π' that starts like π does, but keeps looping in the cycle. By Lemma 2.4.1.3 it now follows that $c \models \forall\mathsf{U}_k(\varphi_1, \varphi_2)$.

2. Using point 1, rewrite φ into an equivalent form without $\forall\mathsf{U}/\forall\mathsf{V}$-operators. Then use induction on the structure of φ.

3. Using the definition of $\mathsf{U}_{i+1}(\varphi_1, \varphi_2)$, it is easy to define a function $\mathcal{F} : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ such that for every $\varphi_1, \varphi_2 \in \forall$CTL and every i, the set $\|\forall\mathsf{U}_{i+1}(\varphi_1, \varphi_2)\|$ of states satisfying $\forall\mathsf{U}_{i+1}(\varphi_1, \varphi_2)$ (see point 2) equals $\mathcal{F}(\|\forall\mathsf{U}_i(\varphi_1, \varphi_2)\|)$. Hence, if $\mathcal{C} \models \forall\mathsf{U}_i(\varphi_1, \varphi_2) \equiv \forall\mathsf{U}_{i+1}(\varphi_1, \varphi_2)$ for some $i \in \mathbb{N}$, then $\|\forall\mathsf{U}_i(\varphi_1, \varphi_2)\|$ is a fixpoint of $\mathcal{F}$, from which it follows that $\|\forall\mathsf{U}_{i+j}(\varphi_1, \varphi_2)\| = \|\forall\mathsf{U}_i(\varphi_1, \varphi_2)\|$ for every $j \geq 0$; and therefore, by point 2, $\mathcal{C} \models \forall\mathsf{U}_{i+j}(\varphi_1, \varphi_2) \equiv \forall\mathsf{U}_i(\varphi_1, \varphi_2)$ for every $j \geq 0$. By point 1 we now also have $\mathcal{C} \models \forall\mathsf{U}_i(\varphi_1, \varphi_2) \equiv \forall\mathsf{U}(\varphi_1, \varphi_2)$. □

In this chapter, we will consider Abstract Kripke structures whose states always describe sets of concrete states that can be expressed as (characteristic sets of) predicates in *Pred*. It turns out to be convenient to identify abstract states with their concretisations, i.e. an abstract state a with $\gamma(a) = p$ ($p \in$ *Pred*) will be denoted by p, in which case $c \in p$ will stand for $c \in \gamma(a)$. As a consequence, the description relation ρ becomes implicit in such cases.

5.3 Companions

Our goal is to determine conditions for the strong preservation of a given property set $\mathrm{L} \subseteq \forall\mathrm{CTL}$. For a description relation ξ on the level of Kripke structures:

$$\xi(\mathcal{C}, \mathcal{A}) \;\Rightarrow\; \forall_{\varphi\in\mathrm{L}}\, [\mathcal{C} \models \varphi \;\Leftrightarrow\; \mathcal{A} \models \varphi] \tag{5.1}$$

We sharpen this requirement to the level of individual states.

5.3.0.1 D *For $a \in {}_\alpha\Sigma$ and $\varphi \in \forall\mathrm{CTL}$, ρ* strongly preserves *φ in a (cf. Section 3.2.2) i*

$$\forall_{c\in\Sigma}\, [\rho(c, a) \;\Rightarrow\; [c \models \varphi \;\Leftrightarrow\; a \models \varphi]]. \tag{5.2}$$

For $\mathrm{L} \subseteq \forall\mathrm{CTL}$ and $A \subseteq {}_\alpha\Sigma$, ρ strongly preserves L in *A i it strongly preserves every $\varphi \in \mathrm{L}$ in every $a \in A$. For singletons L, A we usually omit set brackets. ρ* strongly preserves L *abbreviates ρ strongly preserves* L *in ${}_\alpha\Sigma$.*

In Section 3.2.2, we saw that a necessary condition for strong preservation is that ρ be *consistent* with L. We recall here Definition 3.2.2.2 of consistency (page 50), specialised for the case of abstract states.

5.3.0.2 D *Let $\mathrm{L} \subseteq \forall\mathrm{CTL}$ and $A \subseteq {}_\alpha\Sigma$. ρ is* consistent with L in *A i for every $\varphi \in \mathrm{L}$ and every $a \in A$, $\forall_{c\in\Sigma}\, [\rho(c, a) \;\Rightarrow\; c \models \varphi]$ or $\forall_{c\in\Sigma}\, [\rho(c, a) \;\Rightarrow\; c \not\models \varphi]$. Again, we omit set brackets for singletons.*

However, weak preservation plus consistency together are not yet a su cient condition for strong preservation, as is illustrated by Figure 5.2. We consider a situation where $\Sigma = \{c_1, c_2\}$ and ${}_\alpha\Sigma = \{a\}$ with $\rho(c_1, a)$ and $\rho(c_2, a)$. In (both parts of) the figure, as well as in all other figures in the remainder, an abstract state is indicated by a rectangular box in such a way that all concrete states described by the abstract state lie inside the box. Concrete transitions in R are indicated by thin arrows, while the thick arrows denote the abstract transition relation ${}_\alpha\mathrm{R}$.

In Figure 5.2a, we consider a singleton property set $\mathrm{L} = \{p \vee q\}$. Although both concrete states in a satisfy $p \vee q$, i.e. a is consistent with L, a itself does not

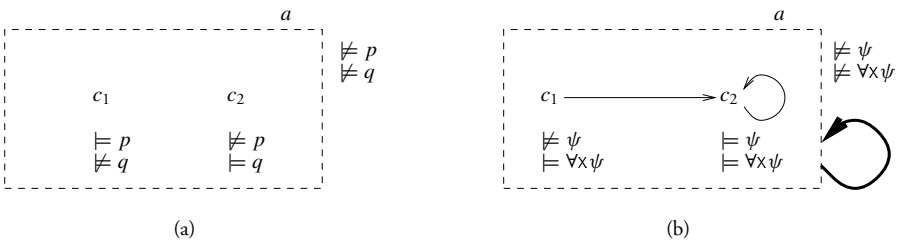


Figure 5.2: Abstraction is consistent but not strongly preserving.

satisfy this formula (see Definition 4.2.1.1 of $_\alpha\|\cdot\|_{\mathrm{Lit}}$), i.e. it is not strongly preserved. In part b of the figure, $\mathrm{L} = \{\forall \mathsf{X}\varphi\}$ where φ is some $\forall$CTL formula. Again, ρ is clearly consistent with $\{\forall \mathsf{X}\varphi\}$, however, it does not strongly preserve $\{\forall \mathsf{X}\varphi\}$. In both situations, the point seems to be that ρ is not consistent with the subformulae in L.

Searching for su cient conditions on subformulae, we find the following result.

5.3.0.3 L *Let $a \in {}_\alpha\Sigma$, $p \in \mathrm{Lit}$ and $\varphi, \varphi_1, \varphi_2 \in \forall\mathrm{CTL}$.*

1. *ρ strongly preserves p in a i it is consistent with p in a.*
2. *If ρ strongly preserves φ_1 in a and strongly preserves φ_2 in a, then it strongly preserves $\varphi_1 \vee \varphi_2$ in a.*
3. *If ρ strongly preserves φ_1 in a and strongly preserves φ_2 in a, then it strongly preserves $\varphi_1 \wedge \varphi_2$ in a.*
4. *If ρ is consistent with $\forall \mathsf{X}\varphi$ in a and strongly preserves φ in every ($_\alpha$R-)successor of a, then it strongly preserves $\forall \mathsf{X}\varphi$ in a.*

P . As the states in $_\alpha\Sigma$ do already (weakly) preserve $\forall$CTL, for strong preservation we only have to show the $\Rightarrow$ direction of the biimplication in 5.2 on page 122. Recall that $c \in \gamma(a)$ i $\rho(c, a)$.

1. Directly from Definition 5.3.0.2 of consistency and Definition 4.2.1.1 of $_\alpha\|\cdot\|_{\mathrm{Lit}}$.
2. We use contraposition. Suppose $a \not\models \varphi_1 \vee \varphi_2$. By definition of $\models$, this is equivalent to $a \not\models \varphi_1$ and $a \not\models \varphi_2$. Because ρ strongly preserves φ_1 in a and strongly preserves φ_2 in a, this implies that $\forall_{c\in\gamma(a)}\ c \not\models \varphi_1$ and $\forall_{c\in\gamma(a)}\ c \not\models \varphi_2$, i.e. $\forall_{c\in\gamma(a)}\ c \not\models \varphi_1 \vee \varphi_2$.
3. Similar to the previous case.

4. Suppose that $a \not\models \forall X\varphi$, i.e. we can choose an ${}_{\alpha}\mathrm{R}$-successor b of a such that $b \not\models \varphi$. By definition of ${}_{\alpha}\mathrm{R}$, which is equal (see Convention 5.2.0.1) to ${}_{\alpha}\mathrm{R}^{F}$, defined in Definition 4.2.3.3, we can choose $c \in \gamma(a)$ and $d \in \gamma(b)$ such that $\mathrm{R}(c, d)$. Because ρ strongly preserves φ in b and $b \not\models \varphi$, we have $d \not\models \varphi$. So $c \not\models \forall X\varphi$. Because ρ is consistent with $\forall X\varphi$ in a, it must be that $c' \not\models \forall X\varphi$ for every $c' \in \gamma(a)$. □

Note that this lemma only gives *su cient* conditions for strong preservation, except point 1, where the condition is necessary as well. In search for the weakest possible conditions, it turns out that these depend, in each of the cases, on the specific form of the abstract model $\mathcal{A}$, as we will now show.

In point 2, where the disjunction is considered, we observe the following. If $a \not\models \varphi_1 \vee \varphi_2$, then the given condition is the weakest possible — i.e. it is necessary as well, as can easily be seen. However, when $a \models \varphi_1 \vee \varphi_2$, the fact that ρ strongly preserves $\varphi_1 \vee \varphi_2$ in a does not imply anything about the strong preservation of the subformulae φ_1 and φ_2. Thus, point 2 of Lemma 5.3.0.3 may be strengthened to: "ρ strongly preserves $\varphi_1 \vee \varphi_2$ in a if and only if [$a \models \varphi_1 \vee \varphi_2$] or [$\rho$ strongly preserves φ_1 in a and strongly preserves φ_2 in a]".

For point 3, the conjunction, the problem is di erent. First, notice that when $a \models \varphi_1 \wedge \varphi_2$, then strong preservation of $\varphi_1 \wedge \varphi_2$ *does* imply strong preservation of both φ_1 and φ_2. Now consider the case that $a \not\models \varphi_1 \wedge \varphi_2$, i.e. we have $a \not\models \varphi_1$ or $a \not\models \varphi_2$. We consider three subcases.

- In case that $a \models \varphi_1$ — and hence necessarily $a \not\models \varphi_2$ — we have by weak preservation that $c \models \varphi_1$ for every $c \in \gamma(a)$. If $\varphi_1 \wedge \varphi_2$ is to be strongly preserved, then we must furthermore have that $c \not\models \varphi_1 \wedge \varphi_2$ for every such c, which can only be the case if $c \not\models \varphi_2$ for every c, implying that φ_2 is strongly preserved by a. Furthermore, note that also φ_1 is strongly preserved as $a \models \varphi_1$.

- A symmetrical argument holds for the case that $a \models \varphi_2$ and $a \not\models \varphi_1$.

- However, it may also be the case that $a \not\models \varphi_1$ *and* $a \not\models \varphi_2$. In this case, strong preservation of both formulae is not needed in order to have strong preservation of their conjunction. In fact, neither of them needs to be strongly preserved, as illustrated by Figure 5.3. The necessary and su cient condition in this case is that for every $c \in \gamma(a)$ we have $c \not\models \varphi_1$ or $c \not\models \varphi_2$. This condition is not satisfactory as it cannot be expressed in terms of strong preservation or consistency of φ_1 and/or φ_2 by a. A slightly stronger condition that *is* satifying in this respect is that at least one of them should be strongly preserved.

Thus, point 3 of Lemma 5.3.0.3 may be strengthened to: "if $a \models \varphi_1 \wedge \varphi_2$ then: ρ strongly preserves $\varphi_1 \wedge \varphi_2$ in a if and only if it both strongly preserves φ_1 in a and

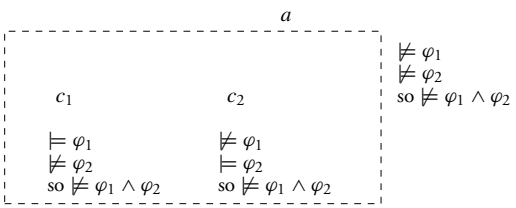


Figure 5.3: $\varphi_1 \wedge \varphi_2$ strongly preserved while neither conjunct is.

strongly preserves φ_2 in a; otherwise, ρ strongly preserves $\varphi_1 \wedge \varphi_2$ in a if it strongly preserves φ_1 in a or strongly preserves φ_2 in a".

Also for point 4 similar considerations may be made.

We went into this bit of hairsplitting to show that we cannot weaken the conditions given by the lemma without getting dependent on $\mathcal{A}$. As we would like to develop a method for the construction of strongly preserving abstract models regardless of their specific form, we use the results of Lemma 5.3.0.3 instead of the stronger facts discussed above.

Furthermore, we wish to develop a method that is *global* in the sense that it establishes things like consistency and strong preservation for all states of the abstract model, instead of considering it per state. For this reason, the following definition does not involve the individual abstract states that strong preservation or consistency has to hold for. It defines the *companion comp*(L) of L as a set of formulae such that consistency of ρ with *comp*(L) is a su cient condition for ρ to strongly preserve L. Note how the points 1, 2, 3 and 4 are inspired by the corresponding points from Lemma 5.3.0.3 above. Points 5 and 6 can be understood from Lemma 5.2.0.3.

5.3.0.4 D *Let* $\mathrm{L} \subseteq \forall\mathrm{CTL}$. *The function comp* : $\mathrm{L} \to \mathcal{P}(\forall\mathrm{CTL})$ *is defined inductively as follows. Let* $p \in \mathrm{Lit}$ *and* $\varphi_1, \varphi_2 \in \forall\mathrm{CTL}$.

1. $comp(p) = \{p\}$
2. $comp(\varphi_1 \wedge \varphi_2) = comp(\varphi_1) \cup comp(\varphi_2)$
3. $comp(\varphi_1 \vee \varphi_2) = comp(\varphi_1) \cup comp(\varphi_2)$
4. $comp(\forall \mathsf{X}\varphi) = \{\forall \mathsf{X}\varphi\} \cup comp(\varphi)$
5. $comp(\forall \mathsf{U}(\varphi_1, \varphi_2)) = \bigcup\{comp(\forall \mathsf{U}_i(\varphi_1, \varphi_2)) \mid i \in \mathbb{N}\}$
6. $comp(\forall \mathsf{V}(\varphi_1, \varphi_2)) = \bigcup\{comp(\forall \mathsf{V}_i(\varphi_1, \varphi_2)) \mid i \in \mathbb{N}\}$

comp is extended to sets of formulae by $comp(\Phi) = \bigcup\{comp(\varphi) \mid \varphi \in \Phi\}$.

Note that formulae in a companion may still contain $\forall\mathsf{U}/\forall\mathsf{V}$ operators (without the i subscripts).

5.3.0.5 E *The companion of the formula* $p \to \forall\mathsf{U}(q, \forall\mathsf{U}(r, s))$ *contains the formulae* $\neg p$, *false*, q, r, s *as well as* $\forall\mathsf{X}$*false*, $\forall\mathsf{X}(s \vee (r \wedge \forall\mathsf{X}$*false*$))$, $\forall\mathsf{X}(s \vee (r \wedge \forall\mathsf{X}(s \vee (r \wedge \forall\mathsf{X}$*false*$))))$, ... *and* $\forall\mathsf{X}(\forall\mathsf{U}(r, s) \vee (q \wedge \forall\mathsf{X}$*false*$))$, $\forall\mathsf{X}(\forall\mathsf{U}(r, s) \vee (q \wedge \forall\mathsf{X}(\forall\mathsf{U}(r, s) \vee (q \wedge \forall\mathsf{X}$*false*$))))$, ...

The following lemma justifies this definition.

5.3.0.6 L *Let* $\mathrm{L} \subseteq \forall\mathrm{CTL}$ *be a property set. If* ρ *is consistent with* $comp(\mathrm{L})$, *then* L *is strongly preserved (in every* $a \in {}_\alpha\Sigma$*).*

P Using Lemmata 5.3.0.3 and 5.2.0.3. □

In the next sections, we develop algorithms that establish consistency by *splitting* abstract states.

5.4 A Generic Splitting Algorithm

In the sequel we often write "a is consistent with φ" when we mean "ρ is consistent with φ in a".

Consistency with φ in abstract state a can be established by "splitting" a into a_1 and a_2 (and adapting ρ) in such a way that $\forall_{c\in\Sigma}\ [\rho(c, a_1) \ \Rightarrow\ c \models \varphi]$ and $\forall_{c\in\Sigma}\ [\rho(c, a_2) \ \Rightarrow\ c \not\models \varphi]$.

5.4.0.1 D *Let* $A \subseteq {}_\alpha\Sigma$ *and* $\varphi \in \forall\mathrm{CTL}$. $split(A, \varphi) = \{a \wedge \|\varphi\|, a \wedge \|\neg\varphi\| \mid a \in A\}$. $split(a, \varphi)$ *abbreviates* $split(\{a\}, \varphi)$. *For* $p = \|\varphi\|$, *we sometimes write* $split(A, p)$ *for* $split(A, \varphi)$.

Obviously, we have the following

5.4.0.2 L *Every* $a \in split(A, \varphi)$ *is consistent with* φ.

The simple generic algorithm of Figure 5.4 constructs a strongly preserving abstraction (for L) by successively splitting abstract states. Initially, the abstract model consists of the single abstract state *true*, describing all concrete states. The abstract model is repeatedly refined by choosing a formula from the companion of L and splitting the states with respect to this formula. Computation of the transition relation ${}_\alpha\mathrm{R}$, initial states ${}_\alpha\mathrm{I}$ and interpretation function ${}_\alpha\|\cdot\|_{\mathrm{Lit}}$ for the literals is specified by the following functions.

$_\alpha\Sigma := \{\mathit{true}\}$;
$_\alpha\mathrm{R} := \mathit{rel}(_\alpha\Sigma)$; $_\alpha\mathrm{I} := \mathit{init}(_\alpha\Sigma)$; $_\alpha\|\cdot\|_{\mathrm{Lit}} := \mathit{intp}(_\alpha\Sigma)$;
ready := *false*;
while not *ready* **do**
choose $\varphi \in \mathit{comp}(\mathrm{L})$;
$_\alpha\Sigma := \mathit{split}(_\alpha\Sigma, \varphi)$;
$_\alpha\mathrm{R} := \mathit{rel}(_\alpha\Sigma)$; $_\alpha\mathrm{I} := \mathit{init}(_\alpha\Sigma)$; $_\alpha\|\cdot\|_{\mathrm{Lit}} := \mathit{intp}(_\alpha\Sigma)$;
update *ready*
od;

Figure 5.4: Generic splitting algorithm.

5.4.0.3 D *The functions* $\mathit{rel} : \mathcal{P}(_\alpha\Sigma) \to \mathcal{P}(_\alpha\Sigma \times _\alpha\Sigma)$, $\mathit{init} : \mathcal{P}(_\alpha\Sigma) \to \mathcal{P}(_\alpha\Sigma)$ *and* $\mathit{intp} : \mathcal{P}(_\alpha\Sigma) \to (\mathrm{Lit} \to \mathcal{P}(_\alpha\Sigma))$ *are defined as follows. (Note that these functions are defined relative to the concrete transition system* $\mathcal{C} = (\Sigma, \mathrm{R}, \mathrm{I}, \|\cdot\|_{\mathrm{Lit}})$*.) For* $A \subseteq _\alpha\Sigma$*,*

- $\mathit{rel}(A) = \{(a, b) \in A \times A \mid \exists_{c\in a, d\in b}\ \mathrm{R}(c, d)\}$.
- $\mathit{init}(A) = \{a \in A \mid \exists_{c\in a}\ c \in \mathrm{I}\}$.
- $\mathit{intp}(A) = \lambda_{p\in\mathrm{Lit}}.\{a \in A \mid a \subseteq \|p\|_{\mathrm{Lit}}\}$.

With these definitions of the abstract transition relation, initial states and literal interpretation, we have weak preservation. Because all abstract states are "disjoint" (i.e. for any two abstract states a and b we have $\gamma(a) \cap \gamma(b) = \emptyset$), $\mathit{rel}(A)$ is equal to the free abstract transition relation $_\alpha\mathrm{R}^F$. Also, $\mathit{init}(A)$ and $\mathit{intp}(A)$ correspond to $_\alpha\mathrm{I}$ and $_\alpha\|\cdot\|_{\mathrm{Lit}}$ as defined in Chapter 4.

The variable *ready* controls the termination of the algorithm. At this generic level it is not further specified. Possible conditions for setting it to *true* include the following.

1. All formulae from $\mathit{comp}(\mathrm{L})$ have been chosen to split $_\alpha\Sigma$. In this case, $_\alpha\Sigma$ is consistent with $\mathit{comp}(\mathrm{L})$ by Lemma 5.4.0.2, and from Lemma 5.3.0.6 it then follows that L is strongly preserved, in which case both the positive and the negative results of abstract model checking will carry over to the concrete model.

2. A model check of the property of interest has been done and the result is positive. Because the abstract model being constructed is weakly preserving (after every step of the while loop), positive results carry over to the concrete model.

3. Although not all formulae from *comp*(L) have been explicitly treated, it may be possible to determine stabilisation of the abstract model, meaning that no subsequent split operations will cause changes. When *comp*(L) is an infinite set, this approach must be used. Also in this case, L is strongly preserved after stabilisation.

We can already make an optimisation to this generic algorithm. As properties are interpreted in the initial states, unreachable states may be safely omitted from the abstract model. Formally, this observation is justified by the following lemma, which says that unreachability of abstract states is preserved under splitting.

5.4.0.4 L *Let* $\mathcal{A} = ({}_\alpha\Sigma, \mathit{rel}({}_\alpha\Sigma), \mathit{init}({}_\alpha\Sigma), \mathit{intp}({}_\alpha\Sigma))$, $a \in {}_\alpha\Sigma$ *and* $\mathrm{L} \subseteq \forall\mathrm{CTL}$. *If* a *is unreachable in* $\mathcal{A}$, *then: for every* $\varphi \in \mathrm{L}$, *if* $\mathcal{A}' = ({}_\alpha\Sigma', \mathit{rel}({}_\alpha\Sigma'), \mathit{init}({}_\alpha\Sigma'), \mathit{intp}({}_\alpha\Sigma'))$ *where* ${}_\alpha\Sigma' = \mathit{split}({}_\alpha\Sigma, \varphi)$, *then every* $a' \in \mathit{split}(a, \varphi)$ *is unreachable in* $\mathcal{A}'$.

P Suppose that a' is reachable in $\mathcal{A}'$. By an induction on the length of the path from an initial state to a', it is easily shown that a contradiction follows. □

So, unreachable abstract states may be ignored in the verification of a property and may be removed from the constructed abstract model. This provides us with a method to optimise the expected running-time and memory demands of the splitting algorithm: the unreachable states do not have to be kept in memory, and no splitting has to be performed on them. Although the worst-case complexity will remain the same, in practice we may expect a considerable gain. Of course, the price to be paid is the repeated recomputation of reachability information.

Another point that is worth noting is that the algorithm may be started on any set of abstract states — it does not necessarily have to be $\{\mathit{true}\}$. For example, it may be invoked on an Abstract Kripke structured obtained by the abstract interpretation techniques presented in the previous chapter, when it turns out that the chosen level of abstraction is too coarse. The partition refinement algorithm then performs a further refinement of the abstract domain.

Instantiations of this algorithm for specific choices of L will have to compute the characteristic predicates of all formulae in *comp*(L), which is usually an expensive operation. Because the characteristic predicates are defined inductively on the structure

of formulae, practical instantiations of the algorithm should make use of a stratification of *comp*(L) where each higher stratum is defined in terms of formulae from the previous. If such a stratification exists, the order in which formulae for splitting are chosen from *comp*(L) should follow these strata, so that characteristic predicates can be computed inductively. In the following sections, we will exemplify this by specialising the algorithm for two cases in which such stratifications are easily given, namely $\mathrm{L} = \forall\mathrm{CTL}$ and $\mathrm{L} = \{\varphi\}$ for some $\varphi \in \forall\mathrm{CTL}$. The latter case easily generalises to any finite property set in ∀CTL (where new chances for optimisation may occur).

5.5 Splitting for ∀CTL

In this section we consider the case $\mathrm{L} = \forall\mathrm{CTL}$. Because the complexity of the splitting algorithm depends on the size of *comp*(∀CTL), we first try to minimise the set of formulae that we have to split for. We start with the definition of a fragment of ∀CTL containing only formulae that are built from literals, disjunctions and $\forall\mathsf{X}$ operators in some restricted way. For a finite set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ of formulae, $\bigvee \Phi$ denotes the formula $\varphi_1 \vee \ldots \vee \varphi_n$.

5.5.0.1 D *The logic* $\forall\mathrm{CTL}^-$ *is the set of state formulae given by the following inductive definition.*

1. *Every* $p \in \mathrm{Lit}$ *is a* $\forall\mathrm{CTL}^-$ *formula.*
2. *If* Φ *is a finite set of* $\forall\mathrm{CTL}^-$ *formulae, then* $\forall\mathsf{X}(\bigvee \Phi)$ *is a* $\forall\mathrm{CTL}^-$ *formula.*

5.5.0.2 L *Let* $A \subseteq {}_\alpha\Sigma$. *If* A *is consistent with* $\forall\mathrm{CTL}^-$, *then it is consistent with* *comp*(∀CTL).

P We show that every formula in *comp*(∀CTL) is equivalent, on $\mathcal{C}$ (recall Convention 5.2.0.1 in which it was assumed that $\mathcal{C}$, of which $\mathcal{A} = ({}_\alpha\Sigma, {}_\alpha\mathrm{R}, {}_\alpha\mathrm{I}, {}_\alpha\|\cdot\|_{\mathrm{Lit}})$ is an abstraction, is finite), with a formula in $\forall\mathrm{CTL}^-$. Let $\varphi \in$ *comp*(∀CTL). From Definition 5.3.0.4 it is easily shown that then $\varphi \in \mathrm{Lit}$ or it is of the form $\forall\mathsf{X}\varphi'$ where φ' is an arbitrary ∀CTL formula. The first case is obvious. Otherwise, by Lemma 5.2.0.3, point 1, φ' is equivalent to an ∀CTL formula φ'' without $\forall\mathsf{U}$ and $\forall\mathsf{V}$ operators. By induction on the level of φ'' it can now easily be shown that φ'' is equivalent to a $\forall\mathrm{CTL}^-$ formula. □

The definition of $\forall\mathrm{CTL}^-$ suggests the following stratification.

5.5.0.3 D

1. $\forall\mathrm{CTL}^-_0 = \mathrm{Lit}$.

2. $\forall\mathrm{CTL}^-_{i+1} = \{\forall\mathsf{X}(\bigvee\Phi) \mid \Phi \subseteq \bigcup_{0\leq j\leq i}\forall\mathrm{CTL}^-_j\} \setminus \bigcup_{0\leq j\leq i}\forall\mathrm{CTL}^-_j$.

Note that $\forall\mathrm{CTL}^-_i$ is the subset of $\forall\mathrm{CTL}^-$ consisting of all formulae of level i (Section 2.3, page 20).

The computation of characteristic predicates by the algorithm can be done inductively along the lines of this definition. For the formulae in each higher stratum ($\forall\mathrm{CTL}^-_{i+1}$), the characteristic predicates can be computed by taking the $\widetilde{pre}$ of finite disjunctions of the characteristic predicates of formulae in all previous strata ($\bigcup_{0\leq j\leq i}\forall\mathrm{CTL}^-_j$), except those that have been computed in a lower stratum already. We arrive at the instantiation of the generic splitting algorithm given by Figure 5.5. Variable i is auxiliary and only used in the annotations. Note that in the annotations we confuse (for readability's sake) between formulae and their characteristic predicates. Comparing this to Figure 5.4, the order in which the formulae from *comp*(L) are selected has been straightened out into an initial part where all the literals are dealt with and a while-loop that iterates along the sets $\forall\mathrm{CTL}^-_{i+1}$.

Next, we focus on the termination condition of this algorithm.

5.5.1 Termination

As the underlying concrete model is finite, the abstract model that is constructed by the algorithm will eventually become stable, meaning that for all $\varphi \in \forall\mathrm{CTL}^-$, $\mathit{split}({}_\alpha\Sigma, \varphi) = {}_\alpha\Sigma$. The question is how we can detect this situation. We cannot take $\mathit{ready} := (P = P')$ as a criterion: the program will then never terminate as every stratum contains new formulae. In similar partition refinement algorithms, e.g. the algorithm of [BFH$^+$92] that splits for CTL, or the algorithms for preorder checking in [CC95], *one-step-stability* is used as a termination criterion: if the constructed model (or, relation in [CC95]) remains unchanged during one step of the while loop, it may be concluded that no subsequent steps will cause changes anymore. The technical reason that this works is that there is a function $\mathcal{F}$ such that for every i, the model ${}_\alpha\Sigma_{i+1}$ that is obtained in the ith step, by splitting the states of ${}_\alpha\Sigma_i$, is equal to $\mathcal{F}({}_\alpha\Sigma_i)$. Hence, if the constructed model remains stable during one step, i.e. if ${}_\alpha\Sigma_{i+1} = {}_\alpha\Sigma_i$, then ${}_\alpha\Sigma_i$ is a fixpoint of $\mathcal{F}$. Obviously, we then have ${}_\alpha\Sigma_{i+j} = {}_\alpha\Sigma_i$ for any $j \geq 0$ (cf. the proof of Lemma 2.4.2.5). This is not the case for our algorithm, because the way in which ${}_\alpha\Sigma_{i+1}$ is obtained from ${}_\alpha\Sigma_i$ is by splitting with respect to a set of formulae that is di erent for every i. As a result, it may happen that the abstract model remains unchanged during one or more steps but changes again in some later step. The example depicted in Figure 5.6 illustrates this. The propositions p, q, r, s, t and u are assumed to be mutually exclusive. The partitionings after every step of the while loop are indicated by the rectangular blocks in the figure. The outermost rectangle gives the initial configuration, where the only abstract state is

{ initialise: }
${}_\alpha\Sigma := \{true\}$;
${}_\alpha\mathrm{R} := rel({}_\alpha\Sigma)$; ${}_\alpha\mathrm{I} := init({}_\alpha\Sigma)$; ${}_\alpha\|\cdot\|_{\mathrm{Lit}} := intp({}_\alpha\Sigma)$;
ready := *false*;
{ initial split for literals: }
for each $\varphi \in \mathrm{Lit}$ **do** ${}_\alpha\Sigma := split({}_\alpha\Sigma, \varphi)$ **od**;
$P :=$ Lit;
${}_\alpha\mathrm{R} := rel({}_\alpha\Sigma)$; ${}_\alpha\mathrm{I} := init({}_\alpha\Sigma)$; ${}_\alpha\|\cdot\|_{\mathrm{Lit}} := intp({}_\alpha\Sigma)$;
update *ready*;
$i := 0$;
{ iteratively split for formulae in ith stratum: }
while not *ready* **do**
    $\{\ P = \bigcup_{0 \le j \le i} \forall\mathrm{CTL}^-_j\ \}$
    $P' := \{\widetilde{pre}_{\mathrm{R}}(\bigvee Q) \mid Q \subseteq P\} \setminus P$;
    $\{\ P' = \forall\mathrm{CTL}^-_{i+1}\ \}$
    for each $p \in P'$ **do**
        ${}_\alpha\Sigma := split({}_\alpha\Sigma, p)$
    od;
    ${}_\alpha\mathrm{R} := rel({}_\alpha\Sigma)$; ${}_\alpha\mathrm{I} := init({}_\alpha\Sigma)$; ${}_\alpha\|\cdot\|_{\mathrm{Lit}} := intp({}_\alpha\Sigma)$;
    update *ready*;
    $P := P \cup P'$;
    $i := i + 1$
od;

Figure 5.5: Splitting algorithm for ∀CTL without termination.

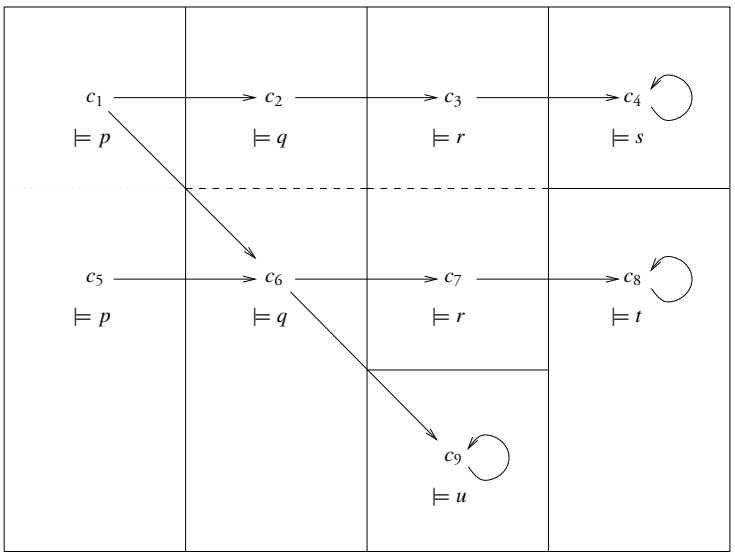


Figure 5.6: One-step-stability $\neq$ stability.

true (i.e. all concrete states belong to the same abstract state). The solid lines indicate the situation after splitting for the literals: the p-, q- and r-states remain pairwise together. In the first step of the while loop, the splittings indicated by the dashed lines are e ected. For example, a distinguishing formula of level 1 for the two q-states is $\forall \mathsf{X} r$, for the r-states it is $\forall \mathsf{X} s$. The second step of the loop does *not* change the abstract model: there is no $\forall$CTL$^-$ formula of level 2 that distinguishes between the p-states (note that there *is* such a formula in CTL: $\exists \mathsf{X} \forall \mathsf{X} r$). However, in the third time, also the abstract state containing the two p-states is split. A distinguishing formula is $\forall \mathsf{X} \forall \mathsf{X} \forall \mathsf{X} \neg s$ (indicated by a dotted line).

So, we cannot use stability during one step of the while-do loop as a termination criterion. Furthermore, there is no reasonable fixed upper bound on the number of times the abstraction may remain unchanged until it changes again. This bound depends on the number of (reachable) states of the concrete system. Figure 5.7 illustrates a situation where there is "silence" for 4 steps before the states change again. Clearly, this picture can be generalised to any number k of silent steps.

In order to find a solution, we analyse what is going on exactly.

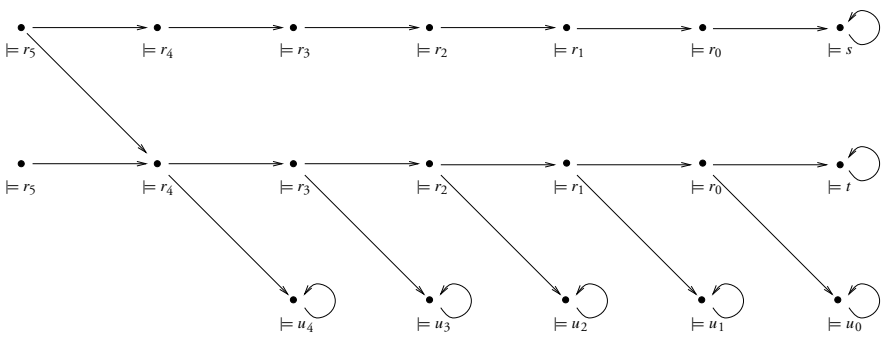


Figure 5.7: k-step-stability $\neq$ stability.

Simulation equivalence

The algorithm for $\forall\mathrm{CTL}^-$ splitting may alternatively be viewed as an algorithm that labels the states of the concrete transition system with $\forall\mathrm{CTL}^-$ formulae that are satisfied in these states. In every next step, formulae of the next level are considered. Concrete states are "$\forall\mathrm{CTL}_i^-$-equivalent" i they have the same labels after step i; the equivalence classes thus created form the abstract states. There is an intimate relation between this $\forall\mathrm{CTL}_i^-$-equivalence and *simulation equivalence* (Definition 2.4.2.6), as explained by Corollary 5.5.1.2 below. For a state s, $\forall\mathrm{CTL}_i^-(s) = \{\varphi \in \forall\mathrm{CTL}_i^- \mid s \models \varphi\}$.

5.5.1.1 L *Let $s, t \in \Sigma$. Then $sim^i(s,t)$ i $\forall\mathrm{CTL}_i^-(s) \supseteq \forall\mathrm{CTL}_i^-(t)$.*

P The $\Rightarrow$ direction is easily proven by induction on the formulae of $\forall\mathrm{CTL}^-$, the $\Leftarrow$ direction by induction on i. □

5.5.1.2 C *Let $s, t \in \Sigma$. $simeq^i(s,t)$ i $\forall\mathrm{CTL}_i^-(s) = \forall\mathrm{CTL}_i^-(t)$.*

Apparently, yet another way to view the algorithm is that it computes $simeq^i$ equivalence classes. From the examples above we know that $simeq^i = simeq^{i+1}$ does not imply $\forall_{j\geq 0}\ simeq^i = simeq^{i+j}$ in general. However, we do know that the underlying simulation relations sim^i (recall that $simeq^i = sim^i \cap (sim^i)^{-1}$ as the domain and range of the simulation equivalence coincide) *do* have that property (see Property 2.4.2.5). Hence, as soon as this simulation relation remains stable for one step, it has stabilised and we can be sure that also $simeq^i$ is stable:

$$sim^{i+1} = sim^i \ \Rightarrow\ \forall_{j\geq 0}\ simeq^{i+j} = simeq^i$$

In order to detect stabilisation of the algorithm, we therefore maintain the simulation relation in a variable *SIM*. We do not have to do this on the level of the concrete states: as the abstract model $\mathcal{A}_i$ that results after splitting for the first i levels, strongly preserves $\forall\mathrm{CTL}_i^-$, we have for all abstract states a and b in $\mathcal{A}_i$ and all concrete states $c \in a$ and $d \in b$: $sim^i(a,b)$ i $sim^i(c,d)$. We can use Lemma 5.5.1.1 to update *SIM*, because after every step of the loop *SIM* should contain those pairs (a,b) of abstract states such that $\forall\mathrm{CTL}_i^-(a) \supseteq \forall\mathrm{CTL}_i^-(b)$. Because also $P = \bigcup_{0\le j\le i} \forall\mathrm{CTL}_j^-$ holds at this point, we may compute *SIM* as $\{(a,b) \in {}_\alpha\Sigma \times {}_\alpha\Sigma \mid P(a) \supseteq P(b)\}$, where for every state s, $P(s) = \{\varphi \in P \mid s \models \varphi\}$. This function $P(\cdot)$ can easily be computed whenever states are being split. Of course, *SIM* may be updated rather than be recomputed, because states that have not been split in two do not have to be considered. We arrive at the algorithm of Figure 5.8. The reader is invited to run this algorithm on the transition system of Figure 5.6. What happens is that although during the second step of the while-loop no splitting occurs, *SIM* *does* change, causing *ready* to remain *false*. Namely, the pair (c_2, c_6), that is still an element of *SIM* after the first step (when splits are e ected for formulae of level 1), disappears from *SIM* in the second step of the loop. The reason is that the level-2 formula $\forall\mathsf{X}\forall\mathsf{X}\neg s$ is an element of $P'(c_6)$ at this point, while it is not in $P'(c_2)$. On the other hand, after the 3rd step, the value of *SIM* does not change any more and hence the algorithm terminates.

The fact that the partition refinement algorithm can be viewed as computing simulation equivalence classes has another implication. It can be shown that simulation equivalence also[3] coincides with the equivalence $\equiv_{\forall\mathrm{CTL}^*}$ induced by $\forall\mathrm{CTL}^*$, as follows. Because, clearly, $\equiv_{\forall\mathrm{CTL}^*} \subseteq \equiv_{\forall\mathrm{CTL}}$ and furthermore $\equiv_{\forall\mathrm{CTL}} \subseteq \mathit{simeq}$ as was shown above, it su ces to show that $\mathit{simeq} \subseteq \equiv_{\forall\mathrm{CTL}^*}$. This is straightforward by induction on the structure of $\forall\mathrm{CTL}^*$ formulae. Several proofs of this sort will be presented in the next chapter. As a consequence, the algorithm presented in this section constructs an abstract model that strongly preserves not only $\forall\mathrm{CTL}$ but $\forall\mathrm{CTL}^*$ as well.

In the beginning of this chapter we explained that partition refinement algorithms would be developed for logical equivalences that do not have corresponding "nice" behavioural definitions. The fragment considered in this section, however, *does* turn out to induce a nice behavioural equivalence. Indeed, algorithms exist that compute the equivalence classes of simulation equivalence — see Section 5.7 on related work. However, the advantage of our approach, being based on logical partition

[3]The fact that $\forall\mathrm{CTL}$ and $\forall\mathrm{CTL}^*$ induce the same equivalence does *not* imply that they have the same expressive power: recall Lemma 2.4.1.4 (page 25) and the remarks below it.

{ initialise: }
${}_\alpha\Sigma := \{\mathit{true}\}$;
${}_\alpha\mathrm{R} := \mathit{rel}({}_\alpha\Sigma)$; ${}_\alpha\mathrm{I} := \mathit{init}({}_\alpha\Sigma)$; ${}_\alpha\|\cdot\|_{\mathrm{Lit}} := \mathit{intp}({}_\alpha\Sigma)$;
ready := *false*;
{ initial split for literals: }
for each $\varphi \in \mathrm{Lit}$ **do** ${}_\alpha\Sigma := \mathit{split}({}_\alpha\Sigma, \varphi)$ **od**;
$P := \mathrm{Lit}$;
${}_\alpha\mathrm{R} := \mathit{rel}({}_\alpha\Sigma)$; ${}_\alpha\mathrm{I} := \mathit{init}({}_\alpha\Sigma)$; ${}_\alpha\|\cdot\|_{\mathrm{Lit}} := \mathit{intp}({}_\alpha\Sigma)$;
$\mathit{SIM} := \{(a,b) \in {}_\alpha\Sigma \times {}_\alpha\Sigma \mid P(a) \supseteq P(b)\}$;
ready := *false*;
$i := 0$;
{ iteratively split for formulae in ith stratum: }
while not *ready* **do**
    { $P = \bigcup_{0 \le j \le i} \forall\mathrm{CTL}^-_j$ }
    $P' := \{\widetilde{\mathit{pre}}_{\mathrm{R}}(\bigvee Q) \mid Q \subseteq P\} \setminus P$;
    { $P' = \forall\mathrm{CTL}^-_{i+1}$ }
    for each $p \in P'$ **do**
        ${}_\alpha\Sigma := \mathit{split}({}_\alpha\Sigma, p)$
    od;
    ${}_\alpha\mathrm{R} := \mathit{rel}({}_\alpha\Sigma)$; ${}_\alpha\mathrm{I} := \mathit{init}({}_\alpha\Sigma)$; ${}_\alpha\|\cdot\|_{\mathrm{Lit}} := \mathit{intp}({}_\alpha\Sigma)$;
    $\mathit{SIM}' := \{(a,b) \in {}_\alpha\Sigma \times {}_\alpha\Sigma \mid P'(a) \supseteq P'(b)\}$;
    ready := $(\mathit{SIM}' = \mathit{SIM})$;
    $P := P \cup P'$;
    $\mathit{SIM} := \mathit{SIM}'$;
    $i := i + 1$
od;

Figure 5.8: Splitting algorithm for ∀CTL with termination.

refinement, is that it is easily adapted to less regular fragments of CTL*. In the next section we consider such a case where there is not a nice correspondence between the property set to be strongly preserved and some behavioural equivalence.

5.6 Splitting for $\varphi \in \forall$CTL

If we know in advance that interest is restricted to a limited set of ∀CTL-properties, then we may consider constructing a strongly preserving model for this subset rather than for full ∀CTL as was done in the previous section. In this section, we focus on the case that $L = \{\varphi\}$, with $\varphi \in \forall$CTL — the resulting approach may be generalised to any finite subset of ∀CTL.

Like in the previous section, we start by identifying the companion of the property set that we are interested in, as this contains the formulae that the abstract states have to be split for. *subform*(φ) denotes the set of all syntactic subformulae of φ. The following lemma may be viewed as an alternative definition of *comp*.

5.6.0.1 L *Let $\varphi \in \forall$CTL. comp(φ) is the union of the following sets:*

1. *subform*(φ) ∩ Lit.
2. *The set of formulae in subform(φ) that are of the form $\forall X\varphi'$.*
3. *If $\forall U(\varphi_1, \varphi_2) \in$ subform(φ), then {false} as well as the set of all formulae $\forall X \forall U_i(\varphi_1, \varphi_2)$, for $i \geq 0$.*
4. *If $\forall V(\varphi_1, \varphi_2) \in$ subform(φ), then {true} as well as the set of all formulae $\forall X \forall V_i(\varphi_1, \varphi_2)$, for $i \geq 0$.*

P Easy, using Definition 5.3.0.4. □

The next step is to stratify the formulae in *comp*(φ) so as to allow for an e - cient computation of their characteristic predicates. We could classify the formulae according to their levels again. However, note that if φ contains nested ∀U/∀V operators, then *comp*(φ) may contain formulae with ∀U/∀V operators (without the i subscripts). Consider Example 5.3.0.5, in particular the last two formulae mentioned there: $\forall X(\forall U(r, s) \vee (q \wedge \forall X\mathit{false}))$ and $\forall X(\forall U(r, s) \vee (q \wedge \forall X(\forall U(r, s) \vee (q \wedge \forall X\mathit{false}))))$. Although both have level ω, it seems natural to compute the characteristic predicate of the second formula after that of the first one, as the first formula is a subformula of the second. Therefore, we choose a more refined stratification in which for each ∀U/∀V subformula of φ, all the approximants are computed in order of acsending levels, before any longer subformulae of φ are considered. This choice

```
_α Σ := {true};
_α R := rel(_α Σ); _α I := init(_α Σ); _α ‖·‖_Lit := intp(_α Σ);
S := {φ' ∈ subform(φ) | φ' : ≠ : φ'_1 ∧ φ'_2, φ' : ≠ : φ'_1 ∨ φ'_2};
L := length(φ);
l := 1;
while l ≤ L do
        Q := {φ ∈ S | length(φ) = l};
        for each φ ∈ Q do
            if φ ∈ Lit →
                _α Σ := split(_α Σ, φ);
            [] φ :=: ∀Xφ' →
                c := pre~(‖φ'‖);
                _α Σ := split(_α Σ, c);
            [] φ :=: ∀U(φ_1, φ_2) →
                i := 0;
                c := false;
                _α Σ := split(_α Σ, c);
                ready := false;
                while not ready do
                    i := i + 1;
                    c' := ‖φ_2‖ ∨ (‖φ_1‖ ∧ pre~(c));
                    _α Σ := split(_α Σ, pre~(c));
                    ... ;
                    ready := (c' = c);
                    c := c';
                od
            [] φ :=: ∀V(φ_1, φ_2) →
                ...
            fi
        od;
        ...
        l := l + 1
od;
```

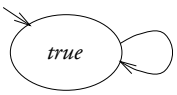
Figure 5.9: Splitting algorithm for $\varphi \in \forall$CTL.

is reflected in the (sketch of an) algorithm in Figure 5.9. The symbol :=: means "is of the form". The *length* of a ∀CTL formula is the heigth of its syntactic parse tree, where the ∀U and ∀V operators are considered single symbols, and the length of a literal is set to 1. Note that termination of this algorithm is guaranteed: execution of the outermost while-loop and for-loop obviously terminates, while the termination of the inner loop follows from Lemma 5.2.0.3.

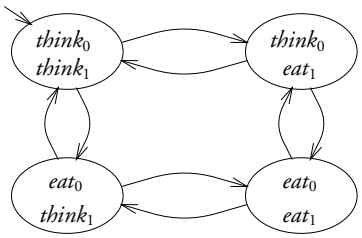
5.6.1 Example

We illustrate the algorithm on the dining mathematicians (Section 4.3.1). Because the concrete transition system of that program is infinite, we make the assumption that the initial value of n is such that during execution it will never take any value greater than some bound N.

Again, we are interested in the mutual-exclusion and non-starvation properties, listed on page 71. We illustrate splitting for formula 4.4. Recall that it is an abbreviation of the ∀CTL formula $\forall\mathsf{V}(\mathit{false}, \neg(\ell_0 = \mathit{eat}) \vee \forall\mathsf{U}(\mathit{true}, \ell_1 = \mathit{eat}))$, whose length L is 4. The initial value of the abstract model $\mathcal{A}$ consists of the abstract initial state *true* with a transition to itself (see Definition 5.4.0.3 of *rel* and *init*):



The subformulae of length 1 are $\ell_0 = \mathit{eat}$ and $\ell_1 = \mathit{eat}$. They are both in Lit. Splitting for them yields a model with 4 states:



In each abstract state, the valuation of the literals has been indicated. $\ell_i = \mathit{think}$ is abbreviated by think_i for $i = 0, 1$; similarly for $\ell_i = \mathit{eat}$. We use the fact that think_i is the negation of eat_i. The abstract transitions can be computed using the function $\mathit{pre}_{\mathrm{R}}$, where R is the concrete transition relation. From Definition 5.4.0.3 of *rel* it follows that there is an abstract transition from a to b i $a \wedge \mathit{pre}(b) \not\equiv \mathit{false}$.

As an example we consider the transition from $think_0 \wedge think_1$ to $eat_0 \wedge think_1$. First note that because we assume an interleaving semantics, the concrete transition relation R is the union of the individual relations R_0 and R_1 corresponding to each mathematician in isolation (as defined by the lines Act1, Act2 and by Act3, Act4 resp. of Figure 4.4 on page 71). In computing pre and $\widetilde{pre}$ we use the following properties, which are easily proven:

$$\begin{aligned} pre_{R_0 \cup R} &\equiv pre_{R_0} \vee pre_{R_1} \\ \widetilde{pre}_{R_0 \cup R_1} &\equiv pre_{R_0} \wedge pre_{R_1} \end{aligned}$$

We compute:

$$\begin{aligned} & think_0 \wedge think_1 \wedge pre_{R_0 \cup R_1}(eat_0 \wedge think_1) \\ \equiv & \\ & think_0 \wedge think_1 \wedge (pre_{R_0}(eat_0 \wedge think_1) \vee pre_{R_1}(eat_0 \wedge think_1)) \\ \equiv & \\ & think_0 \wedge think_1 \wedge ((think_0 \wedge think_1 \wedge odd(n)) \vee (eat_0 \wedge eat_1 \wedge even(n))) \\ \equiv & \\ & think_0 \wedge think_1 \wedge odd(n) \\ \not\equiv & \\ & false \end{aligned}$$

Model checking property 4.4 over the abstract model thus obtained yields "*no*". As the algorithm has not yet terminated, the abstract model is not guaranteed to be strongly preserving (cf. point 2 on page 128). Hence, we cannot know whether the property indeed does not hold in the concrete model, or that the abstraction just does not yet expose su cient detail. The algorithm continues with the subformulae of length 2, being $\forall \mathsf{U}(true, eat_1)$ $(= \forall \mathsf{F} eat_1)$ only. This involves repeated splittings for the approximants. As $\forall \mathsf{U}_0(true, eat_1) \equiv false$ and $\forall \mathsf{U}_1(true, eat_1) \equiv eat_1 \vee (true \wedge \forall \mathsf{X} \forall \mathsf{U}_0(true, eat_1)) \equiv eat_1 \vee (true \wedge false) \equiv eat_1$ (because the concrete transition relation R is assumed to be total (Convention 5.2.0.1), $\forall \mathsf{X} false \equiv false$ over $\mathcal{C}$), splitting for the approximants 0 and 1 has no e ect. As for the next approximant, we compute:

$$\begin{aligned} & \|\forall \mathsf{F}_2 eat_1\| \\ \equiv & \\ & \|eat_1 \vee \forall \mathsf{X} eat_1\| \\ \equiv & \\ & eat_1 \vee \widetilde{pre}_{R_0 \cup R_1}(eat_1) \end{aligned}$$

$$\equiv$$

$$eat_1 \vee (\widetilde{pre}_{R_0}(eat_1) \wedge \widetilde{pre}_{R_1}(eat_1))$$

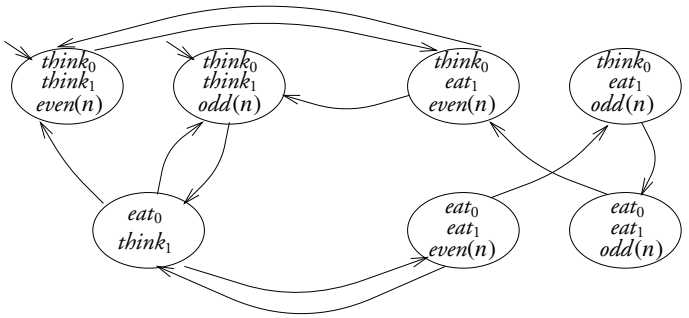
$$\equiv$$

$$eat_1 \vee ((eat_1 \vee (think_0 \wedge even(n))) \wedge ((think_1 \wedge even(n)) \vee (think_1 \wedge odd(n)) \vee (eat_1 \wedge odd(n))))$$

$$\equiv$$

$$(eat_1 \wedge odd(n)) \vee (think_0 \wedge think_1 \wedge even(n))$$

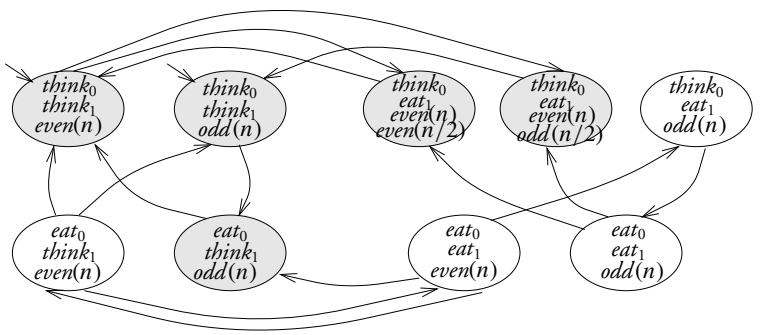
Note that we introduce the "symbolic" representations $even(n)$ and $odd(n)$ during these calculations. How these predicates are represented by the algorithm is of no concern here. By our assumption that the underlying concrete model is finite, effective representations do exist. Splitting states and updating the abstract transitions yields:



Because model checking still yields a negative answer, the model is split for $\forall \mathsf{F}_3 eat_1$. Computing the characteristic predicate of this formula gives:

$$(eat_1 \wedge odd(n)) \vee (think_0 \wedge think_1 \wedge even(n)) \vee (think_0 \wedge eat_1 \wedge even(n/2)) \vee (eat_0 \wedge think_1 \wedge odd(n))$$

and splitting for it yields the following model:



Updating the abstract transition relation and reachability renders only 5 states reachable (the states coloured gray). Model checking now establishes validity of property 4.4. It can even be strengthened to

$$\forall \mathsf{G}((\ell_0 = \mathit{eat} \;\rightarrow\; \forall \mathsf{X}^2 \ell_1 = \mathit{eat}))$$

Furthermore, we can use the same abstract model for verification of formula 4.3. This illustrates an advantage of building an abstract model while computing characteristic predicates of formulae.

For the verification of property 4.5, further splitting is needed (either starting from a single *true* state, or taking the above abstract model as a starting point, which might save some steps). It turns out that splitting for the approximants $\forall \mathsf{F}_i \mathit{eat}_0$ results in a sequence of refinements that distinguish between $\mathit{even}(n/2^i)$ and $\mathit{odd}(n/2^i)$. We have to use the fact that for some k all states where $n/2^i$ is even or odd for $i \geq k$ will be empty, due to our assumption that n will never exceed N. This fact causes stabilisation of the abstract model, allowing the verification of 4.5. Note that the verification of the other two properties was performed independent of N.

5.7 Related Work

This chapter is based on [DGG93a] and partly on [DGD$^+$94]. The notion of companion is similar to that of Fischer-Ladner closure ([FL79, Gol92]) while splitting is reminiscent of *filtration* ([HC84, Gol92]). Our splitting algorithm for ∀CTL can be viewed as a partition refinement algorithm for simulation equivalence. Such an algorithm has been developed before in [CC95]; recently, the topic has enjoyed new interest in the context of hybrid systems — see [HHK95]. The partition refinement algorithm for arbitrary finite subsets of ∀CTL is novel; however, the iterative computation of characteristic predicates is common in symbolic model checking, e.g. [BCM$^+$92]. A paper that is closely related to the idea of "splitting for a single formula" (Section 5.6) is [ASSSV94]. It studies the behavioural equivalence that is induced by a single CTL formula. This equivalence is used to reduce the size of automata in a compositional framework.

Partition refinement algorithms that are based on behavioural definitions of equivalences return in the next chapter — see Section 6.7 for more research related to that topic.

5.8 Concluding Remarks

In this chapter we have investigated conditions for the strong preservation of ∀CTL-properties. The companion *comp*(L) of a set L of properties was introduced and we showed that consistency of the description relation ρ with the formulae in *comp*(L) is a su cient condition for the statewise strong preservation of L. On the other hand, we argued that, in a certain sense, this is also a necessary condition, namely if we require consistency of ρ to hold uniformly in all states of the abstract Kripke structure. This requirement is needed if we seek to establish consistency by a partition refinement algorithm that uniformly splits states. Such partition refinement algorithms were presented for ∀CTL as well as for the case that the property set contains a single ∀CTL-property.

These investigations have exposed a number of interesting facts. First, as stated in Corollary 5.5.1.2, the equivalence induced by ∀CTL (Section 2.4.1) turns out to coincide with simulation equivalence (Definition 2.4.2.6). As a consequence, the splitting algorithm for ∀CTL presented in Section 5.5 can be viewed as a partition refinement algorithm for simulation equivalence. Partition refinement algorithms that divide out the equivalence classes of some behavioural equivalence by successive refinement of the blocks of a partition have been presented for bisimulation (e.g. in [PT87, BFH$^+$92]) and stuttering (or branching) equivalence (e.g. in [BCG88, GV90]). These equivalences both correspond to logics that are closed under negation, namely CTL* and CTL*(U) (also see the next chapter). For simulation equivalence, the only previous decision algorithm that we know of is [CC95], which does not fit the partition refinement scheme. More recently, [HHK95] presents an algorithm that indeed uses ideas from [PT87].

In general, the behavioural equivalence corresponding to a property set that is an arbitrary subset of a temporal logic will not have a nice regular form like bisimulation, simulation equivalence or stuttering equivalence. When reduction or decision algorithms for such equivalences are to be developed, it may be worth considering a type of partition refinement algorithm that is directly based on the (companion of) the property set, as suggested by the results in this chapter. In contrast to the partition refinement algorithms that are based on a behavioural equivalence like bisimulation, and will henceforth be called *behavioural partition refinement algorithms*, we denote the type of algorithm presented in this chapter as *logical partition refinement*.

Another interesting observation is the fact that the "one-step-stabilisation" property that holds for the approximants of e.g. bisimulation, does not hold for simulation equivalence.

A few limitations, assumed in this chapter for simplicity, may easily be lifted. As already observed in Section 5.5, the splitting algorithm for ∀CTL may be used to split

for ∀CTL* too. A natural question is whether also the splitting algorithm for a single $\varphi \in \forall$CTL may be generalised for $\varphi \in \forall$CTL*. The results in [Dam94] imply that every formula in ∀CTL* can be expressed in terms of the modality $\forall\mathsf{X}$ and a fixpoint construct (indeed, expressed in the μ-calculus). Based on this observation, we expect that a similar splitting algorithm for $\varphi \in \forall$CTL* exists. However, the translation from ∀CTL* to the μ-calculus may blow up the size of the formula considerably. The development of splitting algorithms for the existential fragments ∃CTL and ∃CTL* is similar — and also the combination to obtain algorithms that split for fragments of the full logics CTL and CTL* should be straightforward.

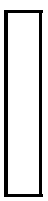
Another limitation is the assumption, in Convention 5.2.0.1, that the underlying concrete model $\mathcal{C}$ is finite. We conjecture that when this limitation is weakened to the requirement that $\mathcal{C}$ is finitely branching, most of the results will remain unchanged[4] — in particular, the same algorithms can still be used, with the di erence that termination is no longer guaranteed. However, *if* they terminate, then the constructed abstract models will be strongly preserving.

Although these results may please the theoretician, it remains to be seen what the implications are for the practice of model checking. The computation of characteristic predicates, that is necessary to perform the splitting, is also performed in (symbolic) model checking algorithms. The di erence is that our algorithms construct an abstract model at the same time. On the one hand, this means that even more operations have to be performed (the splitting); on the other hand the availability of a weakly preserving abstraction, at any stage of the algorithm, may have advantages. For example, in Section 5.6.1, the abstraction that was constructed by splitting for one property could be used to establish the truth of another property as well. Also, it may be that the satisfaction of the formula is often established at an earlier stage of the algorithm, before stability has been reached. Only practical experiments can tell.

[4] More precisely, point 1 of Lemma 5.2.0.3 will become false, but the proofs of points 2 and 3 can be adapted. Also the proof of Lemma 5.5.0.2 can be adapted.

Chapter 6

Logics, Equivalences and Behavioural Partition Refinement



In this chapter we take behavioural equivalences as a starting point to develop partition refinement algorithms. A number of fragments of CTL and the induced equivalences are studied. In addition, the correspondence between these induced behavioural equivalences and partition refinement algorithms is investigated, leading to a generic scheme.*

6.1 Introduction

The approach to achieve strong preservation that was introduced in the previous chapter is partition refinement of the abstract model. The splitting of abstract states can be viewed as the construction of a *quotient system* where each abstract state represents a class of concrete states that all agree on certain properties, as is captured by the notion of consistency. In order to construct a model that strongly preserves some set Φ of formulae, abstract states have to be split up such that each of the resulting parts is consistent with all formulae in Φ's companion. Taking Φ equal to all of $\forall$CTL turned out to lead to the construction of a quotient that coincides with the quotient under simulation equivalence. Indeed, there are many correspondences between the equivalence induced (Definition 2.4.1) by a logic on the one hand and some behavioural equivalence defined directly in terms of states and transitions of a system on the other. Furthermore, partition refinement algorithms have been developed for a number of such behavioural equivalences. For example, the partition refinement algorithm developed in [BFH$^+$92] computes the equivalence classes under bisimulation equivalence. Another example is the partition refinement algorithm for stuttering equivalence (branching bisimulation) of [GV90]. In [BCG88], stuttering equivalence has been shown to coincide with the equivalences induced by CTL(U) and CTL*(U). In those cases, these algorithms may be used to construct abstract models that strongly preserve the corresponding logic.

In this chapter, we investigate such partition refinement algorithms that are based on behavioural definitions of equivalences induced by temporal logics. Such algorithms are called *behavioural partition refinement algorithms*, abbreviated as BPRA. To find the equivalence classes of the equivalence $\equiv_{\mathrm{CTL}^*}$ induced by CTL* (two states are equivalent i they make the same CTL* formulae true), for example, we can use the fact that this equivalence coincides with *bisimulation*, and use a BPRA that has been developed for this behavioural equivalence. Thus, we are interested in two kinds of correspondences:

- One between a temporal logic and a behavioural equivalence.
- One between a behavioural equivalence and a BPRA.

Each of these will be further clarified in the following subsections.

6.1.0.1 C *Throughout this chapter, we assume a Kripke structure* $\mathcal{T} = (\Sigma, \rightarrow, \pounds)$ *whose transition relation* $\rightarrow$ *will be written in infix notation. Like in the previous chapter, we assume that* $\rightarrow$ *is* total, *i.e. for every* $s \in \Sigma$ *there exists* $s' \in \Sigma$ *such that* $s \rightarrow s'$. $\pounds$ *is the state-labelling function; the initial states remain anonymous as they are irrelevant. Variables* s *and* t *range over* Σ, *unless stated*

otherwise. We recall that paths are by definition infinite *sequences of* $\rightarrow$*-related states; finite such sequences are called* prefixes.

6.1.1 Logics and equivalences

In order to formalise the "compatibility relations" between temporal logics and (behavioural) equivalences, we give the following definitions[1].

6.1.1.1 D *Let* $\equiv \subseteq \Sigma \times \Sigma$ *be an equivalence relation and* L *a logic interpreted over* Σ.

- $\equiv$ *is* fine for L *i* $\equiv \subseteq \equiv_{\mathrm{L}}$.
- $\equiv$ *is* abstract for L *i* $\equiv \supseteq \equiv_{\mathrm{L}}$.
- $\equiv$ *is* adequate for L *i* *it is both fine and abstract for* $\equiv_{\mathrm{L}}$[2].

We will consider some cases where L is a logic like CTL* or one of its fragments. In each of these cases, we seek to identify a behavioural equivalence $\equiv$ that is fine for L and that can be used as the basis of a partition refinement algorithm (see Section 6.1.2 below) that constructs the quotient $\mathcal{T}/\equiv$. The states of such a quotient system are identified with the equivalence classes of $\equiv$. The transition relation of the quotient is[3] $\rightarrow^{\exists\exists}$. For every logic L that we consider and every behavioural equivalence $\equiv$ that is fine for it, the quotient system strongly preserves L. This can be proven by showing that there exists a behavioural equivalence, of the same type (e.g. bisimulation, stuttering) as $\equiv$, which relates every state a of $\mathcal{T}/\equiv$ to each of the concrete states that a contains. If in addition $\equiv$ is abstract, the quotient is also the smallest: every abstract Kripke structure that strongly preserves L (statewise) is at least as large.

Adequacy has been the subject of many studies not only in computer science but also in logic. See the section on related work towards the end of this chapter for an

[1] Indeed, this is a refinement of Definition 3.2.2.1, specialised for the case that ρ is an equivalence relation.

[2] In [Pnu86], this is (conversely) called adequacy of the logic with respect to the equivalence — we also sometimes do this. Also, in some places (e.g. [ACH94]) adequacy is defined/used as what we call fineness.

[3] These quotient systems may indeed be viewed as Abstract Kripke structures, introduced in Chapter 4 (Definition 4.2.3.7). There is an abstract state a for each equivalence class C of $\Sigma/\equiv$, and each such a is mapped by the concretisation function γ to C. The transition relation of the quotient then is the free transition relation ${}_{\alpha}\rightarrow^{F}$ (Definition 4.2.3.3), defined with respect to the concrete relation $\rightarrow$. Because the associated abstraction function α is such that for each concrete state c, $\alpha(\{c\})$ is the $\equiv$-equivalence class of c, we have $a\ {}_{\alpha}\rightarrow^{F}\ b$ i $\exists_{c\in\gamma(a)}\ \exists_{d\in\gamma(b)}\ c \rightarrow d$. In this chapter, however, we consider strong preservation and therefore we interpret both the universal and existential path quantifiers of CTL* over the same transition relation. Hence, we prefer not to present it as the free transition relation.

(incomplete) overview. In this chapter we recall some well-known results and present several new ones. An interesting trade-o in this area between logics and behavioural equivalences is that between the expressive power and the distinguishing power of a logic (Section 2.4.1). On the one hand, we would like our logic to be expressive so that we can describe our properties in a su ciently detailed way. On the other hand, it should induce an equivalence that is as coarse as possible, because this allows for a good reduction of Kripke structures. Interestingly, the distinguishing power of a logic does not necessarily strictly increase when its expressive power does, as is witnessed by the fact that various logics of di erent expressive power, among which CTL* and CTL, induce the same equivalence (bisimulation). Of course, the distinguishing power cannot decrease with increasing expressive power (see Lemma 2.4.1.4). But it may pay o to look for a maximally expressive logic that still induces a certain equivalence[4].

6.1.2 Equivalences and partition refinement algorithms

The relation between equivalences and partition refinement algorithms has been studied less extensively — and certainly in a less structured manner — than that between logics and equivalences. As already indicated by their name, such algorithms compute equivalence classes by iterative refinement of a partition. It seems that there is a close relation between the format of the definition of the equivalence on the one hand, and the way in which the blocks of a partition are being refined during such an algorithm on the other.

In this chapter, we instigate a more systematic investigation of this field.

6.1.3 Overview of the chapter

As an introduction, we refer to, and at some points generalise, some correspondence results that have been studied rather extensively. In Section 6.2 we look at CTL* and bisimulation, while Section 6.3 deals with CTL*(U), obtained from CTL* by dropping the Next operator, and stuttering equivalence. Section 6.4 then shows how partition refinement algorithms for these cases are indeed induced by the definition of the behavioural equivalences. By varying the notion of *splitter*, which is central to these algorithms, we obtain both algorithms as instances of a generic scheme. Section 6.5 analyses how we may manipulate the splitter to obtain better reductions of the transition system, leading to a new behavioural equivalence. The search for the

[4]However, one should also keep in mind that di erent expressive power may come at the cost of a more expensive model checking algorithm. For example, although CTL* induces the same equivalence as CTL (see Section 6.2), it is known (see e.g. [Eme90], pp. 1044–1045) to have a higher model-checking complexity.

corresponding adequate fragment of CTL* brings along some surprises. Section 6.6 presents the appropriate instance of the partition refinement scheme. Section 6.7 contains an overview of some related work, and Section 6.8 concludes.

6.2 CTL*, CTL, and Bisimulation

In this section we recall a well-known result, namely that bisimulation (Definition 2.4.2.7) over finitely-branching Kripke structures is adequate for both CTL* and CTL. This case has been studied extensively, not only in the context of computer science, but also in the field of modal logic. The references given in Section 6.7 on related work should help the interested reader in tracing the origins of the reported results. None of these results is new — proofs are only given for completeness. The purpose of this section is to serve as a primer, illustrating the general form of the definitions of behavioural equivalences that we will encounter, the kind of lemmata that will play a role, and the form of their proofs.

We repeat Definition 2.4.2.7 in a slightly di erent form.

6.2.0.1 D *Let $\equiv \subseteq \Sigma \times \Sigma$ be a symmetric relation such that for every $s, t \in \Sigma$, $s \equiv t$ implies:*

1. $\pounds(s) = \pounds(t)$.
2. *For*[5] *every $s \to s'$, there exists $t \to t'$ such that $s' \equiv t'$.*

Then $\equiv$ is called a bisimulation. The largest bisimulation[6] *is denoted $\equiv_{\text{bis}}$.*

One direction of the adequacy result, namely that $\equiv_{\text{bis}}$ is fine for CTL* (and hence for CTL), holds for arbitrary Kripke structures.

6.2.0.2 L *If $s \equiv_{\text{bis}} t$, then $\forall_{\varphi \in \text{CTL}^*}\ s \models \varphi \ \Leftrightarrow\ t \models \varphi$.*

The proof of this lemma is based on an inductive argument on the structure of the formulae. Because the inductive definition of these formulae (Definition 2.3.0.1, page 19) involves path formulae, bisimulation equivalence is extended to paths so that the induction hypothesis can be strengthened with a part stating that any two bisimilar paths satisfy the same CTL* path formulae.

6.2.0.3 D *Let $\bar{s}$ and $\bar{t}$ be paths in $\mathcal{T}$. $\bar{s} \equiv_{\text{bis}} \bar{t}$ i $\forall_{i \geq 0}\ \bar{s}(i) \equiv_{\text{bis}} \bar{t}(i)$.*

[5] More precisely, this condition is supposed to mean "for every s' such that $s \to s'$, there exists t' such that $t \to t'$ and $s' \equiv t'$". In the remainder, similar conditions will be abbreviated likewise.

[6] The largest bisimulation exists because the empty relation is a bisimulation and bisimulations are closed under union.

6.2.0.4 L *If $s \equiv_{\text{bis}} t$, then for every $\bar{s} \in \mathit{paths}(s)$ there exists $\bar{t} \in \mathit{paths}(t)$ such that $\bar{s} \equiv_{\text{bis}} \bar{t}$.*

This lemma is easily proven by inductively constructing the path $\bar{t}$. In the sequel, we encounter more of such "state-path lemmata", which are not always proven as easily.

The other half of the adequacy result, abstractness of $\equiv_{\text{bis}}$ for CTL* and CTL, follows directly from the following lemma.

6.2.0.5 L *Assume that $\mathcal{T}$ is finitely branching. If $\forall_{\varphi \in \text{CTL}}\ s \models \varphi \Leftrightarrow t \models \varphi$, then $s \equiv_{\text{bis}} t$.*

P . Assume that $\forall_{\varphi \in \text{CTL}}\ s \models \varphi \Leftrightarrow t \models \varphi$. We have to show that $s \equiv_{\text{bis}} t$. Because $\equiv_{\text{bis}}$ is the largest bisimulation, we have to show that the pair (s, t) is an element of some bisimulation $\equiv \subseteq \Sigma \times \Sigma$. We define this relation as follows: $u \equiv v$ i $\forall_{\varphi \in \text{CTL}}\ u \models \varphi \Leftrightarrow v \models \varphi$. Clearly $s \equiv t$. We show that $\equiv$ is a bisimulation.

1. $\pounds(s) = \pounds(t)$ as CTL includes all literals.
2. Suppose that $s \to s'$. We have to show that there exists t' such that $t \to t'$ and $s' \equiv t'$. (*) Suppose that this is *not* the case. Consider the set T of all successors of t. Because $\to$ is total (recall Convention 6.1.0.1), T is nonempty. Because $\mathcal{T}$ is finitely branching, T is finite, say $T = \{t'_1, \ldots, t'_n\}$, with $n \geq 1$. By our assumption (*), there exist formulae $\varphi_1, \ldots, \varphi_n \in \text{CTL}$ such that for every $1 \leq i \leq n$, $s' \models \varphi_i$ while $t'_i \not\models \varphi_i$. But then $s \models \exists\mathsf{X}(\varphi_1 \wedge \cdots \wedge \varphi_n)$ while $t \not\models \exists\mathsf{X}(\varphi_1 \wedge \cdots \wedge \varphi_n)$, implying that $s \not\equiv t$. Contradiction. □

That the condition of image-finiteness is essential, is shown by Figure 6.1. The states s and t are not bisimilar. An illuminating way to see this is by considering the definition of bisimulation in terms of a game ([Ehr61, Fra54]). Such a game is played by two players: Attacker, whose intention it is to show that two states are not bisimilar, and Defender, who wants to prove the opposite. They take turns in making moves, where Attacker always takes the initiative and Defender responds by a "countermove". The definition of what are allowed moves and countermoves depends on the equivalence being defined. In the case of bisimulation, we may view the transition system as the games' board. There are two pebbles, a black and a white, which are placed on states of the transition system and may be moved around by the players. Initially, the pebbles are placed (in any order) on the states s and t that have to be shown to be (not) bisimilar. A move, of Attacker, consists in choosing one of the pebbles (a new choice may be made in every move) and moving it to a successor of the state that it is currently on. In a countermove, Defender must take the other pebble and move it to a successor state. After every round (i.e. pair of moves), as well as initially, the states s' and t' that the pebbles are placed on should

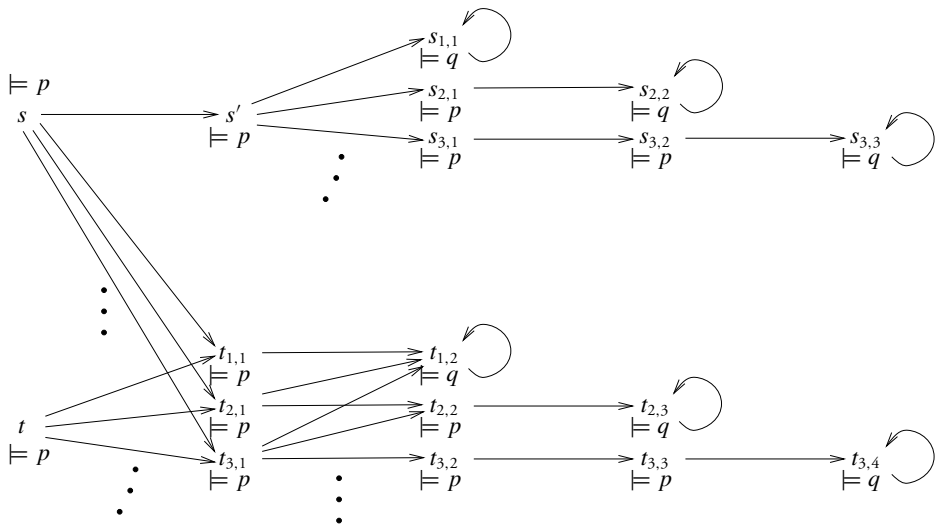


Figure 6.1: Non-image-finite structures that are CTL*-equivalent but not bisimilar.

satisfy $\pounds(s') = \pounds(t')$. Now, s and t are bisimilar i Defender has a winning strategy, i.e. if she can always choose a countermove such that the resulting configuration is permitted again — regardless of how Attacker chooses its moves.

When this game is played on Figure 6.1, Attacker may choose the pebble that is initially on state s, say it is white, and move it to s'. Defender then has to move the black pebble from t to one of its infinitely many successors. Suppose she chooses $t_{i,1}$. Attacker then may choose the white pebble again and move it to $s_{i+1,1}$. Clearly, after $i - 1$ rounds, Defender will not be able to perform a permitted countermove any more. Thus, we see that there is no winning strategy for Defender, and hence $s \not\equiv_{\text{bis}} t$.

On the other hand, it can be proven ([Hol96]) that there is no CTL* formula that distinguishes between s and t. Intuitively, the essential point is that it is not possible in CTL* to form infinite conjunctions — and an infinite conjunction would be the only way to express the di erence between s and t: namely that s has a successor, s', such that for *every* positive number n there is a path starting from s' that reaches a q-state in precisely n steps, whereas t does not have such a successor.

We summarise:

6.2.0.6 C *Assume that $\mathcal{T}$ is image-finite (recall that $\rightarrow$ is total by Convention 6.1.0.1). Then $\equiv_{\text{bis}}$ is adequate for both* CTL* *and* CTL.

P From Lemmata 6.2.0.2 and 6.2.0.5. □

The fact that in order for two paths to satisfy the same CTL* formulae, the states on them must be pairwise bisimilar can be intuitively explained by observing that the Next operator X of CTL* allows to di er between paths of di erent lengths. The expressive power caused by the presence of the Next operator has been questioned as will be discussed in Section 6.3.1. In Section 6.3.2, we consider logics that are obtained by dropping the Next operator from CTL* and CTL.

6.3 Nextless CTL* and CTL

6.3.1 The Next operator

The notion of a *step* made by a system and the closely related notion of a *next state* take an important place in the modelling of behaviour. We discuss some considerations that influence the definition of these notions.

What is a step?

From the point of view of the system itself, we may informally define a step to be determined by the next moment at which something in the system's state changes. We refer to such a step as a *system step*. For real-life systems, such a next moment will often be non-existent because there are parts of the state that change continuously, e.g. the positions of electrons in a wire of some electrical device. By (mathematically) modelling a real-life system we can abstract from state changes that need not be observable for the purposes we intend to use the model for. However, it is often not straightforward to remove all irrelevant aspects in the modelling process. For example, when modelling processes by labelled transition systems (where transitions carry symbols from an alphabet of *actions*) or Kripke structures, it is often possible to further abstract such models by reducing them with respect to some behavioural equivalence like bisimulation. The reason is that bisimilar states cannot be distinguished by the specification formalism one is using in any case, be it a temporal logic like CTL* or a process specification language like μCRL ([GP95]).

So, from the point of view of an observer of the system, what a step is depends very much on the observer's observational power. We will consider here the case that this power consists in the ability to evaluate the truth value of formulae from some temporal logic L — we might say that the observer *is* L. The aspects of a system that are observable by L are certain local properties, as determined by the propositional literals and propositional formulae of L, and furthermore dynamical

properties determined by the temporal operators and path quantifiers. Such temporal operators can look forward (or backward) along a computation path and specify that certain properties (as expressed by the operands) hold at some point(s) along it. At which points exactly, depends on the particular form of the operator. An Eventual[7] operator (cf. the F operator) says that something should hold at some state along a path, an Invariant (cf. G) operator that something should hold at all states along a path, and an Until operator is a combination of these two.

As opposed to a system step, we assume a *logical step* to be determined by the next moment (or: some moment, in the case that nondeterminism is present) that some L-property changes. Whereas the notion of a system step can be defined locally (namely in terms of the variables in the state space), a logical step is a global notion in the sense that a certain change in some variable may well be a logical step in some cases but not in others.

Many temporal logics include a Next operator, which specifies that something holds in the next state along a path, where the next state is the state that is reached by taking one *system* step. When considered in the light of the distinction between the "real" system and the way it occurs to the observer L, this Next operator may be troublesome because it refers to the steps of the "real" system that are caused by changes in the values of state variables that are not visible through L otherwise. In other words, it does not make sense to abstract from certain variables while still making their changes observable. States of the system that cannot be distinguished by the logic are not of interest and therefore it does not make sense to have a notion of step that is "finer" than the observed notion of step, i.e. the notion of step induced by the logic.

A number of solutions have been proposed. In [Lam83], Lamport simply drops the Next operator from the logic. This approach, as well as a closely related development in the field of process equivalences, is discussed in the following section. Another solution consists in parametrising the Next operator with the state variable whose alteration it refers to, allowing to express for example that "at the next moment that p changes, φ will hold". The operator $\bigcirc_\sigma$ of [ACH94] takes such an approach. In that framework, which considers actions, it is the occurrence of the next input symbol (σ) that determines the notion of step. A variation on this idea is found in the definition of the timed μ-calculus T_μ [HNSY94]. When the interpretation of the μ-calculus is extended to systems with continuous time, the Next operator of the untimed μ-calculus becomes meaningless because the notion of next moment is not well-defined any more. In the timed version it is replaced by the binary operator $\triangleright$ called "single-step Until". The formula $\varphi \triangleright \psi$ intuitively has the same meaning as $\mathsf{U}(\varphi, \psi)$ with the di erence that fulfilment of the eventuality ψ has to take place

[7] To avoid confusion we capitalise these adjectives.

within one "$\triangleright^{\delta}$-step" consisting of a time delay $\delta \geq 0$ followed by the instantaneous change of zero or more state variables.

Dropping the Next

One of the early pleas against the Next operator is found in [Lam83]. The objection there is that inclusion of the Next operator renders a temporal logic *too* expressive for the goal it has been conceived for, to wit, the specification of (concurrent) computer programs at any level between an abstract specification and a concrete implementation. It allows one to write formulae that specify irrelevant details, like a specification of a queue that includes the requirement that "putting an element in the queue should take exactly 17 steps". Not only is the number of steps that an operation takes not a meaningful concept when one gives an abstract, high level specification of a queue, but it also is unnecessarily restrictive in a refinement setting, where a "step" may at some later point be implemented by a number of steps at a lower lever.

A related development is the research on comparative concurrency semantics [vG90b] in the context of process algebras with *silent moves* ([vG93a]). In comparing models for concurrent systems without silent moves, bisimulation equivalence is commonly considered to be the finest equivalence (i.e. it makes most distinctions) that one may need, or equivalently, the coarsest equivalence that respects the branching structure of a process without silent moves. If one is considering silent (τ-) moves, bisimilarity could be adapted by treating τ-moves as if they were observable, but this would yield too fine an equivalence. For example, it would distinguish between the processes a and $\tau.a$ and this does not correspond with the intended meaning of τ. Several *observational* or *weak* equivalences have been proposed to more faithfully capture the notion of equivalence in the presence of silent moves: *observational equivalence* in the sense of [HM85], *observation equivalence* (called *τ-bisimulation equivalence* in [BK85]) of [Mil80], η-bisimulation of [BG87], delay bisimulation of [Mil83], and *branching bisimulation* of [GW89]. In [vG93b], van Glabbeek convincingly argues that, in a sense, branching bisimulation equivalence (branching equivalence for short) is the coarsest equivalence that respects the branching structure of a process with silent moves.

An essential feature of (semantic models based on) branching equivalence is that a silent move is only observable if it results in a change of "potential" in the system. Intuitively, this means that the value of some proposition changes, or that the possibility to follow some execution path has disappeared because another direction was chosen. It turns out that a silent move is observable i it coincides with a "CTL*(U) step" (Section 2.3.1) in the sense that we discussed above. This follows from the

results of [BCG88] and [DNV90b], which imply that, roughly[8], branching equivalence is adequate for CTL*(U). Adding back the Next operator to CTL*(U) results in a logic whose induced equivalence is finer than branching equivalence and thus too fine according to [vG93b].

So, the development from (strong) bisimulation to branching bisimulation as the proper notion of equivalence in the world of comparative concurrency semantics, is the parallel of the shift of attention from CTL* to CTL*(U) in the world of specification logics. As such, it provides another piece of evidence for the objections against the Next operator.

6.3.2 CTL*(U), CTL(U) and stuttering equivalence

We introduce the following notation to represent transitions that "stutter" under some notion of equivalence.

6.3.2.1 D *If $\equiv$ is an equivalence relation on Σ, then the transition relation $-\equiv\rightarrow \subseteq \Sigma \times \Sigma$ is defined by $s -\equiv\rightarrow t$ i $s \rightarrow t \wedge s \equiv t$.*

6.3.2.2 D *For a path $\bar{s}$ and an equivalence relation $\equiv$ over Σ, $partit_{\equiv}(\bar{s})$ is the partitioning of $\bar{s}$ into maximal parts (i.e. "subpaths"; see page 21) such that within each part, all states are $\equiv$-equivalent.*

By dropping the Next operator from CTL$^\circ$ (see Notation 2.3.1.2 on page 21), we loose the possibility to distinguish between paths of di erent length. It is therefore not surprising that the behavioural equivalence that is adequate for CTL$^\circ$(U) does not contain a clause of the form "if s can make a transition to some s', then t can make a transition to some t' such that s' and t' are equivalent again", but rather something of the form "if s can make a number of transitions to some s', then t can make some (possibly di erent) number of transitions to some t' such that s' and t' are equivalent again". It turns out that in addition to the recursive requirement that s' and t' be equivalent, also all the states that lie on the prefix from s to s' (including s but excluding s') have to be equivalent to the states on the prefix from t to t' (including t but excluding t'). Intuitively, this can be understood by considering the expressive power of the Until operator (which is the only temporal operator in the logic): if the formula $\mathsf{U}(\varphi, \psi)$ is to hold in both s and t, then it must not only be that there are states s' and t', reachable from s and t respectively, which both satisfy ψ — which can be any formula again so that, recursively, s' and t' have to be equivalent

[8] There are some technical points relating to the di erence between LTSs and Kripke structures and to the divergence blindness of branching equivalence.

—, but also all states on the prefixes between s and s' and between t and t' have to satisfy φ — so that all these states have to be equivalent as well.

Before presenting *stuttering equivalence*, which is the behavioural equivalence adequate for CTL$^\circ$(U), we give the definition of *divergence blind stuttering equivalence* (*dbs-equivalence*) [DNV90b, GV90], both on states and on paths.

6.3.2.3 D *Let $\equiv \subseteq \Sigma \times \Sigma$ be a symmetric relation such that for every $s, t \in \Sigma$, $s \equiv t$ implies:*

1. *$\pounds(s) = \pounds(t)$.*

2. *For every $s \xrightarrow{-} s_1 \xrightarrow{-} \ldots \xrightarrow{-} s_{k-1} \to s_k$ such that $k \geq 0$, there exists $t \xrightarrow{-} t_1 \xrightarrow{-} \ldots \xrightarrow{-} t_{l-1} \to t_l$ such that $l \geq 0$ and $s_k \equiv t_l$.*

Then $\equiv$ is called a divergence blind stuttering equivalence (dbs-equivalence). The largest divergence blind stuttering equivalence is denoted $\equiv_{\text{dbs}}$.

$\equiv_{\text{dbs}}$ is extended to paths by defining $\bar{s} \equiv \bar{t}$ i , letting $partit_{\equiv_{\text{dbs}}}(\bar{s}) = \bar{s}_{I_1}, \bar{s}_{I_2}, \ldots$ and $partit_{\equiv_{\text{dbs}}}(\bar{t}) = \bar{t}_{J_1}, \bar{t}_{J_2}, \ldots$, for every $i \geq 0$, every $s' \in \bar{s}_{I_i}$ and $t' \in \bar{t}_{J_i}$, $s' \equiv_{\text{dbs}} t'$.

Just like in the case of bisimulation, it can be checked easily that the largest divergence blind stuttering equivalence exists. This remark applies to all other similar definitions in this chapter; we will not repeat it.

dbs-Equivalence is "almost" adequate for CTL$^\circ$(U). There is a subtle point that causes a di erence. In proving that two dbs-equivalent states satisfy the same CTL*(U) formulae, one has to prove, just like in the case of bisimulation and CTL* (cf. Lemma 6.2.0.4), that if $s \equiv_{\text{dbs}} t$, then for every s-path $\bar{s}$ there exists a t-path $\bar{t}$ that is dbs-equivalent to $\bar{s}$. This is not possible because the definition of dbs-equivalence allows us to match a positive number of transitions, taken from s, with zero transitions taken from t (i.e. "staying in t"; this corresponds to taking $l = 0$ in point 2 of Definition 6.3.2.3 above and should not be confused with taking a single transition $t \to t$ leading back to t). So, although for every s-path there exists a dbs-equivalent *t-prefix*, this is not necessarily a *t-path*. It is this point that makes the equivalence blind to divergence. For example, the states s and t in Figure 6.2 cannot be distinguished. In [DNV90b] it is proven that if the interpretation of CTL$^\circ$(U) path formulae is altered in such a way that $\forall$ and $\exists$ quantify over all *prefixes*, then the equivalence induced by CTL$^\circ$(U) coincides with dbs-equivalence. Here, we adapt the behavioural equivalence instead of the logic.

The incompatibility is repaired in the following definition of *(divergence sensitive) stuttering equivalence* (*s-equivalence*).

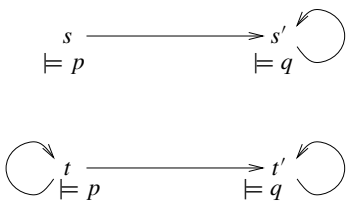


Figure 6.2: Example of dbs-equivalent structures.

6.3.2.4 D *Let $\equiv$ be an equivalence relation on Σ. We say that s* has infinite $\equiv$-stuttering, *denoted $infstut_{\equiv}(s)$, i there exists an s-path $\bar{s}$ such that for all states s' on $\bar{s}$, $s' \equiv s$.*

6.3.2.5 D *Let $\equiv \subseteq \Sigma \times \Sigma$ be a symmetric relation such that for every $s, t \in \Sigma$, $s \equiv t$ implies:*

1. *$\pounds(s) = \pounds(t)$.*
2. *$infstut_{\equiv}(s)$ i $infstut_{\equiv}(t)$.*
3. *For every $s \longrightarrow_{\equiv} s_1 \longrightarrow_{\equiv} \ldots \longrightarrow_{\equiv} s_{k-1} \rightarrow s_k$ such that $k \geq 0$, there exists $t \longrightarrow_{\equiv} t_1 \longrightarrow_{\equiv} \ldots \longrightarrow_{\equiv} t_{l-1} \rightarrow t_l$ such that $l \geq 0$ and $s_k \equiv t_l$.*

Then $\equiv$ is called a (divergence sensitive) stuttering equivalence (dss-equivalence). The largest stuttering equivalence is denoted $\equiv_{\text{stut}}$.

$\equiv_{\text{stut}}$ is extended to paths by defining $\bar{s} \equiv_{\text{stut}} \bar{t}$ i , letting $partit_{\equiv_{\text{stut}}}(\bar{s}) = \bar{s}_{I_1}, \bar{s}_{I_2}, \ldots$ and $partit_{\equiv_{\text{stut}}}(\bar{t}) = \bar{t}_{J_1}, \bar{t}_{J_2}, \ldots$, for every $i \geq 0$, every $s' \in \bar{s}_{I_i}$ and $t' \in \bar{t}_{J_i}$, $s' \equiv_{\text{stut}} t'$.

This adapted definition enables us to prove the following state-path lemma.

6.3.2.6 L *If $s \equiv_{\text{stut}} t$, then for every $\bar{s} \in paths(s)$ there exists $\bar{t} \in paths(t)$ such that $\bar{s} \equiv_{\text{stut}} \bar{t}$.*

P . Let $\bar{s}$ be an s-path. We distinguish the following cases.

1. $partit_{\equiv_{\text{stut}}}(\bar{s})$ is an infinite sequence $B_0, B_1 \ldots$ of blocks. Then each block is finite. For every $i \geq 0$, let b_i be the first state of B_i (these b_i exist because, by definition of partitioning, each B_i is non-empty). Let $c_0 = t$. By Definition 6.3.2.5, there exists a state c_1 such that $c_0 \longrightarrow_{\equiv_{\text{stut}}} c_0^1 \longrightarrow_{\equiv_{\text{stut}}} \ldots \longrightarrow_{\equiv_{\text{stut}}} c_0^j \rightarrow c_1$ and $b_1 \equiv_{\text{stut}} c_1$. Let C_0 be the block $c_0, c_0^1, \ldots, c_0^j$. This way, we can inductively define states c_i and blocks C_i for all $i \geq 0$. It is now easily seen that for the path $\bar{t}$ formed by $C_0, C_1, \ldots$, we have $\bar{s} \equiv_{\text{stut}} \bar{t}$.

2. $\mathit{partit}_{\equiv_{\mathrm{stut}}}(\bar{s})$ is a finite sequence $B_0, B_1, \ldots, B_k$ of blocks. Then B_k is infinite. For every $0 \leq i \leq k$, let b_i, c_i and C_i be as in the previous case, with the exception of C_k. Because $\mathit{infstut}_{\equiv_{\mathrm{stut}}}(b_k)$ holds (as is easily seen) and $b_k \equiv_{\mathrm{stut}} c_k$, by Definition 6.3.2.5 also $\mathit{infstut}_{\equiv_{\mathrm{stut}}}(c_k)$ holds. Hence, there is an infinite c_k-path of $\equiv_{\mathrm{stut}}$-equivalent states; let this be C_k. It is now easily seen that for the path $\bar{t}$ formed by $C_0, C_1, \ldots, C_k$, we have $\bar{s} \equiv_{\mathrm{stut}} \bar{t}$. □

The above definition of stuttering equivalence is di erent from that given in [DNV90b]. In that paper, the transition relation of a Kripke structure does not have to be total. s-equivalence of states is defined as dbs-equivalence in the so-called *live-lock extension*, which is obtained by the applying the following transformation to the transition system. For each state that has no outgoing transitions or occurs on a cycle of states that all have the same labels, a new outgoing transition is added. This transition leads to a "new" state (called s_0 in [DNV90b]) that is labelled by a "new" proposition that occurs in no other label. Another di erence is that in [DNV90b], Kripke structures are assumed to be finite. For finite structures, the condition $\mathit{infstut}_{\equiv_{\mathrm{stut}}}(s)$ is true i s occurs on a cycle of states that all have the same label. Therefore, condition 2 of Definition 6.3.2.5 is not needed in the setting of [DNV90b]. These variations cause some di erences between the proof of Lemma 6.3.2.6 above and the proof[9] of Lemma 3.13 in [DNV90b]. However, having established Lemma 6.3.2.6, fineness of $\equiv_{\mathrm{stut}}$ for $\mathrm{CTL}^{\circ}(\mathsf{U})$, stated in the following lemma, can be proven in the same way as Lemma 3.14 in [DNV90b].

6.3.2.7 L *If* $s \equiv_{\mathrm{stut}} t$, *then* $\forall_{\varphi \in \mathrm{CTL}^{\circ}(\mathsf{U})}\ s \models \varphi \ \Leftrightarrow\ t \models \varphi$.

The converse, abstractness, again only holds for Kripke structures that satisfy a certain (strong) form of finite-branchingness.

6.3.2.8 D *Let* $\equiv$ *be an equivalence relation on* Σ. *We say that* $\mathcal{T}$ *is* finitely branching under $\equiv$-stuttering *i the reflexive transitive closure* $-\!\equiv\!\rightarrow^*$ *of the relation* $-\!\equiv\!\rightarrow$ *is image-finite. For a logic* L, *"finitely branching under* $\equiv_L$*-stuttering" is usually abbreviated by "finitely branching under* L*-stuttering".*

6.3.2.9 L *Let* $\equiv_1$ *and* $\equiv_2$ *be equivalence relations on* Σ *such that* $\equiv_1 \subseteq \equiv_2$. *If* $\mathcal{T}$ *is finitely branching under* $\equiv_2$*-stuttering then* $\mathcal{T}$ *is finitely branching under* $\equiv_1$*-stuttering.*

6.3.2.10 L *Assume that* $\mathcal{T}$ *is finitely branching and also finitely branching under* $\mathrm{CTL}(\mathsf{U})$*-stuttering. If* $\forall_{\varphi \in \mathrm{CTL}(\mathsf{U})}\ s \models \varphi \ \Leftrightarrow\ t \models \varphi$, *then* $s \equiv_{\mathrm{stut}} t$.

[9] Proofs can be found in the full version of that paper.

P . Assume that $\forall_{\varphi \in \mathrm{CTL}(\mathsf{U})}\ s \models \varphi \Leftrightarrow t \models \varphi$. We have to show that $s \equiv_{\mathrm{stut}} t$. Because $\equiv_{\mathrm{stut}}$ is the largest stuttering equivalence, we have to show that the pair (s, t) is an element of some stuttering equivalence $\equiv \subseteq \Sigma \times \Sigma$. We define this relation as follows: $u \equiv v$ i $\forall_{\varphi \in \mathrm{CTL}(\mathsf{U})}\ u \models \varphi \Leftrightarrow v \models \varphi$. Clearly $s \equiv t$. We show that $\equiv$ is a stuttering equivalence. Recall that by Convention 6.1.0.1, $\rightarrow$ is total.

1. It is trivial that $\pounds(s) = \pounds(t)$.
2. Suppose that $\mathit{infstut}_{\equiv}(s)$, i.e. we can choose an s-path $\bar{s}$ such that for every $i \geq 0$, $\bar{s}(i) \equiv s$. We have to show that also $\mathit{infstut}_{\equiv}(t)$. Suppose that this is *not* the case. Then every t-path contains a state from the set $T = \{t'' \mid t \mathrel{-\!\!\equiv\!\!\rightarrow}^* t' \rightarrow t'' \wedge t' \not\equiv t''\}$. Because $\rightarrow$ is total, T is nonempty. Because $\mathcal{T}$ is finitely branching under CTL(U)-stuttering, i.e. under $\equiv$-stuttering, $\{t' \mid t \mathrel{-\!\!\equiv\!\!\rightarrow}^* t'\}$ is finite. Furthermore, because $\mathcal{T}$ is (plainly) finitely branching, also $\{t'' \mid t' \rightarrow t'' \wedge t' \not\equiv t''\}$ is finite for every t'. Hence, because "finite $\times$ finite = finite", T is finite, say $T = \{t''_1, \ldots, t''_n\}$. Because for every $1 \leq i \leq n$, $t''_i \not\equiv t$ and $t \equiv s$, we have $t''_i \not\equiv s$. Hence, by definition of $\equiv$, we can choose formulae $\varphi_i \in \mathrm{CTL}(\mathsf{U})$ such that $s \models \varphi_i$ and $t''_i \not\models \varphi_i$, for every $1 \leq i \leq n$. Because all states on $\bar{s}$ are $\equiv$-equivalent to s, we have $s \models \exists\mathsf{G}(\varphi_1 \wedge \cdots \wedge \varphi_n)$, but because every t-path contains some t''_i, $t \not\models \exists\mathsf{G}(\varphi_1 \wedge \cdots \wedge \varphi_n)$, implying that $s \not\equiv t$, as $\exists\mathsf{G}(\varphi_1 \wedge \cdots \wedge \varphi_n) \in \mathrm{CTL}(\mathsf{U})$. Contradiction.
3. Suppose that $s \mathrel{-\!\!\equiv\!\!\rightarrow} s_1 \mathrel{-\!\!\equiv\!\!\rightarrow} \ldots \mathrel{-\!\!\equiv\!\!\rightarrow} s_{k-1} \rightarrow s_k$ with $k \geq 0$. We have to show that there exists $t \mathrel{-\!\!\equiv\!\!\rightarrow} t_1 \mathrel{-\!\!\equiv\!\!\rightarrow} \ldots \mathrel{-\!\!\equiv\!\!\rightarrow} t_{l-1} \rightarrow t_l$ with $l \geq 0$ and $s_k \equiv t_l$. Suppose (*) that this is *not* the case. Consider the set $T = \{\langle t', t''\rangle \mid t \mathrel{-\!\!\equiv\!\!\rightarrow}^* t' \rightarrow t''\}$. Because $\rightarrow$ is total, T is nonempty. Because $\mathcal{T}$ is finitely branching under $\equiv$-stuttering and also (plainly) finitely branching, T is finite, say $T = \{\langle t'_1, t''_1\rangle, \ldots, \langle t'_n, t''_n\rangle\}$. For every $1 \leq i \leq n$, choose formulae $\varphi'_i, \varphi''_i \in \mathrm{CTL}(\mathsf{U})$ as follows.

   - $\varphi'_i = \mathit{true}$ if $t'_i \equiv t''_i$; otherwise, choose φ'_i such that $t'_i \models \varphi'_i$ (and hence $s \models \varphi'_i$) and $t''_i \not\models \varphi'_i$, which is possible by definition of $\equiv$.
   - choose φ''_i such that $s_k \models \varphi''_i$ and $t''_i \not\models \varphi''_i$ — this is possible by our assumption (*).

   Define $\varphi = \exists\mathsf{U}(\varphi'_1 \wedge \cdots \wedge \varphi'_n, \varphi''_1 \wedge \cdots \wedge \varphi''_n)$. Then clearly $s \models \varphi$. Next, we show that $t \not\models \varphi$. Suppose that, conversely, $t \models \varphi$, i.e. we can choose a t-path $\bar{t}$ such that $\bar{t} \models \mathsf{U}(\varphi'_1 \wedge \cdots \wedge \varphi'_n, \varphi''_1 \wedge \cdots \wedge \varphi''_n)$. This means that we can choose $l \geq 0$ such that $\bar{t}(l) \models \varphi''_1 \wedge \cdots \wedge \varphi''_n$ while for every $0 \leq j < l$, $\bar{t}(j) \models \varphi'_1 \wedge \cdots \wedge \varphi'_n$. On the other hand, every t-path contains one of the t''_i; in particular, by definition of T, we can choose $1 \leq i \leq n$ such that either $\bar{t}(l)$ is t''_i, or we can choose $0 \leq j < l$ such that $\bar{t}(j)$ is t''_i. But in the first case, we have $\bar{t}(l) \not\models \varphi''_i$, implying $\bar{t}(l) \not\models \varphi''_1 \wedge \cdots \wedge \varphi''_n$ and in the second case we have $\bar{t}(j) \not\models \varphi'_i$, implying $\bar{t}(j) \not\models \varphi'_1 \wedge \cdots \wedge \varphi'_n$. In both cases we have a contradiction. So we conclude that $t \not\models \varphi$. But then $s \not\equiv t$, as $\varphi \in \mathrm{CTL}(\mathsf{U})$. Contradiction. □

Although the condition that $\mathcal{T}$ is finitely branching under CTL(U)-stuttering is the weakest condition that su ces to prove the above lemma, it may be impractical

to check. Note that by Lemma 6.3.2.9, it follows that finite branchingness under CTL(U)-stuttering is implied by finite branchingness under Lit-stuttering.

6.3.2.11 C *Assume that $\mathcal{T}$ is finitely branching and also finitely branching under* CTL(U)*-stuttering (recall that by Convention 6.1.0.1, $\rightarrow$ is total). Then $\equiv_{\text{stut}}$ is adequate for both* CTL*(U) *and* CTL(U).

Simplifications of stuttering equivalence

In [DNV90b], de Nicola and Vaandrager prove adequacy of CTL$^\circ$(U) for a notion of stuttering equivalence on KSs that is defined slightly di erently. That their definition (Definition 6.3.2.12 below) yields the same equivalence as $\equiv_{\text{stut}}$ from Definition 6.3.2.5 (on finite KSs without deadlock) is stated by Lemma 6.3.2.13.

The definition of branching bisimulation, originally defined on LTSs, when adapted to KSs in a straightforward fashion (Definition 6.3.2.14 below), seems slightly weaker than stuttering equivalence. Lemma 6.3.2.15 below shows that in fact both definitions result in the same equivalence.

6.3.2.12 D *Consider a symmetric relation $\equiv \subseteq \Sigma \times \Sigma$ such that for every $s, t \in \Sigma$, $s \equiv t$ implies:*

1. $\pounds(s) = \pounds(t)$.
2. *infstut*$_{\equiv}(s)$ *i* *infstut*$_{\equiv}(t)$.
3. *For every $s \rightarrow s'$, there exists $t \mathrel{-\!\!\equiv\!\!\rightarrow} t_1 \mathrel{-\!\!\equiv\!\!\rightarrow} \cdots \mathrel{-\!\!\equiv\!\!\rightarrow} t_{m-1} \rightarrow t_m$ such that $m \geq 0$ and $s' \equiv t_m$.*

The largest such relation is denoted $\equiv_{\text{stut1}}$.

6.3.2.13 L *On every Kripke structure $\mathcal{T}$ satisfying the assumptions of Convention 6.1.0.1, $\equiv_{\text{stut1}} = \equiv_{\text{stut}}$.*

P $\equiv_{\text{stut}} \subseteq \equiv_{\text{stut1}}$ is obvious. Also $\equiv_{\text{stut}} \supseteq \equiv_{\text{stut1}}$ is easily proven. □

6.3.2.14 D *Consider a symmetric relation $\equiv \subseteq \Sigma \times \Sigma$ such that for every $s, t \in \Sigma$, $s \equiv t$ implies:*

1. $\pounds(s) = \pounds(t)$.
2. *infstut*$_{\equiv}(s)$ *i* *infstut*$_{\equiv}(t)$.

3. *For every* $s \to s'$*, there exists*[10] $t \xrightarrow{-\equiv^0\to} t_1 \xrightarrow{-\equiv^0\to} \cdots \xrightarrow{-\equiv^0\to} t_{m-1} \to t_m$ *such that* $m \geq 0$, $t \equiv t_{m-1}$ *and* $s' \equiv t_m$.

The largest such relation is denoted $\equiv_{\text{stut2}}$.

That this definition yields the same equivalence as Definition 6.3.2.5 follows immediately from the following lemma.

6.3.2.15 L (S L K)

- *Let* $s_0, \ldots, s_n \in \Sigma$. *If* $s_0 \xrightarrow{-\equiv^0\to} \cdots \xrightarrow{-\equiv^0\to} s_n$ *and* $s_0 \equiv_{\text{stut2}} s_n$, *then for all* $0 \leq i, j \leq n$, $s_i \equiv_{\text{stut2}} s_j$.
- *The same holds for the divergence blind version of* $\equiv_{\text{stut2}}$ *(i.e. without the infstut condition).*

P .

- Let $s_0, \ldots, s_n \in \Sigma$ such that $s_0 \xrightarrow{-\equiv^0\to} \cdots \xrightarrow{-\equiv^0\to} s_n$ and $s_0 \equiv_{\text{stut2}} s_n$. Define $\equiv = \equiv_{\text{stut2}} \cup \{(s_i, s_j) \mid 0 \leq i, j \leq n\}$. We show below that $\equiv$ satisfies the three conditions of Definition 6.3.2.14. Hence, it is included in $\equiv_{\text{stut2}}$, from which it follows that for all $0 \leq i, j \leq n$, $s_i \equiv_{\text{stut2}} s_j$.

  Let $s, t \in \Sigma$ such that $s \equiv t$. If $s \equiv_{\text{stut2}} t$, then the three conditions of Definition 6.3.2.14 are clearly satisfied. Now we consider the case that s and t are some s_i and s_j.

  1. By assumption, for any $0 \leq i, j \leq n$, $s_i \equiv^0 s_j$, so s and t satisfy condition 1 of Definition 6.3.2.14.
  2. Condition 2 is proven in a similar way as the next point.
  3. Let $0 \leq i, j \leq n$. Without loss of generality, we may assume that $i \leq j$.
     (a) Assume that $s_i \to s'$. So, $s_0 \xrightarrow{-\equiv^0\to} s_1 \xrightarrow{-\equiv^0\to} \cdots \xrightarrow{-\equiv^0\to} s_i \to s'$. Because $s_0 \equiv_{\text{stut2}} s_n$, it can easily be shown (by induction on i) that $s_n \xrightarrow{-\equiv^0\to} s_n^1 \xrightarrow{-\equiv^0\to} \cdots \xrightarrow{-\equiv^0\to} s_n^{k-1} \to s_n^k$ for some $k \geq 0$ such that $s_n^k \equiv_{\text{stut2}} s'$. So, $s_j \xrightarrow{-\equiv^0\to} s_{j+1} \xrightarrow{-\equiv^0\to} \cdots \xrightarrow{-\equiv^0\to} s_n \xrightarrow{-\equiv^0\to} s_n^1 \xrightarrow{-\equiv^0\to} \cdots \xrightarrow{-\equiv^0\to} s_n^{k-1} \to s_n^k$ such that $s_n^k \equiv s'$.
     (b) The "vice versa" is similar but easier.

  So s and t satisfy condition 3 of Definition 6.3.2.14.

- A similar proof can be given for the divergence blind version. □

6.3.2.16 C *On every Kripke structure* $\mathcal{T}$ *satisfying the assumptions of Convention 6.1.0.1,* $\equiv_{\text{stut2}} = \equiv_{\text{stut}}$.

[10] Recall Definition 2.4.0.1, page 22, of $\equiv^0$.

Note that if there exists an s-prefix that cycles back to s again, such that all states on this prefix have the same labels, then $\mathit{infstut}_{\equiv_{\mathrm{stut2}}}(s)$ holds. It is this fact that justifies the definition of stuttering equivalence as dbs-equivalence in the livelock extension ([DNV90b]).

Historical note

Stuttering equivalence was first defined in [BCG88] on finite Kripke structures. The definition given in that paper is as follows (in our terminology).

6.3.2.17 D *The sequence $E_0, E_1, E_2, \ldots$ of equivalence relations on Σ is defined as follows.*

1. $s\ E_0\ t\ i\ \ \pounds(s) = \pounds(t)$.
2. $s\ E_{n+1}\ t\ i$
   (a) *For every $\bar{s} \in \mathit{paths}(s)$ there exists $\bar{t} \in \mathit{paths}(t)$ such that*
      (*) *there exist partitions $B_1, B_2, \ldots$ and $C_1, C_2, \ldots$ of $\bar{s}$ and $\bar{t}$ respectively such that for all $i \geq 0$:*
         i. *B_i and C_i are both non-empty and finite.*
         ii. $\forall_{s' \in B_i, t' \in C_i}\ s'\ E_n\ t'$.
   (b) *Vice versa.*

For paths, $\bar{s}\ E_n\ \bar{t}\ i$ they satisfy condition (*) above.

$s\ E\ t\ i\ \ \forall_{n \in \mathbb{N}}\ s\ E_n\ t$.

One di erence is that in this definition, for every path starting in s there has to be an equivalent path from t, while in all definitions given before, only s-prefixes had to be matched by t-prefixes $t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_m$ (even empty prefixes ($m = 0$) were allowed as matches). An advantage of Definition 6.3.2.17 is that the condition $\mathit{infstut}_{E_n}(s)$ i $\mathit{infstut}_{E_n}(t)$ can be omitted because divergence-sensitivity is already obtained by the formulation in terms of paths. A disadvantage is that the proof of abstractness of E for CTL(U) is complicated by the fact that s and t may be non-equivalent while for every prefix from s there is an equivalent prefix from t. This requires the help of König's lemma. On the other hand, the previous definitions in terms of prefixes do not have this complication, however, there we need an additional state-path lemma of the form of Lemma 6.3.2.6.

Another disadvantage of Definition 6.3.2.17 is that the development of a partition refinement algorithm for stuttering equivalence seems to require an alternative

definition in terms of finite prefixes anyway (see Section 5 of [BCG88]). The condition $\mathit{infstut}_{E_n}(s)$ i $\mathit{infstut}_{E_n}(t)$ still enters the picture there[11].

6.4 Partition Refinement Algorithms

As already noted at the beginning of this chapter, one motivation for being interested in adequate behavioural equivalences for temporal logics is the fact that we can base partition refinement algorithms on them. Indeed, for both bisimulation and stuttering equivalence such algorithms have been developed, see [BFH$^+$92], [BCG88] and [GV90]. In this section, we generalise the common concepts that underly both algorithms, and present a generic scheme of which they are instantiations that are induced by the format of the definition of the behavioural equivalence. This renders the relation between the definition of the behavioural equivalence and the PRA explicit, thereby suggesting directions to look for equivalences having "cheaper" PRAs. These new equivalences as well as the temporal logics that they are adequate for, form the subject of the rest of this chapter.

A *partition* Π of a set A is a set of non-empty, pairwise disjoint subsets of A that covers A, i.e. $\bigcup \Pi = A$. The set of partitions of A is partially ordered by the *refinement* order $\trianglelefteq$ defined by $\Pi_1 \trianglelefteq \Pi_2$ i $\forall_{B_1 \in \Pi_1}\, \exists_{B_2 \in \Pi_2}\, B_1 \subseteq B_2$. With this order, the partitions of A form a lattice (see e.g. [Bir67]); in particular, its glb $\triangle$ is defined by $\Pi_1 \triangle \Pi_2 = \{B_1 \cap B_2 \mid B_1 \in \Pi_1, B_2 \in \Pi_2, B_1 \cap B_2 \neq \emptyset\}$. Clearly, an equivalence relation $\equiv$ on A induces a partition, denoted $\mathit{Part}(A, \equiv)$, consisting of the equivalence classes of $\equiv$. Reversely, the equivalence induced by a partition Π is denoted $\mathit{Eq}(A, \Pi)$. In the sequel we consider partitions and equivalences on Σ. $\mathit{Part}(\equiv)$ abbreviates $\mathit{Part}(\Sigma, \equiv)$ while $\mathit{Eq}(\Pi)$ abbreviates $\mathit{Eq}(\Sigma, \Pi)$.

In order to bring out the analogies in the definitions of bisimulation and stuttering equivalence, we try to make their formats as similar as possible. To begin with, we drop the requirement about infinite stuttering, i.e. we consider in fact Definition 6.3.2.3 of divergence blind stuttering equivalence. The results in [GV90] indicate that a PRA for (divergence sensitive) stuttering equivalence can be obtained by adding a simple preprocessing phase to a PRA for divergence blind stuttering equivalence. Comparing Definition 6.2.0.1 of $\equiv_{\mathrm{bis}}$ to Definition 6.3.2.3 of $\equiv_{\mathrm{dbs}}$, we see that each equivalence is defined as the largest symmetric equivalence $\equiv$ such that:

1. $\equiv \subseteq \equiv^0$.

[11] The *loop* predicate in [BCG88] is actually stronger than our *infstut*. However, it follows from the Stuttering Lemma (Lemma 6.3.2.15) that also in [BCG88] it may be weakened.

2. If $s \equiv t$, then for every s-prefix $\hat{s}$ of a certain form, there exists a t-prefix $\hat{t}$ of similar form.

In the case of bisimulation, this "certain form" of $\hat{s}$ requires that $\hat{s}$ has length 2 and that it ends in a specific $\equiv$-equivalence class; $\hat{t}$ should then also have length 2 and end in the same equivalence class. In the case of divergence blind stuttering equivalence, the lengths of $\hat{s}$ and $\hat{t}$ may be arbitrary but finite, while they should both stay in the same $\equiv$-equivalence class with the exception of their last states, which should both lie in the same di erent class. Thus, in both cases the requirements on the prefixes consist of (1) a restriction on their lengths, and (2) a restriction on the equivalence classes that they "follow".

6.4.0.1 D *Let $\mathcal{B} \subseteq \mathcal{P}(\Sigma)$ be a collection of sets of states and $\bar{s}$ a path or prefix. For $i = 1, 2, \ldots$, let $B_i \in \mathcal{B}$; assume $B_i \neq \emptyset$ and $B_i \neq B_{i+1}$ for every i. We say that $\bar{s}$* follows *$B_1, B_2, \ldots$ i $\bar{s}$ can be partitioned into non-empty blocks $\bar{s}_{I_1}, \bar{s}_{I_2}, \ldots$ such that for every i, every state of $\bar{s}_{I_i}$ is an element of B_i. This definition is extended to sequences that possibly contain adjacent sets that are equal ($B_i = B_{i+1}$); in this case, $\bar{s}$* follows *$B_1, B_2, \ldots$ i $\bar{s}$ follows the sequence obtained by removing, for every i, all B_{i+1} that are equal to B_i.*

The basic refinement step in the algorithms mentioned above is performed by taking blocks B, B' of the current partition, and then splitting B into those states that have prefixes, of appropriate length, following B, B', and those who do not. (For the definition of *prefixes*, see Section 2.4.)

6.4.0.2 D *Let $\mathcal{B} \subseteq \mathcal{P}(\Sigma)$ be a collection of sets of states. A* splitter *is a pair $\langle l, \langle B_1, B_2 \rangle\rangle$ where*[12] *$l \in \mathbb{N} \cup \{\mathbf{fin}\}$, $B_i \in \mathcal{B}$ and $B_i \neq \emptyset$ for $i = 1, 2$.*

Let $S = \langle l, \bar{B} \rangle$ be a splitter and $\hat{s}$ a prefix. $S(\hat{s})$ i $((l \in \mathbb{N} \wedge \mathit{length}(\hat{s}) = l) \vee (l = \mathbf{fin} \wedge \mathit{length}(\hat{s}) < \omega)) \wedge \hat{s}$ follows $\bar{B}$. For a state s, we define $S(s)$ i $\exists_{\hat{s} \in \mathit{prefixes}(s)} S(\hat{s})$.

A set of splitters induces an equivalence relation on states.

6.4.0.3 D *Let T be a collection of splitters. The* splitter equivalence induced by *T, is the equivalence relation $\equiv_T \subseteq \Sigma \times \Sigma$ defined by $s \equiv_T t$ i $\forall_{S \in T} S(s) \Leftrightarrow S(t)$.*

Equipped with these notions, we are now able to define the sets of splitters for bisimulation and dbs-equivalence PRAs.

[12] l will be interpreted as the length of a prefix. The value **fin** then means "any finite length".

6.4.0.4 D *Let $\mathcal{B} \subseteq \mathcal{P}(\Sigma)$.*

- ${}_{\text{bis}}\text{SPL}(\mathcal{B}) = \{\langle 2, \langle B_1, B_2\rangle\rangle \mid B_1, B_2 \in \mathcal{B}, B_1 \neq \emptyset, B_2 \neq \emptyset\}$.
- ${}_{\text{dbs}}\text{SPL}(\mathcal{B}) = \{\langle \mathbf{fin}, \langle B_1, B_2\rangle\rangle \mid B_1, B_2 \in \mathcal{B}, B_1 \neq \emptyset, B_2 \neq \emptyset\}$.

These definitions are relative to a given set $\mathcal{B}$ of state sets. In the partition refinement algorithms, a partition Π of the set of all states is maintained; it is this partition that serves as the parameter to ${}_{\text{bis}}$SPL and ${}_{\text{dbs}}$SPL. Initially, Π is taken equal to the partitioning $\mathit{Part}(\equiv^0)$ (Definition 2.4.0.1). As Π is only *refined* during the course of the algorithm, this guarantees that the equivalence $Eq(\Pi)$ that is induced by the resulting partition Π of the PRA will satisfy condition 1 on page 163. Formally, the algorithms can be viewed as computing, successively for every $n \in \mathbb{N}$, (the partitions induced by) the equivalence relations $\equiv^n_{\text{bis}}$ and $\equiv^n_{\text{dbs}}$. These are the approximants of $\equiv_{\text{bis}}$ and $\equiv_{\text{dbs}}$, defined as follows.

6.4.0.5 D

1. *$s \equiv^0_{\text{bis}} t$ and $s \equiv^0_{\text{dbs}} t$ are both defined as $\pounds(s) = \pounds(t)$.*
2. *$s \equiv^{n+1}_{\text{bis}} t$ i $s \equiv^n_{\text{bis}} t$ and for every $s \to s'$, there exists $t \to t'$ such that $s' \equiv^n_{\text{bis}} t'$.*

   $s \equiv^{n+1}_{\text{dbs}} t$ i $s \equiv^n_{\text{dbs}} t$ and for every $s \mathrel{-\!\equiv^n_{\text{dbs}}\!\to} s_1 \mathrel{-\!\equiv^n_{\text{dbs}}\!\to} \ldots \mathrel{-\!\equiv^n_{\text{dbs}}\!\to} s_{k-1} \to s_k$ such that $k \geq 0$, there exists $t \mathrel{-\!\equiv^n_{\text{dbs}}\!\to} t_1 \mathrel{-\!\equiv^n_{\text{dbs}}\!\to} \ldots \mathrel{-\!\equiv^n_{\text{dbs}}\!\to} t_{l-1} \to t_l$ such that $l \geq 0$ and $s_k \equiv^n_{\text{dbs}} t_l$.

Thus, in every iteration, the algorithms compute $\mathit{Part}(\equiv^{n+1}_{xx})$ ($xx \in \{\text{bis}, \text{stut}\}$) by refining the blocks of $\mathit{Part}(\equiv^n_{xx})$ with regard to the splitters in ${}_{xx}\text{SPL}(\mathit{Part}(\equiv^n_{xx}))$. Under certain conditions on the branchingness of the transition system, this process will eventually distinguish every pair of states that are not $\equiv_{xx}$-equivalent, as is implied by the following lemma.

6.4.0.6 L

1. *Assume that $\mathcal{T}$ is finitely branching. Then $s \equiv_{\text{bis}} t$ i $\forall_{n\in\mathbb{N}}\, s \equiv^n_{\text{bis}} t$.*
2. *Assume that $\mathcal{T}$ is finitely branching and also finitely branching under* Lit-*stuttering. Then $s \equiv_{\text{dbs}} t$ i $\forall_{n\in\mathbb{N}}\, s \equiv^n_{\text{dbs}} t$.*

P Point 1 is standard, see e.g. [HM85]. Point 2 is similar. □

The notion of refining a block with respect to a set of splitters is formalised as follows.

6.4.0.7 D *Let $B \subseteq \Sigma$ and S a set of splitters. $split(B, S) = Part(B, \equiv_S)$.*

The following lemma justifies the idea that for every $n \in \mathbb{N}$, $Part(\equiv_{xx}^{n+1})$ may be computed by splitting the blocks of $Part(\equiv_{xx}^{n})$ with respect to ${}_{xx}\mathrm{SPL}(Part(\equiv_{xx}^{n}))$.

6.4.0.8 L *Let $xx \in \{\mathrm{bis}, \mathrm{stut}\}$, $\equiv \subseteq \Sigma \times \Sigma$ and $n \in \mathbb{N}$. Then the following are equivalent:*

1. $\equiv \subseteq \equiv_{xx}^{n}$ *and for every* $B \in Part(\equiv)$, $split(B, {}_{xx}\mathrm{SPL}(\equiv_{xx}^{n})) = \{B\}$.
2. $\equiv \subseteq \equiv_{xx}^{n+1}$.

P Straightforward from the definitions of $\equiv_{xx}^{n}$ and ${}_{xx}\mathrm{SPL}$. □

This lemma suggests the algorithm of Figure 6.3 to compute $Part(\equiv_{xx})$ on finite transition systems. Note that the transition relation $\rightarrow$ is a parameter of this algorithm, as $split(B, {}_{xx}\mathrm{SPL}(\Pi))$ depends on it.

$\Pi := Part(\equiv^{0})$;
$n := 0$;
stable := *false*;
while not *stable* **do**
  $\{\ \Pi = Part(\equiv_{xx}^{n})\ \}$
  $\Pi' := \emptyset$;
  for each $B \in \Pi$ **do**
    $\Pi' := \Pi' \cup split(B, {}_{xx}\mathrm{SPL}(\Pi))$
  od;
  $\{\ \Pi' = Part(\equiv_{xx}^{n+1})\ \}$
  $n := n + 1$;
  stable := $(\Pi' = \Pi)$;
  $\Pi := \Pi'$
od;

Figure 6.3: A generic partition refinement algorithm.

6.4.0.9 L *Let $xx \in \{\text{bis}, \text{stut}\}$ and assume that $\mathcal{T}$ is finite. Consider the algorithm of Figure 6.3.*

1. *After the nth iteration of the while-loop, $\Pi = \mathit{Part}(\equiv^n_{xx})$.*
2. *After termination of the algorithm, $\Pi = \mathit{Part}(\bigcap_{m\in\mathbb{N}} \equiv^m_{xx})$.*
3. *The algorithm terminates.*

P .

1. This follows from the validity of the assertions $\Pi = \mathit{Part}(\equiv^n_{xx})$ and $\Pi' = \mathit{Part}(\equiv^{n+1}_{xx})$ in the algorithm. For $n = 0$, the first assertion clearly holds by the initialisation of the algorithm. Furthermore, for every n, executing the for-each loop (including the initialisation $\Pi' := \emptyset$) in state $\Pi = \mathit{Part}(\equiv^n_{xx})$ results in a state with $\Pi' = \mathit{Part}(\equiv^{n+1}_{xx}$); this follows from Lemma 6.4.0.8.
2. The algorithm terminates if for every $B \in \Pi$, $\mathit{split}(B, {}_{xx}\mathrm{SPL}(\equiv^n_{xx})) = \{B\}$. By point 1 and Lemma 6.4.0.8 this implies that for every $n' \geq n$, $\mathit{Eq}(\Pi) \subseteq \equiv^{n'}_{xx}$, which in turn implies that $\Pi \subseteq \mathit{Part}(\bigcap_{m\in\mathbb{N}} \equiv^m_{xx})$. Because also $\Pi = \mathit{Part}(\equiv^n_{xx})$, it follows that $\Pi = \mathit{Part}(\bigcap_{m\in\mathbb{N}} \equiv^m_{xx})$.
3. Because $\mathcal{T}$ is finite, the partitions of $\mathcal{T}$'s state set, ordered by the refinement relation $\trianglelefteq$, form a wellfounded poset in which Π strictly decreases for every step (the last excepted) of the while loop. □

6.4.0.10 C *Let $xx \in \{\text{bis}, \text{stut}\}$. If $\mathcal{T}$ is finite, then the algorithm calculates $\mathit{Part}(\equiv_{xx})$ in Π.*

6.5 Flat CTL and CTL*

We are interested in instantiations of the partition refinement algorithm that give good reductions. The algorithm can be "tuned" by varying the definition of the set ${}_{xx}\mathrm{SPL}$ of splitters for a certain equivalence $\equiv_{xx}$. Splitters are interpreted as predicates over states (see Definition 6.4.0.2). So, to obtain better reductions, these predicates have to be less distinctive, i.e. should have uniform valuations over all states of a block more often. The splitters for dbs-equivalence are less distinctive than those for bisimulation because they weaken the condition on the length of a path. In this section, we consider another variation, obtained (from ${}_{\text{dbs}}\mathrm{SPL}$) by weakening the restriction imposed by the blocks that a path has to follow. Instead of taking both blocks from the current partition Π in the algorithm, we only take the second block from Π and choose the first block (cf. B_1 in Definition 6.4.0.4) from the $\equiv^0$-equivalence. In other words, a prefix $\hat{s}$ satisfies such a splitter i along some initial

part all states have the same labelling, and the rest of the prefix stays inside some block from the current partition. We conjecture that the corresponding behavioural equivalence fits with the fragment of CTL*(U) in which the first argument of the Until operator must be a proposition or a boolean combination of propositions. In order to investigate this, let us define this logic as well as the behavioural equivalence we have in mind.

6.5.0.1 D *bool*(Prop) *is the set of propositional formulae (i.e. combinations built using negation and conjunction) over* Prop. *The logic* flatCTL*(U) *is the set of state formulae φ defined inductively by the following grammar, where $p \in$ bool*(Prop).

$$\textit{state formulae:} \quad \varphi \;:=\; p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\psi.$$

$$\textit{path formulae:} \quad \psi \;:=\; \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{U}(p, \psi).$$

The abbreviation $\mathsf{F}\varphi$ is defined as usual (Definition 2.3.0.1). However, $\mathsf{G}\varphi$ abbreviates[13] $\neg\mathsf{F}\neg\varphi$.

Clearly, $\mathsf{G}\varphi$ may be considered as a flatCTL*(U) formula for any $\varphi \in$ flatCTL*(U). The following lemma implies that also $\mathsf{U}(\varphi, p)$ can be expressed in flatCTL*(U):

6.5.0.2 L *Let $p \in$ bool*(Prop) *and $\varphi \in$* flatCTL*(U). *We have* $\mathsf{U}(\varphi, p) \equiv \mathsf{U}(\mathit{true}, p) \wedge \neg\mathsf{U}(\neg p, \neg\varphi \wedge \neg p)$.

P Via CTL*: $\models \mathsf{U}(\varphi, p) \equiv \neg\mathsf{V}(\neg\varphi, \neg p) \equiv \neg\mathsf{W}(\neg p, \neg\varphi \wedge \neg p) \equiv \neg(\mathsf{G}\neg p \vee \mathsf{U}(\neg p, \neg\varphi \wedge \neg p)) \equiv \neg(\neg\mathsf{U}(\mathit{true}, p) \vee \mathsf{U}(\neg p, \neg\varphi \wedge \neg p)) \equiv \mathsf{U}(\mathit{true}, p) \wedge \neg\mathsf{U}(\neg p, \neg\varphi \wedge \neg p)$. The second step uses the relation between V and W (page 24). The third step applies the definition of W, and the fourth the definition of G (Definition 6.5.0.1 above). □

As this logic is a fragment of CTL*, its interpretation is defined by Definition 2.4.1.1. As to the behavioural equivalence, the following seems to be the corresponding adaptation of Definition 6.3.2.5 of $\equiv_{\text{stut}}$. The third clause considers prefixes along which all states, up to the last one, are $\equiv^0$-equivalent; this corresponds to the Until formulae being restricted to propositional formulae in their first arguments. The last states of matching prefixes from s and t should be $\equiv$-equivalent again. This should guarantee that the eventuality expressed by the second argument of the Until, which may be an arbitrary flatCTL*(U) formula again, has the same valuation in both endpoints. The second clause then again prevents that s-prefixes are only matched by t-prefixes of length 1.

[13] Note that this is di erent from Definition 2.3.0.1 as we do not have V.

6.5.0.3 D *Let $\equiv \subseteq \Sigma \times \Sigma$ be a symmetric relation such that for every $s, t \in \Sigma$, $s \equiv t$ implies:*

1. $\pounds(s) = \pounds(t)$.
2. *infstut*$_{\equiv^0}(s)$ *i* *infstut*$_{\equiv^0}(t)$.
3. *For every* $s \xrightarrow{-\equiv^0\rightarrow} s_1 \xrightarrow{-\equiv^0\rightarrow} \cdots \xrightarrow{-\equiv^0\rightarrow} s_{k-1} \rightarrow s_k$ *such that* $k \geq 0$, *there exists* $t \xrightarrow{-\equiv^0\rightarrow} t_1 \xrightarrow{-\equiv^0\rightarrow} \cdots \xrightarrow{-\equiv^0\rightarrow} t_{l-1} \rightarrow t_l$, *such that* $l \geq 0$ *and* $s_k \equiv t_l$.

Then $\equiv$ is called a flat (stuttering) equivalence. *The largest flat equivalence is denoted* $\equiv_{\text{flat}^-}$.

Note that if we would require in addition that $s_{k-1} \equiv_{\text{flat}^-} t_{l-1}$ and $t \equiv_{\text{flat}^-} t_{l-1}$ in point 3 of the above definition, $\equiv_{\text{flat}^-}$ would be the same as $\equiv_{\text{stut}}$. This follows from Lemma 6.3.2.15, the Stuttering Lemma.

The definition of $\equiv_{\text{flat}^-}$ needs to be extended to paths to formulate a state-path lemma (cf. Lemmata 6.2.0.4 and 6.3.2.6). This definition should be such that paths that are $\equiv_{\text{flat}^-}$-equivalent satisfy the same Until formulae from flatCTL$^*(\mathsf{U})$. A similar definition as $\equiv_{\text{stut}}$ for paths, where the paths have to consist of corresponding sequences of $\equiv_{\text{stut}}$-blocks, seems too strong. Intuitively, the following would su ce:

$\bar{s} \equiv_{\text{flat}^-} \bar{t}$ i

1. For every $k \geq 0$ there exists $l \geq 0$ such that $\bar{s}(k) \equiv_{\text{flat}^-} \bar{t}(l)$ and furthermore, letting $\mathit{partit}_{\equiv^0}(\bar{s}_{\{0,\ldots,k-1\}}) = \bar{s}_{I_1}, \ldots, \bar{s}_{I_K}$ and $\mathit{partit}_{\equiv^0}(\bar{t}_{\{0,\ldots,l-1\}}) = \bar{t}_{J_1}, \ldots, \bar{t}_{J_L}$, $K = L$ and for every $0 \leq i \leq K$, every $s' \in \bar{s}_{I_i}$ and $t' \in \bar{t}_{J_i}$, we have $s' \equiv^0 t'$.
2. Vice versa.

However, proving a state-path lemma like Lemma 6.3.2.6 turns out to be impossible. We can only show one of the two halves required for $\equiv_{\text{flat}^-}$-equivalence of paths. Indeed, there is a counter-example showing that $\equiv_{\text{flat}^-}$ is *not* fine for flatCTL$^*(\mathsf{U})$. It is depicted in Figure 6.4. The propositions p, q and r are supposed to be mutually exclusive. Consider the states s_2 and s_5. The reader is invited to verify that $s_2 \equiv_{\text{flat}^-} s_5$. However, the flatCTL$^*(\mathsf{U})$ formula $\varphi = \exists(\mathsf{U}(p, q) \wedge \neg\mathsf{U}(p, p \wedge \neg\exists\mathsf{F}r))$ distinguishes between s_2 and s_5. Namely, the s_2-path $\bar{s} = s_2, s_3, s_4, s_4, \ldots$ clearly satisfies $\mathsf{U}(p, q)$ and does not satisfy $\mathsf{U}(p, p \wedge \neg\exists\mathsf{F}r)$: there is no state on $\bar{s}$ where p holds and at the same time the r-state is not reachable. On the other hand, the s_5-path $\bar{t} = s_5, s_6, s_4, s_4, \ldots$ satisfies both $\mathsf{U}(p, q)$ and $\mathsf{U}(p, p \wedge \neg\exists\mathsf{F}r)$, because the path contains s_6. Hence s_5 satisfies φ while s_2 does not.

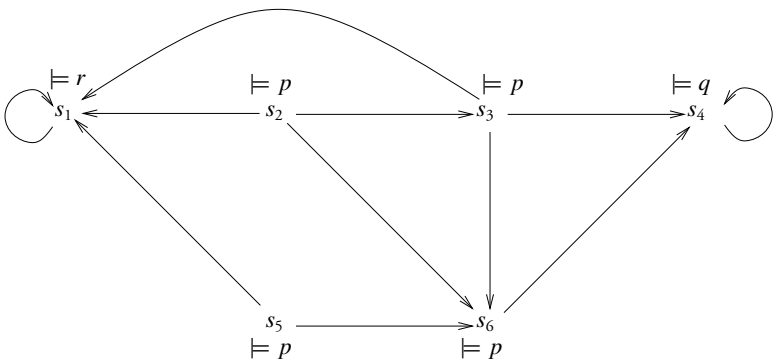


Figure 6.4: $\equiv_{\text{flat}^-}$ is not fine for flatCTL*(U).

We proceed along two lines. First, we restrict the logic flatCTL*(U) in such a way that $\equiv_{\text{flat}^-}$ *is* fine for it. An analysis of the counter-example suggests that it may be the capability of flatCTL*(U) to assert *several* path properties (U-formulae) about a *single* path that renders this logic "too strong" for $\equiv_{\text{flat}^-}$-equivalence. As it turns out in Section 6.5.1 below, the "unstarred" variant of the logic, in which only a single path formula may occur in the scope of every path quantifier, fits nicely with $\equiv_{\text{flat}^-}$-equivalence.

Second, in Section 6.5.2, we strengthen the behavioural equivalence, obtaining a behavioural equivalence adequate for full flatCTL*(U).

6.5.1 flat$^-$CTL(U) and flat equivalence

The counter-example above suggests that the strength of flatCTL*(U) to distinguish between $\equiv_{\text{flat}^-}$-equivalent states lies in its capability to assert various until formulae about the same path. The following restriction delimits this strength.

6.5.1.1 D *The logic* flat$^-$CTL(U) *is the set of state formulae* φ *defined inductively by the following grammar, where* $p \in$ *bool*(Lit).

$$\varphi \;:=\; p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists \mathsf{U}(p, \varphi) \mid \exists \mathsf{G} p.$$

This definition is directly inspired by the clauses of Definition 6.5.0.3. Clause 2 of that definition suggests that formulae of the form $\exists \mathsf{G} p$, with $p \in$ *bool*(Lit), are satisfied in s if and only if they are satisfied in t (actually also the third clause is needed, see the proof of Lemma 6.5.1.2 below). Looking at clause 3, we predict that it will not be di cult to show that when s satisfies a formula of the form $\exists \mathsf{U}(p, \varphi)$,

with $p \in \mathit{bool}(\mathrm{Lit})$, then also t will satisfy this formula. Recall that the operator $\forall\mathsf{U}$ may be defined as a derived operator by $\forall\mathsf{U}(\varphi, p) = \neg\exists\mathsf{U}(\neg p, \neg\varphi \wedge \neg p) \wedge \neg\exists\mathsf{G}\neg p$. Reversely, $\exists\mathsf{G}p$ is equivalent to $\neg\forall\mathsf{U}(\mathit{true}, \neg p)$ (Lemma 2.4.1.2). This means that we could also have replaced $\exists\mathsf{G}p$ by $\forall\mathsf{U}(\varphi, p)$ in the above definition. Thus, we see that the argument of the U operator that has to be restricted depends on the preceding path quantifier.

$\equiv_{\mathrm{flat}^-}$ is fine for flat$^-$CTL(U):

6.5.1.2 L *If* $s \equiv_{\mathrm{flat}^-} t$, *then* $\forall_{\varphi \in \mathrm{flat}^-\mathrm{CTL}(\mathsf{U})}\, s \models \varphi \Leftrightarrow t \models \varphi$.

P . By induction on the structure of the formula.

- Base: $\varphi \in \mathit{bool}(\mathrm{Lit})$. $s \equiv_{\mathrm{flat}^-} t$ implies that $\pounds(s) = \pounds(t)$. From this it follows that $s \models p$ i $t \models p$ for all $p \in \mathit{bool}(\mathrm{Lit})$.
- Induction step:
  1. The cases that φ is a negation or conjunction are straightforward.
  2. $\varphi = \exists\mathsf{U}(p, \varphi')$. Assume that $s \models \varphi$. By Definition 2.4.1.1, this means that there exists an s-path $\bar{s}$ and $n \geq 0$ such that $\bar{s}(n) \models \varphi'$ and for every $0 \leq i < n$, $\bar{s}(i) \models p$. Because $s \equiv_{\mathrm{flat}^-} t$, by clause 3 in Definition 6.5.0.3 there exists a t-path $\bar{t}$ and $m \geq 0$ such that $\bar{s}(n) \equiv_{\mathrm{flat}^-} \bar{t}(m)$ and for every $0 \leq j < m$, $\bar{t}(j) \equiv^0 t$. By the induction hypothesis, it follows from $\bar{s}(n) \models \varphi'$ and $\bar{s}(n) \equiv_{\mathrm{flat}^-} \bar{t}(m)$ that $\bar{t}(m) \models \varphi'$. Because $\bar{s}(0) \models p$ and $s \equiv_{\mathrm{flat}^-} t$ and for every $0 \leq j < m$, $\bar{t}(j) \equiv^0 t$, we have $\bar{t}(j) \models p$ for every $0 \leq j < m$. Hence, $t \models \exists\mathsf{U}(p, \varphi')$.
  3. $\varphi = \exists\mathsf{G}p$. Assume that $s \models \varphi$. By Definition 2.4.1.1, this means that there exists an s-path $\bar{s}$ such that for every $i \geq 0$, $\bar{s}(i) \models p$. Consider $\mathit{partit}_{\equiv^0}(\bar{s}) = B_0, B_1, \ldots$. Along the same lines as the proof of the state-path Lemma 6.3.2.6, using clause 3 from Definition 6.5.0.3 if $\mathit{partit}_{\equiv^0}(\bar{s})$ and in addition clause 2 for the last block if $\mathit{partit}_{\equiv^0}(\bar{s})$ is finite, it is not di cult to show that there exists a t-path $\bar{t}$ such that for $\mathit{partit}_{\equiv^0}(\bar{t}) = C_0, C_1, \ldots$, the states in B_j are $\equiv^0$-equivalent to those in C_j for every j. It follows that $\bar{t}(i) \models p$ for every $i \geq 0$, i.e. $t \models \exists\mathsf{G}p$. □

Reversely, flat$^-$CTL(U) can distinguish any two states that are not $\equiv_{\mathrm{flat}^-}$-equivalent. This follows from the following abstractness result.

6.5.1.3 L *Assume that* $\mathcal{T}$ *is finitely branching and also finitely branching under* Lit*-stuttering. If* $\forall_{\varphi \in \mathrm{flat}^-\mathrm{CTL}(\mathsf{U})}\, s \models \varphi \Leftrightarrow t \models \varphi$, *then* $s \equiv_{\mathrm{flat}^-} t$.

P . Assume that $\forall_{\varphi \in \mathrm{flat}^-\mathrm{CTL}(\mathsf{U})}\, s \models \varphi \Leftrightarrow t \models \varphi$. We have to show that $s \equiv_{\mathrm{flat}^-} t$. Because $\equiv_{\mathrm{flat}^-}$ is the largest flat equivalence, we have to show that the pair (s, t) is an element of some flat equivalence $\equiv \subseteq \Sigma \times \Sigma$. We define this relation as follows: $u \equiv v$ i $\forall_{\varphi \in \mathrm{flat}^-\mathrm{CTL}(\mathsf{U})}\, u \models \varphi \Leftrightarrow v \models \varphi$. Clearly $s \equiv t$. We show that $\equiv$ is a flat equivalence. Recall that by Convention 6.1.0.1, $\rightarrow$ is total.

1. It is trivial that $\pounds(s) = \pounds(t)$.
2. Suppose that $\mathit{infstut}_{\equiv^0}(s)$, i.e. we can choose an s-path $\bar{s}$ such that for every $i \geq 0$, $\bar{s}(i) \equiv^0 s$. We have to show that also $\mathit{infstut}_{\equiv^0}(t)$. Suppose that this is *not* the case. Then every t-path contains a state from the set $T = \{t'' \mid t -\!\!\equiv^0\!\!\to^* t' \to t'' \wedge t' \not\equiv^0 t''\}$. Because $\to$ is total, T is nonempty. Because $\mathcal{T}$ is finitely branching under Lit-stuttering and also (plainly) finitely branching, T is finite, say $T = \{t''_1, \ldots, t''_n\}$. Because for every $1 \leq i \leq n$, $t''_i \not\equiv^0 t$ and $t \equiv s$, we have $t''_i \not\equiv^0 s$. Hence, by definition of $\equiv^0$, we can choose formulae $p_i \in \mathrm{Lit}$ such that $s \models p_i$ and $t''_i \not\models p_i$, for every $1 \leq i \leq n$. Because all states on $\bar{s}$ are $\equiv^0$-equivalent to s, we have $s \models \exists\mathsf{G}(p_1 \wedge \cdots \wedge p_n)$, but because every t-path contains some t''_i, $t \not\models \exists\mathsf{G}(p_1 \wedge \cdots \wedge p_n)$, implying that $s \not\equiv t$, as $\exists\mathsf{G}(p_1 \wedge \cdots \wedge p_n) \in \mathrm{flat}^-\mathrm{CTL}(\mathsf{U})$. Contradiction.
3. Suppose that $s -\!\!\equiv^0\!\!\to s_1 -\!\!\equiv^0\!\!\to \ldots -\!\!\equiv^0\!\!\to s_{k-1} \to s_k$ with $k \geq 0$. We have to show that there exists $t -\!\!\equiv^0\!\!\to t_1 -\!\!\equiv^0\!\!\to \ldots -\!\!\equiv^0\!\!\to t_{l-1} \to t_l$ with $l \geq 0$ and $s_k \equiv t_l$. Suppose (*) that this is *not* the case. Consider the set $T = \{\langle t', t''\rangle \mid t -\!\!\equiv^0\!\!\to^* t' \to t''\}$. Because $\to$ is total, T is nonempty. Because $\mathcal{T}$ is finitely branching under Lit-stuttering and also (plainly) finitely branching, T is finite, say $T = \{\langle t'_1, t''_1\rangle, \ldots, \langle t'_n, t''_n\rangle\}$. For every $1 \leq i \leq n$, choose formulae $p_i \in \mathrm{Lit}$ and $\varphi_i \in \mathrm{flat}^-\mathrm{CTL}(\mathsf{U})$ as follows.

   - $p_i = \mathit{true}$ if $t'_i \equiv^0 t''_i$; otherwise, choose p_i such that $t'_i \models p_i$ (and hence $s \models p_i$) and $t''_i \not\models p_i$, which is possible by definition of $\equiv^0$.
   - choose φ_i such that $s_k \models \varphi_i$ and $t''_i \not\models \varphi_i$ — this is possible by our assumption (*).

   Define $\varphi = \exists\mathsf{U}(p_1 \wedge \cdots \wedge p_n, \varphi_1 \wedge \cdots \wedge \varphi_n)$. Then clearly $s \models \varphi$. Next, we show that $t \not\models \varphi$. Suppose that, conversely, $t \models \varphi$, i.e. we can choose a t-path $\bar{t}$ such that $\bar{t} \models \mathsf{U}(p_1 \wedge \cdots \wedge p_n, \varphi_1 \wedge \cdots \wedge \varphi_n)$. This means that we can choose $l \geq 0$ such that $\bar{t}(l) \models \varphi_1 \wedge \cdots \wedge \varphi_n$ while for every $0 \leq j < l$, $\bar{t}(j) \models p_1 \wedge \cdots \wedge p_n$. On the other hand, every t-path contains one of the t''_i; in particular, by definition of T, we can choose i such that either $\bar{t}(l)$ is t''_i, or we can choose $0 \leq j < l$ such that $\bar{t}(j)$ is t''_i. But in the first case, we have $\bar{t}(l) \not\models \varphi_i$, implying $\bar{t}(l) \not\models \varphi_1 \wedge \cdots \wedge \varphi_n$ and in the second case we have $\bar{t}(j) \not\models p_i$, implying $\bar{t}(j) \not\models p_1 \wedge \cdots \wedge p_n$. In both cases we have a contradiction. So we conclude that $t \not\models \varphi$. But then $s \not\equiv t$, as $\varphi \in \mathrm{flat}^-\mathrm{CTL}(\mathsf{U})$. Contradiction. □

6.5.1.4 C *Assume that $\mathcal{T}$ is finitely branching and also finitely branching under* Lit*-stuttering (recall that by Convention 6.1.0.1, $\to$ is total). Then $\equiv_{\mathrm{flat}^-}$ is adequate for* $\mathrm{flat}^-\mathrm{CTL}(\mathsf{U})$.

A natural question is whether the expressive power of the logic $\mathrm{flat}^-\mathrm{CTL}(\mathsf{U})$ can be strengthened without changing its distinguishing power (cf. the remarks at the end of Section 6.1.1). Adding the "star" is too much, as we have seen above. But it might be possible to add a restricted amount of "path power". We do not go further into this here.

6.5.2 flatCTL*(U) and flat star equivalence

In this subsection we define a behavioural equivalence, stronger than $\equiv_{\text{flat}^-}$, which is adequate for full flatCTL*(U). We have argued above that it is the power of flatCTL*(U) to assert arbitrarily many path properties about a single path that renders it more distinguishing than $\equiv_{\text{flat}^-}$. In order to make the step from flatCTL*(U) to an appropriate behavioural equivalence, we consider the equivalence induced by flatCTL*(U) in game-theoretic terms.

Consider states s and t and suppose that they must satisfy the same flatCTL*(U) formulae. In particular, we consider formulae of the form $\exists\psi$, where ψ is an arbitrary path formula, which may consist of a conjunction of (negations of) smaller path formulae. If t has to satisfy formulae of the same form, then Defender must have a winning strategy to the following two-phase game:

1. Phase 1: Attacker either chooses an s-path, say $\bar{s}$, which should be matched by the choice by Defender of a t-path, say $\bar{t}$, or Attacker chooses a t-path, say $\bar{t}$, which should be matched by the choice by Defender of an s-path, say $\bar{s}$.

   This phase reflects the choice that corresponds to the $\exists$ quantifier in the formula.

2. Phase 2: Attacker chooses either $\bar{s}$ or $\bar{t}$ to proceed. Denoting the result of this choice by $\bar{u}$, Attacker then chooses a state $\bar{u}(k)$ on $\bar{u}$ such that for every $0 \leq i < k$, $\bar{u}(i) \equiv^0 \bar{u}(0)$. Defender now has to proceed from the other path, call it $\bar{v}$ (so, $\bar{v} = \bar{s}$ if $\bar{u} = \bar{t}$ and $\bar{v} = \bar{t}$ if $\bar{u} = \bar{s}$). She should match the move of Attacker with the choice of a position $\bar{v}(l)$ on $\bar{v}$ such that for every $0 \leq i < l$, $\bar{v}(i) \equiv^0 \bar{v}(0)$. The game continues from $\bar{u}(k)$ and $\bar{v}(l)$.

   This phase reflects the statement of an arbitrary path property: The fact that Attacker chooses either $\bar{s}$ or $\bar{t}$ to proceed reflects the fact that this path property may occur in positive or negated form, while the fact that all states up to $\bar{u}(k)$ have to be $\equiv^0$-equivalent carries in it the restriction of Until formulae to a propositional first argument.

Note that this game-theoretic formulation also explains the inadequacy of $\equiv_{\text{flat}^-}$ for flatCTL*(U). The problem is that Defender has to choose $\bar{t}$ in Phase 1, without knowing which k Attacker is going to choose in Phase 2: Definition 6.5.0.3 of $\equiv_{\text{flat}^-}$ only guarantees that Defender can match any move of Attacker *in which the choice for k is made at the same moment at which $\bar{s}$ is chosen.* After these observations, the following definition suggests a plausible candidate for a behavioural flatCTL*(U) equivalence.

6.5.2.1 D *Let $\equiv \subseteq \Sigma \times \Sigma$ be a symmetric relation such that for every $s, t \in \Sigma$, $s \equiv t$ implies:*

1. $\pounds(s) = \pounds(t)$.
2. *infstut*$_{\equiv}(s)$ *i* *infstut*$_{\equiv}(t)$.
3. *For every $\hat{s} \in$ prefixes(s) there exists $\hat{t} \in$ prefixes(t) such that:*
   (a) *For every $\hat{s}_0 \xrightarrow{\equiv^0} \cdots \xrightarrow{\equiv^0} \hat{s}_{k-1} \to \hat{s}_k$ such that $0 \leq k <$ length$(\hat{s})$, there exists $\hat{t}_0 \xrightarrow{\equiv^0} \cdots \xrightarrow{\equiv^0} \hat{t}_{l-1} \to \hat{t}_l$ such that $0 \leq l <$ length$(\hat{t})$ and $\hat{s}_k \equiv \hat{t}_l$.*
   (b) *Vice versa.*

Then $\equiv$ is called a flat star (stuttering) equivalence. *The largest flat star equivalence is denoted $\equiv_{\text{flat}^*}$.*

$\equiv_{\text{flat}^*}$ is extended to paths by defining $\bar{s} \equiv_{\text{flat}^*} \bar{t}$ i

1. *For every $k \geq 0$ there exists $l \geq 0$ such that $\bar{s}(k) \equiv_{\text{flat}^*} \bar{t}(l)$ and furthermore, letting partit$_{\equiv^0}(\bar{s}_{\{0,\ldots,k-1\}}) = \bar{s}_{I_1}, \ldots, \bar{s}_{I_K}$ and partit$_{\equiv^0}(\bar{t}_{\{0,\ldots,l-1\}}) = \bar{t}_{J_1}, \ldots, \bar{t}_{J_L}$, $K = L$ and for every $0 \leq i \leq K$, every $s' \in \bar{s}_{I_i}$ and $t' \in \bar{t}_{J_i}$, we have $s' \equiv^0 t'$.*
2. *Vice versa.*

We can now prove the following "state-path lemma".

6.5.2.2 L *If $s \equiv_{\text{flat}^*} t$, then for every $\bar{s} \in$ paths(s) there exists $\bar{t} \in$ paths(t) such that $\bar{s} \equiv_{\text{flat}^*} \bar{t}$.*

P The structure of the proof is similar to that of Lemma 6.3.2.6. Let partit$_{\equiv_{\text{flat}^*}}(\bar{s})$ be $B_0, B_1, \ldots$. For every $i \geq 0$ for which B_i exists, let b_i be the first state of B_i. Let $c_0 = t$. By point 3 in Definition 6.5.2.1, there exists a t-prefix $\hat{t}$ such that for every $0 \leq k \leq$ length(B_0) (note that by definition of B_0, all states on it are $\equiv^0$-equivalent, and that $\bar{s}($length$(B_0)) = b_1$, if length$(B_0) < \omega$), there exists $0 \leq l \leq$ length$(\hat{t})$ such that for every $0 \leq j < l$, $\hat{t}(j) \equiv^0 t$ and $\bar{s}(k) \equiv_{\text{flat}^*} \hat{t}(l)$, and vice versa. Consider the shortest such t-prefix, $\hat{t}'$. Clearly, all states on $\hat{t}'$, with the exception of its last, are $\equiv^0$-equivalent. Define C_0 to be $\hat{t}'$ with its last state excepted, while c_1 (the first state of block C_1 to be defined) is defined to be the last state of $\hat{t}'$. This way, we can inductively define states c_i and blocks C_i for all $i \geq 0$ for which B_i exists. If some B_i is infinite, then point 2 in Definition 6.5.2.1 guarantees the existence of an appropriate C_i. It is now easily seen that for the path $\bar{t}$ formed by $C_0, C_1, \ldots$, we have $\bar{s} \equiv_{\text{flat}^*} \bar{t}$. □

Fineness now follows easily.

6.5.2.3 L *If* $s \equiv_{\text{flat}^*} t$, *then* $\forall_{\varphi \in \text{flatCTL}^*(\mathsf{U})}\ s \models \varphi \ \Leftrightarrow\ t \models \varphi$.

P . We prove the following two points by induction on the structure of the formula.

1. If $s \equiv_{\text{flat}^*} t$, then for all state formulae $\varphi \in \text{flatCTL}^*(\mathsf{U})$, $s \models \varphi$ i $t \models \varphi$.
2. For paths $\bar{s}$ and $\bar{t}$: if $\bar{s} \equiv_{\text{flat}^*} \bar{t}$, then for all path formulae φ that occur in $\text{flatCTL}^*(\mathsf{U})$ formulae, $\bar{s} \models \varphi$ i $\bar{t} \models \varphi$.

- Base: $\varphi \in \text{Lit}$. $s \equiv_{\text{flat}^*} t$ implies that $\pounds(s) = \pounds(t)$. From this it follows that $s \models p$ i $t \models p$ for all $p \in \text{Lit}$.
- Induction step:
  1. The cases that φ is a conjunction or negation of state or path formulae, or a state formula interpreted over a path, are straightforward.
  2. $\varphi = \mathsf{U}(p, \varphi')$. Assume that $\bar{s} \models \varphi$. By Definition 2.4.1.1, this means that we can choose $k \geq 0$ such that $\bar{s}(k) \models \varphi'$ and for every $0 \leq i < k$, $\bar{s}(i) \models p$. By definition of $\bar{s} \equiv_{\text{flat}^*} \bar{t}$, there exists $l \geq 0$ such that $\bar{t}(l) \equiv_{\text{flat}^*} \bar{s}(k)$ and for every $0 \leq j < l$, there exists $0 \leq i < k$ such that $\bar{s}(i) \equiv^0 \bar{t}(j)$. Using the induction hypothesis, it follows that $\bar{t} \models \varphi$.
  3. $\varphi = \exists\varphi'$. Straightforward using Lemma 6.5.2.2. □

For the other direction, abstractness, we again need to impose certain forms of finite branchingness.

6.5.2.4 L *Assume that* $\mathcal{T}$ *is finitely branching and also finitely branching under* Lit*-stuttering. If* $\forall_{\varphi \in \text{flatCTL}^*(\mathsf{U})}\ s \models \varphi \ \Leftrightarrow\ t \models \varphi$, *then* $s \equiv_{\text{flat}^*} t$.

P . Assume that $\forall_{\varphi \in \text{flatCTL}^*(\mathsf{U})}\ s \models \varphi \ \Leftrightarrow\ t \models \varphi$. We have to show that $s \equiv_{\text{flat}^*} t$. Because $\equiv_{\text{flat}^*}$ is the largest flat star equivalence, we have to show that the pair (s, t) is an element of some flat star equivalence $\equiv \subseteq \Sigma \times \Sigma$. We define this relation as follows: $u \equiv v$ i $\forall_{\varphi \in \text{flatCTL}^*(\mathsf{U})}\ u \models \varphi \ \Leftrightarrow\ v \models \varphi$. Clearly $s \equiv t$. We show that $\equiv$ is a flat star equivalence. Recall that by Convention 6.1.0.1, $\rightarrow$ is total.

1. It is trivial that $\pounds(s) = \pounds(t)$.
2. Suppose that $\mathit{infstut}_{\equiv}(s)$, i.e. we can choose an s-path $\bar{s}$ such that for every $i \geq 0$, $\bar{s}(i) \equiv s$. We have to show that also $\mathit{infstut}_{\equiv}(t)$. Suppose that this is *not* the case. Then every t-path contains a state from the set $T = \{t'' \mid t \mathrel{-\!\!\equiv\!\!\rightarrow^*} t' \rightarrow t'' \ \wedge\ t' \not\equiv t''\}$. Because $\rightarrow$ is total, T is nonempty. Because $\mathcal{T}$ is finitely branching under Lit-stuttering and also (plainly) finitely branching, this implies by Lemma 6.3.2.9 that T is finite, say $T = \{t''_1, \ldots, t''_n\}$. Because for every $1 \leq i \leq n$, $t''_i \not\equiv t$ and $t \equiv s$, we have $t''_i \not\equiv s$. Hence, by definition of $\equiv$, we can choose formulae $\varphi_i \in \text{flatCTL}^*(\mathsf{U})$ such that $s \models \varphi_i$ and $t''_i \not\models \varphi_i$, for every $1 \leq i \leq n$. Because all states on $\bar{s}$ are

$\equiv$-equivalent to s, we have $s \models \exists\mathsf{G}(\varphi_1 \wedge \cdots \wedge \varphi_n)$, but because every t-path contains some t_i'', $t \not\models \exists\mathsf{G}(\varphi_1 \wedge \cdots \wedge \varphi_n)$, implying that $s \not\equiv t$, as $\exists\mathsf{G}(\varphi_1 \wedge \cdots \wedge \varphi_n)$ is equivalent to a flatCTL*(U) formula by Lemma 6.5.0.2. Contradiction.

3. Let $\hat{s}$ be an s-prefix. We have to show that there exists a t-prefix $\hat{t}$ such that:

   (a) For every $\hat{s}_0 \mathrel{-\!\equiv^0\!\to} \cdots \mathrel{-\!\equiv^0\!\to} \hat{s}_{k-1} \to \hat{s}_k$ such that $0 \leq k < \mathit{length}(\hat{s})$, there exists $\hat{t}_0 \mathrel{-\!\equiv^0\!\to} \cdots \mathrel{-\!\equiv^0\!\to} \hat{t}_{l-1} \to \hat{t}_l$ such that $0 \leq l < \mathit{length}(\hat{t})$ and $\hat{s}_k \equiv \hat{t}_l$.

   (b) Vice versa.

   (*) Suppose that this is *not* the case. Consider the set $T = \{\langle t', t''\rangle \mid t \mathrel{-\!\equiv^0\!\to}{}^* t' \to t''\}$. Because $\to$ is total, T is nonempty. Because $\mathcal{T}$ is finitely branching under Lit-stuttering and also (plainly) finitely branching, T is finite, say $T = \{\langle t_0', t_0''\rangle, \ldots, \langle t_n', t_n''\rangle\}$. We let M be the largest number such that $\hat{s}(0) \mathrel{-\!\equiv^0\!\to} \hat{s}(1) \mathrel{-\!\equiv^0\!\to} \cdots \mathrel{-\!\equiv^0\!\to} \hat{s}(M-1) \to \hat{s}(M)$. This implies that $\hat{s}(M-1) \not\equiv^0 \hat{s}(M)$. Therefore, we can choose $p \in \mathrm{Lit}$ such that $\hat{s}(M-1) \models p$ (and hence also $\hat{s}(i) \models p$ for every $0 \leq i \leq M-1$) and $\hat{s}(M) \not\models p$. Furthermore, for every $0 \leq k \leq M$ and $0 \leq j \leq n$, choose formulae $p_j \in \mathrm{Lit}$ and $\varphi_{k,j} \in$ flatCTL*(U) as follows.

   - $p_j = \mathit{true}$ if $t_j' \equiv^0 t_j''$; otherwise, choose p_j such that $t_j' \models p_j$ (and hence $s \models p_j$) and $t_j'' \not\models p_j$, which is possible by definition of $\equiv^0$.
   - $\varphi_{k,j} = \mathit{true}$ if $\hat{s}(k) \equiv t_j''$; otherwise, choose $\varphi_{k,j}$ such that $\hat{s}(k) \models \varphi_{k,j}$ and $t_j'' \not\models \varphi_{k,j}$, which is possible by definition of $\equiv$.

   For $0 \leq k \leq M$, define $\psi_k = \mathsf{U}(p_1 \wedge \cdots \wedge p_n, \varphi_{k,1} \wedge \cdots \wedge \varphi_{k,n})$. Furthermore, for $0 \leq j \leq n$, define $\xi_j = \mathsf{U}(\varphi_{0,j} \vee \cdots \vee \varphi_{M-1,j}, \neg p)$. Define $\varphi = \exists((\bigwedge_{0\leq k\leq M} \psi_k) \wedge (\bigwedge_{0\leq j\leq n} \xi_j))$.
   Then $s \models \varphi$, as can be seen as follows. Consider an s-path $\bar{s}$ that is an extension of $\hat{s}$ (such a path exists because $\to$ is total); so $\bar{s}(k) = \hat{s}(k)$ for every $0 \leq k \leq M$. First, we show that $\bar{s} \models \psi_k$ for every $0 \leq k \leq M$. Let $0 \leq k \leq M$. Then by definition of the $\varphi_{k,j}$, for every $0 \leq j \leq n$, $\bar{s}(k) \models \varphi_{k,j}$ while for every $0 \leq i < k$, by definition of the p_j and by the fact that s is $\equiv^0$-equivalent to $\bar{s}(i)$, we have $\bar{s}(i) \models p_0 \wedge \cdots \wedge p_n$. Second, we show that $\bar{s} \models \xi_j$ for every $0 \leq j \leq n$. Let $0 \leq j \leq n$. By definition of p, we have $\bar{s}(M) \models \neg p$. Furthermore, for every $0 \leq k < M$, we have $\bar{s}(k) \models \varphi_{k,j}$ by definition of the $\varphi_{k,j}$, so $\bar{s}(k) \models \varphi_{0,j} \vee \cdots \vee \varphi_{M-1,j}$.
   Next, we show that $t \not\models \varphi$. Consider a t-path $\bar{t}$ and suppose (**) $\bar{t} \models \bigwedge_{0\leq k\leq M} \psi_k$. We show that then there exists $0 \leq j \leq n$ such that $\bar{t} \not\models \xi_j$. Our first observation is that by assumption (**), it must be the case that for every $0 \leq k \leq M$, there exists $l \geq 0$ such that for every $0 \leq j < l$, $\bar{t}(j) \equiv^0 t$ and $\hat{s}(k) \equiv \bar{t}(l)$. Namely, suppose it were *not* the case. Then we can choose $0 \leq k' \leq M$ such that for every $l \geq 0$, [there exists $0 \leq j < l$ such that $\bar{t}(j) \not\equiv^0 t$] or [$\hat{s}(k') \not\equiv \bar{t}(l)$]. In the first case, $p_0 \wedge \cdots \wedge p_n$ does not hold for all $0 \leq j < l$ and in the second, $\varphi_{k',l}$ does not hold in $\bar{t}(l)$, hence $\varphi_{k',1} \wedge \cdots \wedge \varphi_{k',n}$ does not hold in $\bar{t}(l)$. But that means that $\bar{t} \not\models \psi_{k'}$. Contradiction with assumption (**). So for every $0 \leq k \leq M$, there exists $l \geq 0$ such that for every $0 \leq j < l$, $\bar{t}(j) \equiv^0 t$ and $\hat{s}(k) \equiv \bar{t}(l)$. By the definition of M, it is also the case for

such a k that for every $0 \leq i < k$, we have $\hat{s}(i) \equiv^0 s$. So, condition 3a above holds. By assumption (*), it must then be the case that condition 3b does not hold, i.e. we can choose $l' \geq 0$ such that for every $0 \leq j < l'$, $\bar{t}(j) \equiv^0 t$ and for every $k \geq 0$ such that $\hat{s}(i) \equiv^0 s$ for every $0 \leq i < k$, we have $\hat{s}(k) \not\equiv \bar{t}(l')$. Let N be the largest number such that $\bar{t}(0) \mathrel{-\!\!\equiv^0\!\!\to} \bar{t}(1) \mathrel{-\!\!\equiv^0\!\!\to} \cdots \mathrel{-\!\!\equiv^0\!\!\to} \bar{t}(N-1) \to \bar{t}(N)$. Clearly, we also have $l' \leq N$. We show that $\xi_{l'}$ cannot hold. Namely, if it has to hold along $\bar{t}$, then the eventuality $\neg p$ can only be fulfilled in $\bar{t}(N)$ or beyond it — this follows from the definition of p, the fact that $s \equiv t$, and the definition of N. That means that the state formula $\varphi_{0,l'} \vee \cdots \vee \varphi_{M-1,l'}$ has to hold in $\bar{t}(j)$ for every $0 \leq j < N$. However, it does not hold in $\bar{t}(l')$. This follows from the definition of the $\varphi_{k,j}$ and from the fact that by definition of N, for every $0 \leq j < N$, $\bar{t}(j)$ is equal to some t''_h $(0 \leq h \leq n)$.

We conclude that φ distinguishes between s and t. By Lemma 6.5.0.2, it follows that then there is also a formula $\varphi' \in \text{flatCTL}^*(\mathsf{U})$ that distinguishes between s and t, from which it follows that $s \not\equiv t$. Contradiction. So assumption (*) cannot be true.□

6.5.2.5 C *Assume that $\mathcal{T}$ is finitely branching and also finitely branching under* Lit*-stuttering (recall that by Convention 6.1.0.1, $\to$ is total). Then $\equiv_{\text{flat}^*}$ is adequate for* flatCTL$^*(\mathsf{U})$.

A surprise

So far, we have shown that flatCTL$^*(\mathsf{U})$ and flat$^-$CTL(U) have di erent distinguishing powers[14], while both for the case CTL*/CTL and the case CTL$^*(\mathsf{U})$/CTL(U) the starred and unstarred versions of the logic yield the same induced equivalence. Figure 6.5 gives the picture that we have established so far. Logics that occur on the same line have equal distinguishing powers (the induced behavioural equivalences are also given on the same line, between parentheses), while logics that are placed higher have equal or more distinguishing power than lower ones. A continuous line between a higher and a lower logic means that the higher logic has strictly more distinguishing power. In the beginning of this section we have seen an example separating $\equiv_{\text{flat}^*}$ and $\equiv_{\text{flat}^-}$, while separating examples for $\equiv_{\text{bis}}$ and $\equiv_{\text{stut}}$ are easy to construct. A question that remained open so far is whether $\equiv_{\text{stut}}$ is strictly finer than $\equiv_{\text{flat}^*}$. The following lemma implies that there is no separating example: $\equiv_{\text{stut}}$ and $\equiv_{\text{flat}^*}$ coincide.

6.5.2.6 L $\equiv_{\text{stut}} = \equiv_{\text{flat}^*}$.

[14] This is not really surprising, as it can be argued that flat$^-$CTL(U), when compared to flatCTL$^*(\mathsf{U})$, not only has the usual restriction that there may only be a single Until formula inside the scope of a path quantifier, but the use of the negation inside this scope is restricted as well. This is the reason why we did not call it flatCTL(U).

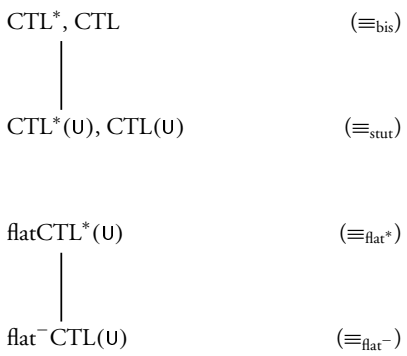


Figure 6.5: Di erent distinguishing powers.

P Because $\equiv_{\text{stut}}$ is adequate for CTL*(U), $\equiv_{\text{flat}^*}$ is adequate for flatCTL*(U), and flatCTL*(U) $\subseteq$ CTL*(U), we clearly have $\equiv_{\text{stut}} \subseteq \equiv_{\text{flat}^*}$. In order to prove $\equiv_{\text{stut}} \supseteq \equiv_{\text{flat}^*}$, we have to show that $\equiv_{\text{flat}^*}$ satisfies the conditions in Definition 6.3.2.5. Points 1 and 2 are easy. As to point 3, assume that $s \xrightarrow{} \equiv_{\text{flat}^*} \xrightarrow{} s_1 \xrightarrow{} \equiv_{\text{flat}^*} \xrightarrow{} \ldots \xrightarrow{} \equiv_{\text{flat}^*} \xrightarrow{} s_{k-1} \xrightarrow{} s_k$ such that $k \geq 0$. By point 3 in Definition 6.5.2.1 of $\equiv_{\text{flat}^*}$, it is easy to see that there exists $t \xrightarrow{} \equiv^0 \xrightarrow{} t_1 \xrightarrow{} \equiv^0 \xrightarrow{} \ldots \xrightarrow{} \equiv^0 \xrightarrow{} t_{l-1} \xrightarrow{} t_l$ such that $l \geq 0$ and $s_k \equiv_{\text{flat}^*} t_l$. Next, we show that any two states t_j and $t_{j'}$ with $0 \leq j \leq j' < l$ are $\equiv_{\text{flat}^*}$-equivalent. Let $0 \leq j \leq j' < l$. By point 3 in Definition 6.5.2.1 of $\equiv_{\text{flat}^*}$, we can choose $0 \leq i \leq i' < k$ such that $t_j \equiv_{\text{flat}^*} s_i$ and $t_{j'} \equiv_{\text{flat}^*} s_{i'}$. By definition, $s_i \equiv_{\text{flat}^*} s_{i'}$. So $t_j \equiv_{\text{flat}^*} t_{j'}$. □

6.6 Partition Refinement for Flat Logics

In this section we indicate how a partition refinement algorithm for flat equivalence can be obtained by instantiating the generic algorithm of Figure 6.3 in an appropriate way. As $\equiv_{\text{flat}^*}$ coincides with $\equiv_{\text{stut}}$, we only consider $\equiv_{\text{flat}^-}$ here. Like in Section 6.4, we sketch a PRA for the "divergence blind" variant of this equivalence, and then indicate how it can be adapted to obtain a PRA for $\equiv_{\text{flat}^-}$. We define $\equiv_{\text{dbflat}}$ as in Definition 6.5.0.3, but without clause 2. It is straightforward to define in addition the approximants $\equiv^n_{\text{dbflat}}$ (cf. Definition 6.4.0.5) and to prove that for a transition system $\mathcal{T}$ that is finitely branching and also finitely branching under Lit-stuttering, $s \equiv_{\text{dbflat}} t$ i $\forall_{n \in \mathbb{N}}\, s \equiv^n_{\text{dbflat}} t$. (cf. Lemma 6.4.0.6).

6.6.0.1 D *Let* $\mathcal{B} \subseteq \mathcal{P}(\Sigma)$.

$$_{\text{flat}}\text{SPL}(\mathcal{B}) = \{\langle \mathbf{fin}, \langle B_1, B_2 \rangle\rangle \mid B_1 \in \mathit{Part}(\equiv^0), B_2 \in \mathcal{B}\}.$$

Lemma 6.4.0.8 now straightforwardly extends to the case $xx = \text{dbflat}$, implying that also Lemma 6.4.0.9 holds for this case, and hence the algorithm of Figure 6.3 can be used to compute $\mathit{Part}(\equiv_{\text{dbflat}})$.

What is left is to indicate how this algorithm is adapted for the divergence sensitive version of the equivalence, $\equiv_{\text{flat}^-}$. This turns out to be simple. Unlike the definition of $\equiv_{\text{stut}}$, the second clause of Definition 6.5.0.3, $\mathit{infstut}_{\equiv_0}(s)$ i $\mathit{infstut}_{\equiv_0}(t)$, does not depend on the equivalence $\equiv$ that is being defined by this fixpoint definition. This implies that $\equiv_{\text{flat}^-}$ can alternatively be defined by $s \equiv_{\text{flat}^-} t$ i $[s \equiv_{\text{dbflat}} t$ and $\mathit{infstut}_{\equiv_0}(s)$ i $\mathit{infstut}_{\equiv_0}(t)]$. As a result, the PRA for $\equiv_{\text{dbflat}}$ is adapted for $\equiv_{\text{flat}^-}$ by adding a postprocessing phase that separates the states that have infinite Lit-stuttering from those that have not. If $\mathcal{T}$ is finite, this boils down to detecting cycles within $\equiv_{\text{dbflat}}$ classes.

6.7 Related Work

In modal logic, bisimulation-like concepts have been around for a long time, e.g. the *zig-zag relations* introduced in the 60s (a survey can be found in [Ben94]), and the *p-relations* in [Ben76].

In computer science, bisimulation is usually attributed to Park, who introduced the term in [Par81], in the context of automata. Milner introduced the notion of simulation in [Mil71] and observational equivalence in [Mil80]. Branching bisimulation has been proposed as an alternative to observational equivalence in, among others, [GW89, vG90a, vG93a, vG93b]. A large variety of behavioural equivalences has been defined and studied over the past ten years; see [DN87] for a discussion. Also [BK90] contains a lot of material on this topic.

The conception of bisimulation took place independent of the development of branching-time temporal logics like CTL ([CE81]), CTL* ([EH86]), μ-calculus ([Koz83]) and Hennessy-Milner Logic ([HM85]). Several later studies have investigated the links between bisimulation and the equivalences induced by these logics ([HM80, GS84, HM85, Sti89, BCG88, LGS$^+$95, Cho95]) as well as between variations on these notions ([BR83, BCG88, DNV90b, BFG$^+$91, GKP92, vBvES94]). In [Jos90], the results of [BCG88] are generalised by considering Kripke structures together with constraints on the interaction with the environment. In particular this answers the question how to characterise fairness. [ASB$^+$94] also studies the equivalences that are induced by CTL and CTL* on Kripke structures with fairness constraints.

Partition refinement algorithms originate in automata theory. Algorithms for minimisation of automata (see [Hop71]) form a precursor; optimisations and adaptations have been presented in [PT87, KS90, BFH$^+$92]. For the case of CTL*(U)/stut-

tering equivalence, partition refinement algorithms can be found in [BCG88, GV90].

The reader who is interested in game characterisations of equivalences could have a look at [Ehr61, Fra54, IK87, Tho93, NC94].

6.8 Concluding Remarks

We have investiged the correspondences between a number of temporal logics and behavioural equivalences, and also between behavioural equivalences and partition refinement algorithms. After recalling the adequacy result of bisimulation for CTL and CTL*, we dropped the Next operator and presented adequacy results of stuttering equivalence for the nextless versions of these logics. Although most of these results are not new, there are a few places where we have established generalisations of older results. Most notably, we did not require finiteness of the transition system, while this is an essential condition in both [BCG88] and [DNV90b]. Indeed, in those papers a stronger correspondence between stuttering equivalence and CTL(U) is proven, namely that every (finite) Kripke structure can be characterised (up to stuttering equivalence) by a single CTL(U) formula[15]. For adequacy, only a certain form of finite branchingness is needed.

These expositions served as a step up to the presentation of a generic partition refinement algorithm in Section 6.4. Central to this algorithm is the notion of *splitter*. We showed how partition refinement algorithms for the aforementioned logics can be obtained by varying the definition of the splitter. Since splitters lie at the heart of the algorithm, its complexity may be expected to be tunable through these splitters. The identification of a "cheaper" notion of splitter has led us to consider (see Section 6.5) further restrictions on the CTL* fragments, resulting in the definition of *flat* logics. The "starred" version that we considered, flatCTL*(U), is obtained from CTL*(U) by restricting the first argument of Until formulae to boolean combinations of propositions. On the other hand, we defined an "unstarred" version, flat$^-$CTL(U), that not only has the usual restriction that there may only be a single Until formula inside the scope of a path quantifier, but also that it restricts the use of the negation inside this scope.

The distinguishing power of flat$^-$CTL(U) is characterised by the equivalence $\equiv_{\text{flat}^-}$. This equivalence is similar to that of *delay bisimulation* introduced in [Mil83]. The only di erence is that the latter is defined over edge-labelled transition systems. So, when flat$^-$CTL(U) is reinterpreted over such transition systems (for details of the di erence between the state and action-based approaches see [DNV90a]), its

[15] [Pnu86], which defines a number of "compatibility" types between logics and equivalences, calls this *expressivity*.

induced equivalence will probably coincide with delay equivalence. To the best of our knowledge, no modal characterisation of this equivalence has been given before.

A surprise occurred when investigating the di erence between $\equiv_{\text{flat}^*}$, the equivalence induced by flatCTL$^*(\mathsf{U})$, and stuttering equivalence: it turns out that there is no di erence. As a consequence, we have shown that nextless CTL* can be flattened without a ecting its distinguishing power.

Finally, Section 6.6 showed that a partition refinement algorithm for restricted flat equivalence, $\equiv_{\text{flat}^-}$, can be obtained by instantiating our generic algorithm with the abovementioned "cheaper" notion of splitter.

Chapter 7

In Conclusion

This final chapter looks back and evaluates the research presented in this thesis. We close by briefly looking ahead.

This chapter evaluates the research presented in this thesis, focussing on the overall picture. After a short summary, we start by briefly looking back at the original research goal, and shed some light on the choice of the specific topics of the chapters. Then, we summarise the main achievements and conclusions. This final chapter ends with a look ahead at on-going and future work. For more specific comments, as well as pointers to related work, the reader is referred to the concluding sections at the ends of the individual chapters.

The topics that have been covered by this thesis are grouped around two main themes, to wit, weak and strong preservation of properties. The context is the application of abstraction methods to alleviate the state-explosion problem in model checking. Weak preservation then means that temporal properties that hold for the abstraction also hold for the original system. Strong preservation requires that in addition, any property absent in the abstraction fails to hold for the original system as well. Besides the investigation of appropriate notions of abstraction and the justification of corresponding preservation results, a major concern of a theory of abstraction is the *construction* of such abstractions. This thesis suggests Abstract Interpretation as an appropriate framework for the construction of weakly-preserving abstractions. For strong preservation, the partition-refinement paradigm is considered.

7.1 Research Goal & Approach

The starting point of our research was to investigate the extension of Abstract Interpretation to reactive systems. This goal became more specific when this rather general topic was narrowed down to the question how to use Abstract Interpretation to tackle the state explosion in model checking. Partial answers turned into notes and articles containing, basically, the ideas presented in Chapter 4 of this thesis, Section 4.8 excepted. The theory of Abstract Interpretation has traditionally been focussed on program *analysis* as explained in the first chapter. It aims at the automatic examination of programs to find properties that can subsequently be used for the optimisation or parallellisation of code for instance. Thus, the theory is based on weak preservation of properties: any property that can be detected in the abstraction increases the knowledge about the concrete system and improves the result. Properties that do not hold in the abstraction may or may not hold in the concrete system — we cannot know and hence such information just does not contribute. One may wonder how e ective this method is when employed in the context of model checking, where definite absence or presence of specific properties needs to be shown. Specifically, what should be done in the case that a property that we are interested in, does not hold in the abstracted system? This question led to the investigation of strongly preserving abstractions of transition systems, which eventually resulted in Chapters 5

and 6, as well as an extension of the abstract interpretation techniques, described in Section 4.8.

7.2 Conclusions

Chapter 3 was initially intended as an introduction to Abstract Interpretation. The approach that we followed was to start from the simple notion of description relation and to motivate a number of conditions that may be imposed on it, step by step. Although a similar approach is taken in several other articles (see Section 3.1 for references), we presented a few new results.

Lemmata 3.2.1.6 and 3.2.1.8 give conditions on the abstraction and concretisation relations, α and γ, which characterise weakenings of the Galois-connection framework. The need for such weaker frameworks has been recognised before (see Section 3.4), and indeed many practical applications of Abstract Interpretation are based on such weakenings. The contribution of Lemmata 3.2.1.6 and 3.2.1.8 is that they identify the relation between α and γ in such cases. It turns out that this relation implies weak versions of monotonicity, reductiveness, and extensiveness — the defining properties of a Galois connection. Thus, we have shown that these weaker frameworks imply a "pre-Galois" connection. These theoretical results are certainly not spectacular. Also, they do not directly play a role in the rest of the thesis. On the other hand, Abstract Interpretation is a relatively young activity that was developed as a unifying theory for a variety of pre-existing applications, and a more fundamental study of the underlying choices seems to be justified (see e.g. [Mar93, CC92a, CC92b]). We think that Section 3.2.1 of this thesis o ers an answer to the question why Galois connections are so intimately related to Abstract Interpretation.

The subsection on the "power construction" (page 44) should also be viewed in this light. The results presented there are theoretically involved (though not "deep"), and are not essential to the rest of this thesis. The investigations were motivated by the shift from the domain C of "elementary" concrete objects to its power set $\mathcal{P}(C)$ that usually takes place. This lifting to the "static" ([CC77]), "collecting" ([CC92b]), or "accumulating" ([JN95]) semantics as it is called in various articles is standard; sometimes it even goes without saying. We have tried to expose the motivations for this shift, while Lemmata 3.2.1.11 and 3.2.1.12 identify conditions under which the resulting framework fits the characterisations in terms of the (pre-)Galois connections discussed before.

Section 3.2.2 on strong preservation indicates how the various choices and assumptions, discussed before, are a ected in the case that we are interested in strong preservation of properties. As such, it is a leg-up to Chapters 5 and 6.

7.2.1 Weak preservation

In Chapter 4, we investigate how the paradigm of Abstract Interpretation may be applied to the analysis of reactive systems. We model such systems by Kripke structures, while the properties that should be verified are expressed in CTL* (although our results remain true if we would have taken the more expressive μ-calculus as our property language instead — see [DGG]). The main result of Sections 4.2 through 4.7 is that weak preservation of full CTL* is possible if we take *mixed transition systems* as descriptions of Kripke structures. Such mixed transition systems accommodate two "dual" transition relations. Universal formulae in CTL* are interpreted over the free transition relation, while existential formulae are evaluated over the constrained relation. An important point is that the appropriateness of such Abstract Kripke structures, i.e. how many CTL* formulae can be verified via the abstraction, may be tuned through the form of the abstract states. In particular, preservation of existential properties is on an equal footing with that of universal properties. This is the consequence of imposing a partially ordered structure on the abstract states. The availability of the "larger" states in this ordering is essential in preserving existential properties — without them, the constrained abstract transition relation tends to be rather sparse, resulting in many existential properties not holding on the Abstract Kripke structure (see Section 4.9.1, page 104).

One may argue that existential properties are less interesting in the practice of verification. Even though this may be true, this should not be a reason for developing a theoretical framework in which there is an asymmetry between the existential and universal fragments. Furthermore, the partial-order structure on the abstract states is useful anyway once it is recognised that it may be desirable to construct *approximations* to the optimal descriptions; see Sections 4.4 and 4.6.

In the abstract interpretation approach, it is the task of the user to provide an appropriate abstract domain. This involves choosing abstract states that have the right "granularity". On the one hand, these descriptions should be detailed enough to induce Abstract Kripke structures over which the properties of interest can successfully be verified. On the other hand, the abstract states should not expose too much detail, to avoid Abstract Kripke structures from blowing up into an intractable size. Together with the set of abstract states, abstract interpretations of the operations in the programming language have to be provided. This allows the automatic construction of Abstract Kripke structure by "running" the program over the abstract values. These choices require intelligence. The choice of an appropriate set of abstract states presupposes a thorough understanding of both the program and the property to be verified. Providing correct abstract operators may turn out to be even more di -

cult[1]. Several attempts may be necessary before the verification can successfully be concluded. On the other hand, it may be argued that the process of choosing an abstract domain, model checking a property, and, if this is unsuccessful, analysing the reasons, is a direct road to understanding the intricacies of both the program and specification.

Nevertheless, the question what to do when verifying the abstraction fails remains a valid one. In particular, approaches with a higher degree of automation seem attractive. It is this question that provoked the investigations reported on in Section 4.8 and Chapters 5 and 6. Thus, starting from Section 4.8, the focus is directed towards methods for strong preservation.

7.2.2 Strong preservation

The approach using abstraction families, sketched in Section 4.8.1, is an extension of the Abstract Interpretation approach. The ideas described there are inspired by similar notions of "tunable" abstractions in the field of logic programming, see for example [Pla84]. A novel aspect is the incremental computation of abstract functions.

In Chapters 5 and 6, a di erent paradigm to construct strongly preserving abstractions is considered: partition refinement algorithms. This approach may be viewed as a process of stepwise refinement of an Abstract Kripke structure. The idea is that in every refinement step, the abstract states are split up, until the structure contains su cient detail to be strongly preserving. If we compare this approach to abstract interpretation, some fundamental di erences can be observed.

No user interaction is required. One could say that, via successive refinements, the appropriate abstract domain is constructed by the algorithm rather than being provided, through trial and error, by the user. Indeed, the transitions are automatically computed too. Second, not only are the successive abstract models automatically refined, but this process is goal directed too: states are being split according to the companion set of the properties of interest (Chapter 5), or to the behavioural equivalence induced by the specification logic (Chapter 6), hence ensuring a conservative approach in which states are only refined when needed. In the abstract interpretation approach the refinement need not necessarily be so goal directed, regardless of whether the user is responsible for choosing an appropriate abstract domain, or the granularity of the abstract domain can be tuned through a parameter as in the abstraction families.

[1] Although the abstractions of arithmetic operators are rather obvious in the examples given in Chapter 4, practical experience shows that correctness proofs may indeed be quite involved once the concrete operations to be abstracted have a more complex nature. This is the case with the unification operator in logic programs for example (see [BCM95]).

A third di erence is that in the partition-refinement approach, the abstract states are always "disjoint"[2]. As a result, the intermediate Abstract Kripke structures that are obtained during the refinement process (which are weakly preserving) may be less suitable for the analysis of existential properties — cf. the remarks in Section 4.9.1. However, as soon as a strongly preserving abstract model is constructed, both truth and falsehood of properties transfer from abstract to concrete model, and hence universal and existential properties are in equal positions.

The last di erence that we mention, concerns the price we are paying for the fact that the process of finding a suitable abstraction can be automated, namely: the partition refinement algorithms need to be able to compute the transition relation of the underlying concrete model. One may ask what the value is of such algorithms in coping with the state explosion, if the concrete model is still needed. One answer is that we envisage applications in combination with symbolic methods, where sets of states and transition relations are represented by BDDs for example. Such symbolic methods may often be combined with approximation techniques, see e.g. [DGD+94, DWT95]. The techniques presented in Chapter 5 may then o er an interesting alternative to *symbolic model checking*: in addition to computing the set of states satisfying the specification, an abstract model is also constructed. Such a model may be useful when other properties need to be verified, possibly serving as a starting point for further refinement. Also, it facilitates the analysis of counter-examples.

Another answer is that there *is* a use for reduction techniques, even when the concrete transition relation is needed as input. In particular, the partition refinement algorithms of Chapter 6, which compute the minimal abstract system that strongly preserves all properties of some given specification logic, are often useful as a preprocessing phase. A subsequent phase may then involve several model checking sessions on this reduced structure; but it can also be that the minimised model is further composed with parallel components. Prior minimisation may then be crucial in keeping the resulting product tractable.

Chapters 5 and 6 also o er some insights that are interesting from a theoretical point of view. The distinction between *logical* and *behavioural* definitions of equivalences gives rise to complementary ways of designing partition refinement algorithms. In the case of an equivalence on states that is defined as agreement on the formulae in (a fragment of) a logic, partition refinement is parameterised by the *companion* of the (fragment of the) logic. This viewpoint is developed in Chapter 5, where the companion is identified as the minimal set of formulae with regard to which the abstract states need to be consistent. Consistency with respect to a formula directly

[2]I.e. for any two abstract states a and b of a model obtained by partition refinement, we have $\gamma(a) \cap \gamma(b) = \emptyset$.

translates to the operational notion of splitting with respect to a formula, hence providing the basis for a partition refinement algorithm. This type of algorithm is novel — in Section 5.8 it is argued that it offers new perspectives in those cases where the equivalence induced by a set of properties does not have a well-behaved behavioural counterpart.

In the case of a behavioural equivalence, partition refinement algorithms exist for bisimulation and stuttering equivalence. Chapter 6 shows how both may be viewed as instances of a generic algorithm that is parameterised by the notion of a *splitter*, which is induced by the form of the definition of the equivalence. It then proceeds by considering two novel variants of CTL*/CTL. One of these induces an equivalence that is coarser than stuttering equivalence — and hence offers possibilities for better model reduction. Yet, the logic's expressive power is still expected to be sufficient for most practical uses. The behavioural equivalence induces a new instantiation of the generic partition refinement algorithm. One could say that, up to optimisations, an algorithm is obtained for free. The equivalence induced by the other logic coincides with stuttering equivalence, surprisingly.

7.3 Looking Ahead

This thesis does not give much experimental evidence for the practical usefulness of the proposed methods — there was simply no more time for a thorough practical evaluation. This may indeed be a weak point of this work. However, we think that it is important to have a firm theoretical basis before starting to build tools. This holds especially when the theory lies at the heart of a quickly expanding research area with a growing number of applications. This certainly is the case as can be seen from the considerable attention that the application of ideas and techniques from Abstract Interpretation to the area of model checking has received lately. Furthermore, we are embarking on some more practical work at present. One line of on-going work has been extensively discussed in Section 4.9.2, and will not be repeated here.

Another, more recent development is the start[3] of SION[4] project nr. 612-33-008, "A Modular Toolset for μCRL developed using μCRL". The goal of this project is the development of a tool set aimed at supporting specification and verification in μCRL ([GP95]). A major concern is keeping the project manageable, now and in the long run. We hope to achieve this by a modular approach, connecting tools that each offer a limited functionality via the ToolBus ([BK95]), and furthermore by

[3] [DG95] reports on a pilot study.

[4] "Stichting Informatica-Onderzoek in Nederland" is the computer science branch of the Dutch national science foundation (NWO).

taking a strict attitude towards specification, verification and documentation. In the context of that project, we envisage the integration of model-checking and abstraction techniques.

Of course, also many questions of a more theoretical nature remain, or are raised by the various chapters. A topic on which on-going research is focussing is the extension of the framework of Chapter 4 to transition systems with *fairness constraints*. One question that we have answered recently is how fairness constraints lift from concrete to abstract systems, in other words, how Büchi (Muller/Streett/Rabin) automata are abstracted. Another possible application of fairness is the generation of constraints during construction of the abstract model: rather than lifting fairness conditions that are already present in the concrete model, such constraints are added to the abstraction so as to constrain its possible behaviours and obtain a more precise description. This research is indeed motivated by the observation that abstraction often introduces "loops" that do not correspond to a similar infinite behaviour in the concrete system. We are currently preparing a publication on the results of these investigations[5]. Another possible extension of Chapter 4 is to consider real-time systems. Verification of such systems (see [ACD93, HNSY94] for entrances) poses a challenge as the discretisation of time adds yet another dimension (besides the possible interleavings and state-space factors that cause a blow-up) to the state explosion. Of a more fundamental nature is the investigation of a framework that enables the comparison and unification of the theories presented in [BBLS92], [Kel95], and Chapter 4.

Also regarding Chapter 6, a possible direction for future work is to extend the results, in particular the definition of flat logics and the corresponding adequacy results, to real-time versions of those logics. The distinguishing power of such logics is very high as a result of the ability to reason about elapsed time. Reduction methods based on the induced equivalences, which are vital in the context of real-time model checking, could be improved by weakening the logics as suggested in this chapter. Indeed, this research was initiated with these motivations in mind. Finally, we mention the research into the distinguishing and expressive powers of the flat variants of CTL and CTL* as a story that is to be continued.

[5]Joint work with R. Gerth and O. Grumberg, partly sponsored by the Netherlands Organisation for International Co-operation in Higher Education (Nu c).

Bibliography

[ABH+92] C.J. Aarts, R.C. Backhouse, P. Hoogendijk, T.S. Voermans, and J. van der Woude. A relational theory of datatypes. Available from `ftp.win.tue.nl/pub/math.prog.construction`, September 1992.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In Cleaveland [Cle92], pages 341–354.

[ACH94] Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. The observational power of clocks. In Jonsson and Parrow [JP94], pages 162–177.

[AH87] S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis Horwood, Chicester, 1987.

[AO91] Krzysztof R. Apt and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs.* Texts and monographs in computer science. Springer-Verlag, New York, 1991.

[ASB+94] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Equivalences for fair Kripke structures. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming*, number 820 in LNCS, pages 364–375. Springer-Verlag, Berlin, 1994.

[ASSSV94] Adnan Aziz, Thomas R. Shiple, Vigyan Singhal, and Alberto L. Sangiovanni-Vincentelli. Formula-dependent equivalence for compositional CTL model checking. In Dill [Dil94], pages 324–337.

[AU77] A.V. Aho and J.D. Ullman. *Principles of Compiler Design*. Addison-Wesley, Amsterdam, 1977.

[BBLS92] S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property preserving simulations. In von Bochmann and Probst [vBP92], pages 251–263.

[BCG88] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Journal of Theoretical Computer Science*, 59:115–131, 1988.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[BCM95] Maurice Bruynooghe, Michael Codish, and Anne Mulkers. Abstract unification: A key step in the design of logic programs. In Jan van Leeuwen, editor, *Computer Science Today*, number 1000 in LNCS, pages 406–425. Springer, Berlin, 1995.

[Ben76] J.F.A.K. van Benthem. *Modal Correspondence Theory*. PhD thesis, Instituut voor Grondslagenonderzoek, University of Amsterdam, 1976.

[Ben94] Johan van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 2 of *Synthese Library. Studies in Epistemology, Logic, Methodology, and Philosophy of Science*, chapter II.4, pages 167–247. Kluwer Academic Publishers, Dordrecht, 1994.

[BFG+91] A. Bouajjani, J.C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching time semantics. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Automata, Languages and Programming*, number 510 in LNCS, pages 76–92. Springer-Verlag, New York, 1991.

[BFH90] A. Bouajjani, J. C. Fernandez, and N. Halbwachs. On the verification of safety properties. Technical Report SPECTRE L12, Laboratoire d'Informatique et de Mathématiques Appliqées de Grenoble, March 1990.

[BFH+92] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18:247–269, 1992.

[BG87] J. C. M. Baeten and R. J. van Glabbeek. Another look at abstraction in process algebra (extended abstract). In Thomas Ottmann, editor, *Automata, Languages and Programming*, number 267 in LNCS, pages 84–94. Springer-Verlag, Berlin, 1987.

[BG93] Orna Bernholtz and Orna Grumberg. Branching time temporal logic and amorphous tree automata. In Eike Best, editor, *CONCUR '93*, number 715 in LNCS, pages 262–277. Springer-Verlag, Berlin, 1993.

[BHA86] G. Burn, C. Hankin, and S. Abramsky. Strictness analysis for higher order functions. *Science of Computer Programming*, 7:249–278, 1986.

[Bir67] Garrett Birkho . *Lattice Theory*, volume 25 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, third edition, 1967.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Journal of Theoretical Computer Science*, 37(1):77–121, 1985.

[BK90] J.C.M. Baeten and J.W. Klop, editors. *CONCUR '90. Theories of Concurrency: Unification and Extension*, number 458 in LNCS. Springer-Verlag, Berlin, 1990.

[BK95] J.A. Bergstra and P. Klint. The disrete time toolbus. Technical Report P9502, Dept. of Math. and Comp. Sc., University of Amsterdam, March 1995.

[BKS83] R. J. R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *2nd ACM SIGACT–SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142. ACM, New York, 1983.

[Bou92] François Bourdoncle. Abstract interpretation by dynamic partitioning. *Journal of Functional Programming*, 2(4):407–435, 1992.

[Bou50] Nicolas Bourbaki. Sur le théorème de Zorn. *Archiv Der Mathematik*, 2:435–437, 1949/50.

[BR83] Stephen D. Brookes and William C. Rounds. Behavioural equivalence relations induced by programming logics. In J. Diaz, editor, *Automata, Languages and Programming*, number 154 in LNCS, pages 97–108. Springer-Verlag, Berlin, 1983.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[Bur92] Jerry R. Burch. *Trace Algebra for Automatic Verification of Real-Time Concurrent Systems*. PhD thesis, School of Comp. Sc., Carnegie Mellon University, Pittsburgh, PA 15213, August 1992.

[BVW94] Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking (extended abstract). In Dill [Dil94], pages 142–155.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[CBM89] Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Verification of synchronous sequential machines based on symbolic execution. In Sifakis [Sif89], pages 365–373.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979.

[CC82] A. Cardon and M. Crochemore. Partitioning a graph in $O(|A| \log_2 |V|)$. *Journal of Theoretical Computer Science*, 19:85–98, 1982.

[CC84] Patrick Cousot and Radhia Cousot. Invariance proof methods and analysis techniques for parallel programs. In Alan W. Biermann, Gérard Guiho, and Yves Kodrato , editors, *Automatic Program Construction Techniques*, chapter 12, pages 243–271. Macmillan Publishing Company, New York, 1984.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13:103–179, 1992.

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[CC92c] Patrick Cousot and Radhia Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Programming Language Implementation and Logic Programming*, number 631 in LNCS, pages 269–295. Springer-Verlag, Berlin, 1992.

[CC94] Patrick Cousot and Radhia Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In *1994 International Conference on Computer Languages*, pages 95–112, Los Alamitos, CA. IEEE Computer Society Press, 1994.

[CC95] U. Celikkan and R. Cleaveland. Generating diagnostic information for behavioral preorders. *Distributed Computing*, 9:61–75, 1995.

[CDY91] M. Codish, D. Dams, and E. Yardeni. Derivation and safety of an abstract unification algorithm for groundness and sharing analysis. In K. Furukawa, editor, *Proceedings of the 8th International Conference of Logic Programming and Symposium of Logic Programming*, pages 79–93. The MIT Press, Cambridge, Mass., 1991.

[CE81] E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In Dexter Kozen, editor, *Logics of Programs: Workshop*, number 131 in LNCS. Springer-Verlag, New York, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGH+93] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ cache coherence protocol. In L. Claesen, editor, *Proceedings of the Eleventh International Symposium on Computer Hardware Description Languages and their Applications*. North-Holland, 1993.

[CGL92] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. In *Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 1992.

[CGL93] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency: Reflections and Perspectives*, number 803 in LNCS, pages 124–175. Springer-Verlag, Berlin, 1993.

[CGL94] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[Cho95] Ching-Tsun Chou. A simple treatment of property preservation via simulation. Technical Report 950014, Comp. Sc. Dept., University of California at Los Angeles, March 1995.

[CIY94] R. Cleaveland, S. P. Iyer, and D. Yankelevich. Abstractions for preserving all CTL* formulae. Technical Report 94-03, Dept. of Comp. Sc., North Carolina State University, Raleigh, NC 27695, April 1994.

[CIY95] Rance Cleaveland, Purush Iyer, and Daniel Yankelevich. Optimality in abstractions of model checking. In Alan Mycroft, editor, *Static Analysis*, number 983 in LNCS, pages 51–63. Springer-Verlag, New York, 1995.

[CK90] E.M. Clarke and R.P. Kurshan, editors. *Computer-Aided Verification*, number 531 in LNCS. Springer-Verlag, New York, 1990.

[Cle92] W.R. Cleaveland, editor. *CONCUR '92*, number 630 in LNCS. Springer-Verlag, Berlin, 1992.

[Cou93] Costas Courcoubetis, editor. *Computer Aided Verification*, number 697 in LNCS. Springer-Verlag, Berlin, 1993.

[CR94] R. Cleaveland and J. Riely. Testing-based abstractions for value-passing systems. In Jonsson and Parrow [JP94], pages 417–432.

[Dam94] Mads Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Journal of Theoretical Computer Science*, 126:77–96, 1994.

[DFFGI95] N. De Francesco, A. Fantechi, S. Gnesi, and P. Inverardi. Model checking of non-finite state processes by finite approximations. In E. Brinksma, W. R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 1019 in LNCS, pages 195–215. Springer, Berlin, 1995.

[DG95] D. Dams and J.F. Groote. Specification and implementation of components of a μCRL toolbox. Logic Group Preprint Series 152, Dept. of Philosophy, Utrecht University, The Netherlands, December 1995.

[DGD+94] D. Dams, R. Gerth, G. Döhmen, R. Herrmann, P. Kelb, and H. Pargmann. Model checking using adaptive state and data abstraction. In Dill [Dil94], pages 455–467.

[DGG] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems. Submitted to ACM Transactions on Programming Languages and Systems.

[DGG93a] D. Dams, R. Gerth, and O. Grumberg. Generation of reduced models for checking fragments of CTL. In Courcoubetis [Cou93], pages 479–490.

[DGG93b] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. Draft, July 1993.

[DGG94] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ∀CTL*, ∃CTL* and CTL*. In E.-R. Olderog, editor, *Proceedings of the IFIP WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, IFIP Transactions. North-Holland/Elsevier, Amsterdam, 1994.

[DHKS95] Werner Damm, Hardi Hungar, Peter Kelb, and Rainer Schlör. Statecharts: Using graphical specification languages and symbolic model checking in the verification of a production cell. In Lewerenz and Lindner [LL95], pages 131–149.

[Dil89] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. The MIT Press, London, 1989.

[Dil94] David L. Dill, editor. *Computer Aided Verification*, number 818 in LNCS. Springer-Verlag, Berlin, 1994.

[DJS95] Werner Damm, Bernhard Josko, and Rainer Schlör. Specification and verification of VHDL-based system-level hardware designs. In Egon Börger, editor, *Specification and Validation Methods*, International

Schools for Computer Scientists, pages 331–409. Oxford University Press, Oxford, 1995.

[DN87] Rocco De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[DNV90a] Rocco De Nicola and Frits Vaandrager. Action versus state based logics for transition systems. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, number 469 in LNCS, pages 407–419. Springer-Verlag, New York, 1990.

[DNV90b] Rocco De Nicola and Frits Vaandrager. Three logics for branching bisimulation. In *1990 IEEE Fifth Annual Symposium on Logic in Computer Science*, pages 118–129. IEEE Computer Society Press, Los Alamitos, CA, 1990. Full version available as Report CS-R9012, Centre for Math. and Comp. Sc., Amsterdam.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, 1990.

[DWT95] David L. Dill and Howard Wong-Toi. Verification of real-time systems by successive over and under approximation. In Wolper [Wol95], pages 409–422.

[EC82] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the Association for Computing Machinery*, 33(1):151–178, 1986.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem of formalised theories. *Fundamenta Mathematicae*, 49:129–141, 1961.

[EL87] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[Eme90] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Formal Models and Semantic*, volume B of *Handbook of Theoretical Computer Science*, chapter 16, pages 995–1072. Elsevier/The MIT Press, Amsterdam, 1990.

[FKM93] J.-C. Fernandez, A. Kerbrat, and L. Mounier. Symbolic equivalence checking. In Courcoubetis [Cou93], pages 85–96.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[FM91] Jean-Claude Fernandez and Laurent Mounier. "On the fly" verification of behavioural equivalences and preorders. In Larsen and Skou [LS91], pages 181–191.

[Fra54] R. Fraïssé. Sur quelques classifications des systèmes de relations. Publ. Sci. Univ. Alger. I, volume 1, pages 35–182. 1954.

[Fra92] Nissim Francez. *Program Verification*. International Computer Science Series. Addison-Wesley, Wokingham, England, 1992.

[Gin68] A. Ginzburg. *Algebraic Theory of Automata*. ACM Monograph Series. Academic Press, New York/London, 1968.

[GKP92] Ursula Goltz, Ruurd Kuiper, and Wojciech Penczek. Propositional temporal logics and equivalences. In Cleaveland [Cle92], pages 222–236.

[GKPP95] Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek. A partial order approach to branching time logic model checking. In *Third Israel Symposium on the Theory of Computing and Systems*, pages 130–139. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[GL93] Susanne Graf and Claire Loiseaux. A tool for symbolic program verification and abstraction. In Courcoubetis [Cou93], pages 71–84.

[God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*. Number 1032 in LNCS. Springer, Berlin, 1996.

[Gol92] Robert Goldblatt. *Logics of Time and Computation*. Number 7 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, second edition, 1992.

[GP95] Jan Friso Groote and Alban Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes, Utrecht 1994*, Workshops in Computing, pages 26–62. Springer-Verlag, New York, 1995.

[Gra94] Susanne Graf. Verification of a distributed cache memory by using abstractions. In Dill [Dil94], pages 207–219. To appear in *Distributed Computing*.

[GS84] S. Graf and J. Sifakis. A modal characterization of observational congruence on finite terms of CCS. In Jan Paredaens, editor, *Proc. of the Eleventh International Colloquium on Automata Languages and Programming (ICALP)*, number 172 in LNCS, pages 222–234. Springer-Verlag, Berlin, 1984.

[GV90] Jan Friso Groote and Frits Vaandrager. An e cient algorithm for branching bisimulation and stuttering equivalence. In M. S. Paterson, editor, *Automata, Languages and Programming*, number 443 in LNCS, pages 626–638. Springer-Verlag, New York, 1990.

[GW89] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G. X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, Amsterdam, 1989. Full version available as Report CS-R9120, Center for Math. and Comp. Sc., Amsterdam, 1991.

[GW91] P. Godefroid and P. Wolper. Using partial orders for the e cient verification of deadlock freedom and safety properties. In Larsen and Skou [LS91], pages 332–342.

[Har80] David Harel. On folk theorems. *Communications of the ACM*, 23(7):379–389, 1980.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HC84] G.E. Hughes and M.J. Cresswell. *A Companion to Modal Logic*. Methuen, London, 1884.

[HHK95] Monika R. Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science*, pages 453–462. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[HK94] J. Helbig and P. Kelb. An OBDD-representation of StateCharts. In *Proceedings. The European Design and Test Conference*, pages 142–149. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J.W. de Bakker and J. van Leeuwen, editors, *Proc. of the Seventh International Colloquium on Automata Languages and Programming (ICALP)*, number 85 in LNCS, pages 299–309. Springer-Verlag, Berlin, 1980.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(1):193–244, 1994.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cli s, NJ, 1985.

[Hol84] Gerard J. Holzmann. Backward symbolic execution of protocols. In Yechiam Yemini, Robert Strom, and Yemini Shaula, editors, *Protocol Specification, Testing, and Verification*, pages 19–30. North-Holland, Amsterdam, 1984.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall Software Series. Prentice-Hall International, London, 1991.

[Hol96] Marco Hollenberg, May 1996. Private communication.

[Hop71] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, New York, 1971.

[IK87] Neil Immerman and Dexter Kozen. Definability with bounded number of bound variables. In *Logic in Computer Science*, pages 236–244. IEEE Computer Society Press, Washington DC, 1987.

[JJ89] Claude Jard and Thierry Jeron. On-line model-checking for finite linear temporal logic specifications. In Sifakis [Sif89], pages 189–196.

[JK90] Ryszard Janicki and Maciej Koutny. Using optimal simulations to reduce reachability graphs. In Clarke and Kurshan [CK90], pages 166–175.

[JN95] Neil D. Jones and Flemming Nielson. Abstract interpretation: a semantics-based tool for program analysis. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science. Semantic Modelling*, volume 4, pages 527–636. Oxford University Press, Oxford, 1995.

[Jos90] Bernhard Josko. A context dependent equivalence relation between Kripke structures. In Clarke and Kurshan [CK90], pages 204–213. An extended version appeared in B. Josko, Modular Specification and Verification of Reactive Systems, Habilitationsschrift, Carl von Ossietzky University of Oldenburg, Germany, 1993.

[JP94] B. Jonsson and J. Parrow, editors. *CONCUR '94: Concurrency Theory*, number 836 in LNCS. Springer-Verlag, Berlin, 1994.

[KDG95] Peter Kelb, Dennis Dams, and Rob Gerth. E cient symbolic model checking of the full μ-calculus using compositional abstractions. Computing Science Reports 95/31, Eindhoven University of Technology, The Netherlands, October 1995.

[Kel95] Peter Kelb. *Abstraktionstechniken für automatische Verifikationsmethoden*. PhD thesis, Carl von Ossietzky University of Oldenburg, Germany, December 1995.

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Journal of Theoretical Computer Science*, 27:333–354, 1983.

[KP92] Shmuel Katz and Doron Peled. Verification of distributed programs using representative interleaving sequences. *Distributed Computing*, 6:107–120, 1992.

[Kri63] S. Kripke. A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[KS90] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[Kup95] Orna Kupferman. *Model Checking for Branching-Time Temporal Logics*. PhD thesis, Technion — Israel Institute of Technology, Haifa, Israel, June 1995.

[Kur90] R. P. Kurshan. Analysis of discrete event coordination. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, number 430 in LNCS, pages 414–453. Springer-Verlag, Berlin, 1990.

[Kur94] Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton Series in Computer Science. Princeton University Press, Princeton, NJ, 1994.

[Lam83] L. Lamport. What good is temporal logic? In R.E.A. Mason, editor, *Information Processing 83*, pages 657–668. IFIP, Elsevier Science Publishers B.V. (North-Holland), 1983.

[Lar89] Kim Guldstrand Larsen. Modal specifications. In Sifakis [Sif89], pages 232–246.

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:11–44, 1995.

[LIC86] *Logic in Computer Science*. IEEE Computer Society Press, 1986.

[LL95] Claus Lewerenz and Thomas Lindner, editors. *Formal Development of Reactive Systems: Case Study Production Cell*. Number 891 in LNCS. Springer-Verlag, Berlin, 1995.

[LNS82] J.-L. Lassez, V. L. Nguyen, and E. A. Sonenberg. Fixed point theorems and semantics: A folk tale. *Information Processing Letters*, 14(3):112–116, 1982.

[Loi94] Claire Loiseaux. *Vérification symbolique de systèmes réactifs à l'aide d'abstractions*. PhD thesis, Université Joseph Fourier – Grenoble I, Grenoble, France, February 1994.

[Lon93] David E. Long. *Model Checking, Abstraction, and Compositional Verification*. PhD thesis, School of Comp. Sc., Carnegie Mellon University, Pittsburgh, PA 15213, July 1993.

[LP85] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107. 1985.

[LS91] K.G. Larsen and A. Skou, editors. *Computer Aided Verification*, number 575 in LNCS. Springer-Verlag, Berlin, 1991.

[LT88] Kim G. Larsen and Bent Thomsen. A modal process logic. In *1988 IEEE Symposium on Logic in Computer Science*, pages 203–210. Computer Society Press, Washington, 1988.

[LV92] B. Le Charlier and P. Van Hentenryck. On the design of generic abstract interpretation frameworks. In M. Billaud, P. Castéran, M.-M. Corsini, K. Musumbu, and A. Rauzy, editors, *Actes WSA'92 Workshop on Static Analysis (Bordeaux)*, volume 81–82 of *Bigre*, Laboratoire Bordelais de Recherche en Informatique, September 1992.

[LY92] D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proc. ACM Symp. on Theory of Computing*, pages 264–274, 1992.

[Mal73] A.I. Mal'cev. *Algebraic Systems*. Number 192 in Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen. Springer-Verlag, Berlin, 1973.

[Mar93] K. Marriott. Frameworks for abstract interpretation. *Acta Informatica*, 30(2):103–129, 1993.

[Mat94] Mathematics of Program Construction Group. Fixed-point calculus. Technical Report 94–48, Eindhoven University of Technology, Dept. of Computing Science, October 1994.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[Mer90] N. Mercouro . *Analyse sémantique des communications entre processus de programmes parallèles*. PhD thesis, Ecole Polytechnique, Laboratoire d'Informatique, Palaiseau, France, September 1990.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Second International Joint Conference on Artificial Intelligence*, pages 481–489. British Computer Society, London, 1971.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Number 92 in LNCS. Springer-Verlag, Berlin, 1980.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Journal of Theoretical Computer Science*, 25:267–310, 1983.

[MJ81] Steven S. Muchnick and Neil D. Jones, editors. *Program Flow Analysis: Theory and Applications*. Prentice-Hall, Englewood Cli s, NJ, 1981.

[MJ86] A. Mycroft and N. Jones. A relational framework for abstract interpretation. In *Programs as data objects*, number 154 in LNCS, pages 536–547. Springer, 1986.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.

[MP95] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Verification*. Springer-Verlag, New York, 1995.

[MS89a] K. Marriott and H. Søndergaard. Semantics-based dataflow analysis of logic programs. In G. X. Ritter, editor, *Information Processing 89*, pages 601–606. North-Holland, Amsterdam, 1989.

[MS89b] K. Marriott and H. Søndergaard. A tutorial on abstract interpretation of logic programs. Notes distributed at the North American Conference on Logic Programming, 1989.

[MS92a] K. Marriott and H. Søndergaard. Bottom-up dataflow analysis of normal logic programs. *Journal of Logic Programming*, 13:181–204, 1992.

[MS92b] K. L. McMillan and J. Schwalbe. Formal verification of the Gigamax cache consistency protocol. In N. Suzuki, editor, *Shared Memory Multiprocessing*. The MIT Press, 1992.

[MSS86] A.C. Melton, D.A. Schmidt, and D.E. Strecker. Galois connections and computer science applications. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Category Theory and Computer Programming*, number 240 in LNCS, pages 299–312. Springer-Verlag, Berlin, 1986.

[Nau63] P. Naur. The design of the Gier Algol compiler, part II. *BIT*, 3:145–166, 1963.

[NC94] Mogens Nielsen and Christian Clausen. Bisimulation, games, and logic. In J. Karhumäki, H. Maurer, and G. Rozenberg, editors, *Results and Trends in Theoretical Computer Science*, number 812 in LNCS, pages 289–306. Springer-Verlag, Berlin, 1994.

[Nie82] F. Nielson. A denotational framework for data flow analysis. *Acta Informatica*, 18:265–287, 1982.

[Nie88] F. Nielson. Strictness analysis and denotational abstract interpretation. *Information and Computation*, 76(1):29–92, 1988.

[Ore44] O. Ore. Galois connexions. *Transactions of the American Mathematical Society*, 55:493–513, 1944.

[Par81] D. Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, number 104 in LNCS, pages 167–183. Springer-Verlag, Berlin, 1981.

[Pic52] G. Pickert. Bemerkungen über Galois-Verbindungen. *Archiv der Mathematik*, 3:285–289, 1952.

[PL90] David K. Probst and Hon F. Li. Using partial-order semantics to avoid the state explosion problem in asynchronous systems. In Clarke and Kurshan [CK90], pages 146–155.

[Pla84] David A. Plaisted. The occur-check problem in Prolog. In *1984 International Symposium On Logic Programming*, pages 272–280. IEEE Computer Society Press, Los Alamitos, CA, 1984.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. 1977.

[Pnu86] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proc. of the Twelfth International Colloquium on Automata Languages and Programming (ICALP)*, number 194 in LNCS, pages 15–32. Springer-Verlag, New York, 1986.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computation*, 16(6):973–989, 1987.

[PZ86] Amir Pnueli and Lenore Zuck. Probabilistic verification by tableaux. In LICS86 [LIC86], pages 322–331.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *International Symposium on Programming*, number 137 in LNCS, pages 337–351. Springer-Verlag, Berlin, 1982.

[Sch86] D. A. Schmidt. *Denotational Semantics: A Methodology for language Development.* Allyn & Bacon, Newton, MA, 1986.

[Sco82] Dana S. Scott. Domains for denotational semantics. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and Programming*, number 140 in LNCS, pages 577–613. Springer-Verlag, Berlin, 1982.

[Sif82] J. Sifakis. Property preserving homomorphisms and a notion of simulation for transition systems. Rapport de Recherche 332, Laboratoire d'Informatique et de Mathématiques Appliqées de Grenoble, November 1982.

[Sif83] J. Sifakis. Property preserving homomorphisms of transition systems. In Edmund Clarke and Dexter Kozen, editors, *4th Workshop on Logics of Programs*, number 164 in LNCS, pages 458–473. Springer-Verlag, Berlin, 1983.

[Sif89] J. Sifakis, editor. *Automatic Verification Methods for Finite State Systems (CAV89)*, number 407 in LNCS. Springer-Verlag, Berlin, 1989.

[Sin72] M. Sintzo . Calculating properties of programs by valuation on specific models. *SIGPLAN Notices*, 7(1):203–207, 1972. (Proc. ACM Conf. Proving Assertions about Programs.)

[Søn90] H. Søndergaard. Semantic based analysis and transformation of logic programs. Technical Report 12, The University of Melbourne, June 1990. Revised version of PhD thesis submitted to the University of Copenhagen, December, 1989.

[Sti89] Colin Stirling. Comparing linear and branching time temporal logics. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, number 398 in LNCS, pages 1–20. Springer-Verlag, Berlin, 1989.

[Sti92] C. Stirling. Modal and temporal logics. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science. Background: Computational Structures*, volume 2, pages 477–563. Oxford University Press, Oxford, 1992.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press, Cambridge, Mass., 1977.

[Tar55] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5:285–309, 1955.

[Tho93] Wolfgang Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In M. *et al.* Gaudel, editor, *TAPSOFT*, number 668 in LNCS, pages 559–568. Springer-Verlag, New York, 1993.

[Val90] Antti Valmari. A stubborn attack on state explosion. In Clarke and Kurshan [CK90], pages 156–165.

[vBP92] G. von Bochmann and D.K. Probst, editors. *Computer-Aided Verification*, number 663 in LNCS. Springer-Verlag, New York, 1992.

[vBvES94] Johan van Benthem, Jan van Eijck, and Vera Stebletsova. Modal logic, transition systems and processes. *Journal of Logic and Computation*, 4(5):811–855, 1994.

[vG90a] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, Free University of Amsterdam/Center for Math. and Comp. Sc., 1990.

[vG90b] R.J. van Glabbeek. The linear time – branching time spectrum. In Baeten and Klop [BK90], pages 278–297.

[vG93a] R.J. van Glabbeek. The linear time – branching time spectrum II. the semantics of sequential systems with silent moves. Preliminary version available from `boole.stanford.edu`, 1993. Extended abstract in *Proc. CONCUR 93*, LNCS 715, pp. 66–81. Springer-Verlag, Berlin, 1993.

[vG93b] R.J. van Glabbeek. What is branching time semantics and why to use it? Technical Report STAN-CS-93-1486, Stanford University, Dept. of Comp. Sc., CA 94305, 1993. Also appeared in The Concurrency Column (M. Nielsen, ed.), Bulletin of the EATCS 53, June 1994, pp. 190–198.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In LICS86 [LIC86], pages 332–344.

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–193. 1986.

[Wol95] Pierre Wolper, editor. *Computer Aided Verification*, number 939 in LNCS. Springer-Verlag, Berlin, 1995.

[YL93] Mihalis Yannakakis and David Lee. An e cient algorithm for minimizing real-time transition systems. In Courcoubetis [Cou93], pages 210–224.

Samenvatting

De toenemende mate waarin de mens afhankelijk is van apparaten leidt tot verhoogde eisen aan de betrouwbaarheid van programmatuur. De reden hiervoor is dat steeds meer toestellen en processen bestuurd worden door geintegreerde processoren en computers — denk bijvoorbeeld aan liften, automatische bagage-afhandelingssystemen en kernreactoren. Deze ontwikkeling vraagt om specificatie- en verificatiemethoden. Om de correctheid van programmatuur met uiterste nauwkeurigheid te kunnen vaststellen, zijn wiskundige ofwel "formele" methoden een waardevol hulpmiddel. Zo een methode dient het volgende te bieden:

- Een wiskundig model waarmee het gedrag van een computerprogramma kan worden beschreven.
- Een formele taal waarin de specificatie kan worden uitgedrukt.
- Een methode om te verifiëren of het model aan de specificatie voldoet.

Model checking is zo een formele methode. Hierin wordt een programma gemodelleerd als een transitiegraaf, die alle mogelijke toestanden weergeeft waarin het programma zich kan bevinden, alsook de toestandsovergangen die kunnen optreden. De specificatie wordt uitgedrukt als een formule in een temporele logica. Dit is een formele taal waarin de volgorde en de noodzakelijkheid van proposities of acties kunnen worden beschreven. Bijvoorbeeld, men kan er eigenschappen in uitdrukken als: "wanneer de lift wordt opgeroepen, dan zal hij uiteindelijk komen". De verificatie gebeurt nu door na te gaan of de formule geldig is in de transitiegraaf. Voor diverse klasses van modellen en temporele logica's bestaan algoritmes waarmee dit automatisch gedaan kan worden. Echter, de afmetingen van de transitiegraaf groeien zeer snel bij toenemende omvang van het programma. Elke extra bit geheugen leidt potentieel tot een verdubbelde toestandsruimte. Indien een programma bestaat uit afzonderlijke componenten die parallel uitgevoerd worden, dan kan ook het aantal mogelijke aaneenschakelingen ("interleavings") van acties van de afzonderlijke componenten zeer groot zijn. Door deze "toestandsexplosie" is model checking alleen

haalbaar voor kleine programma's. In de afgelopen jaren is deze grens voortdurend opgeschoven door toepassing van verschillende technieken om de afmetingen van de transitiegraaf te reduceren.

In dit proefschrift wordt een reductietechniek bestudeerd die uitgaat van de idee dat, afhankelijk van de te checken eigenschappen, sommige aspecten van toestanden genegeerd kunnen worden. Alle toestanden die slechts in die aspecten verschillen, kunnen dan geïdentificeerd worden. Een manier om dit te formaliseren is een verzameling *abstracte toestanden* te veronderstellen zodanig dat iedere abstracte toestand een verzameling (concrete) toestanden representeert. De volgende vragen doen zich hierbij voor:

- Wat is de relatie tussen de te checken eigenschappen en de aspecten die genegeerd kunnen worden?
- Hoe is een transitiegraaf op basis van de abstracte toestanden te definiëren zodanig dat eigenschappen die in dit abstracte model gelden, ook correct zijn met betrekking tot de concrete graaf?
- Hoe kan, gegeven een programma en een verzameling te checken eigenschappen, een abstract model geconstrueerd worden?

Na een algemene inleiding in Hoofdstuk 1 en een verklaring van de te gebruiken formalismen en notaties in Hoofdstuk 2, biedt Hoofdstuk 3 een algemene inleiding tot *abstractietheorieën*. Een abstractietheorie formaliseert de relatie tussen concrete en abstracte objecten en geeft preservatieresultaten, d.w.z. specificeert welke eigenschappen van abstracte objecten ook voor de concrete objecten gelden. Er wordt een onderscheid gemaakt tussen zwakke en sterke preservatie. In het geval van zwakke preservatie is elke eigenschap die een abstract object a heeft, ook geldig voor alle concrete objecten die door a geabstraheerd worden. Bij sterke preservatie is ook het omgekeerde het geval. Het moge duidelijk zijn dat indien de logica waarin eigenschappen geformuleerd worden, een klassieke notie van negatie heeft, sterke preservatie geimpliceerd wordt door zwakke preservatie. Dit onderscheid speelt een hoofdrol in de rest van het proefschrift.

Naast preservatieresultaten biedt een abstractietheorie methoden om abstracte objecten te construeren. In het bijzonder besteedt Hoofdstuk 3 aandacht aan *Abstracte Interpretatie*. Deze theorie is ontwikkeld als formalisering en unificatie van verschillende programma-*analyses*. Deze leiden informatie over het gedrag van een programma af met als doel het te optimaliseren, bijvoorbeeld door het opsporen van bepaalde soorten fouten of het detecteren van stukken code die parallel uitgevoerd

kunnen worden. Doordat zulke analyses in het algemeen op zwakke preservatie gebaseerd zijn, zijn ze niet volledig. Bijvoorbeeld, hoewel gedetecteerde fouten in het abstracte model erop wijzen dat er daadwerkelijk fouten in het programma staan, geeft de afwezigheid van fouten in de abstractie geen garantie voor correctheid van het programma. Men zou kunnen zeggen dat vragen naar de geldigheid van eigenschappen met "ja" of "weet niet" beantwoord worden door de abstractie.

De constructie van zo een abstractie wordt geformaliseerd als de uitvoering van het programma over een "niet-standaard" (abstract) domein van *beschrijvingen* van concrete data, waarbij de programma-operatoren "abstract geïnterpreteerd" worden. Een eenvoudig voorbeeld is het interpreteren van het vermenigvuldigingssymbool over de beschrijvingen "positief" en "negatief" van getallen: -1515×17 wordt aldus geïnterpreteerd als "negatief maal positief". Het abstracte resultaat, "negatief", geeft slechts partiële informatie over het werkelijke (concrete) resultaat — daartegenover staat dat het op gemakkelijker wijze verkregen wordt.

In Hoofdstuk 4 wordt Abstracte Interpretatie opgezet voor het geval dat programma's gemodelleerd worden door transitiegrafen (in het bijzonder *Kripke structuren*) en eigenschappen uitgedrukt worden in de temporele logica CTL* ("*Computation Tree Logic*"). In deze logica kan zowel het bestaan van een gedrag dat aan zekere eigenschappen voldoet (*existentiële eigenschap*), worden uitgedrukt, alsook het feit dat alle gedragingen van een programma aan een bepaalde eigenschap voldoen (*universele eigenschap*). Een notie van Abstracte Kripke structuur waarin twee transitierelaties verenigd zijn, wordt gedefinieerd. Eén van de hoofdresultaten van het hoofdstuk is dat CTL* zwak gepreserveerd wordt voor zulke Abstracte Kripke structuren. Hoewel de negatie van eigenschappen kan worden gespecificeerd in CTL*, hoeft zwakke preservatie toch geen sterke te impliceren, omdat de universele en existentiële eigenschappen geïnterpreteerd worden over verschillende transitierelaties van het abstracte model. Het kan dus het geval zijn dat noch eigenschap φ, noch de negatie $\neg\varphi$ geldt. Verder wordt getoond hoe Abstracte Kripke structuren geconstrueerd kunnen worden door abstracte interpretatie van programma's.

De geschiktheid van het abstracte model, d.w.z. hoeveel CTL*-eigenschappen er in gelden, wordt bepaald door de keuze van het abstracte domein (inclusief de interpretaties van programma-operatoren daarover). Deze keus is aan de gebruiker van de methode. De vraag die voor de hand ligt, is hoe een geschikt abstract domein bepaald kan worden, gegeven een programma en een verzameling te checken eigenschappen. De aanzet tot een antwoord wordt gegeven in Sectie 4.8, waar de notie van *abstractiefamilie* wordt gedefinieerd. Deze kan worden gezien als een abstract domein waarvan de "fijnkorreligheid" kan worden aangepast via een parameter.

Hoofdstuk 5 en 6 benaderen de vraag naar de vorm van een geschikt abstract domein vanuit een meer theoretisch oogpunt, door uit te gaan van de eis van sterke preservatie van CTL*-eigenschappen. Waar de *analyse* van programma's op incomplete methoden kan berusten en dus voldoende heeft aan zwak preserverende abstracties, vereist *verificatie* een ja/nee-antwoord en dus sterke preservatie. In Hoofdstuk 5 worden, uitgaande van een verzameling sterk te preserveren CTL*-eigenschappen, voldoende voorwaarden afgeleid waaronder een abstract model sterk preserverend is met betrekking tot deze eigenschappen. Uit deze voorwaarden wordt een algoritme afgeleid dat herhaald abstracte toestanden opsplitst, totdat een sterk preserverend model is verkregen. Voor verschillende deelverzamelingen van CTL* worden dergelijke *partitie-verfijningsalgoritmes* ontwikkeld. De idee om zulke algoritmes te baseren op een verzameling te preserveren formules is nieuw — we spreken in dit geval van *logische* partitie-verfijning.

Er bestaan reeds partitie-verfijningsalgoritmes die de toestanden van een abstracte transitiegraaf zodanig opdelen, dat uiteindelijk elke abstracte toestand met een equivalentieklasse van een zogenaamde "gedrags-equivalentie" ("*behavioural equivalence*", bijvoorbeeld *bisimulatie* of *stotter-equivalentie*) correspondeert. Ook deze *gedrags*-partitie-verfijningsalgoritmes kunnen worden gebruikt om abstracte modellen te construeren die sterk preserverend zijn, omdat bij elke van deze gedrags-equivalentie een sublogica van CTL* "past". Hoofdstuk 6 laat zien hoe deze bestaande algoritmes gezien kunnen worden als invulling van een generiek algoritme, door het te parametriseren met de notie van een *splitser*. Deze wordt bepaald door de vorm van de definitie van de corresponderende equivalentie. Er wordt een nieuwe sublogica gedefinieerd die correpondeert met een notie van equivalentie die zwakker is dan bisimulatie of stotter-equivalentie — en dus tot betere reductie leidt —, maar nog redelijk expressief is. Het bijbehorende partitie-verfijningsalgoritme is weer een invulling van het generieke algoritme.

Index

$\models$, **22**
$\preceq$, 36, 60, 77
$\sqsubseteq$, 38
$\equiv^0$, **22**, 161
$\equiv_{\text{bis}}$, **27**, **149**
$\equiv_{\text{dbs}}$, **156**
$\equiv_{\text{flat}^-}$, **169**
$\equiv_{\text{flat}^*}$, **174**
$\equiv_{\text{stut}}$, **157**
$-\equiv\!\!\rightarrow$, **155**
$\|\cdot\|_{\text{Lit}}$, **21**
${}_\alpha\|\cdot\|_{\text{Lit}}$, **61**
${}_\alpha\text{R}^C$, **63**
${}_\alpha\text{R}^F$, **65**
${}_\alpha\mathcal{I}$, 67
${}_\alpha\text{I}$, **61**
α^M, **66**
ρ, 31
$\forall$CTL, **20**, 142
$\forall\text{CTL}^-$, **129**
$\forall\text{CTL}^*$, **19**, 101
$\mathcal{C}$, **59**
CTL, 18, **20**, 111
CTL*, 7, 9, 18, **19**, 58, 89–91, 101
$\exists$CTL, **20**
$\exists\text{CTL}^*$, **19**
bool, **168**
comp, **125**, 136
flat$^-$CTL(U), **170**
flatCTL*(U), **168**
$\mathcal{I}$, 67
$\mathit{infstut}_{\equiv}$, **157**
init, **127**
intp, **127**
£, **22**
Lit, **18**
length, **21**
L_μ, 18, 91, 101, 111
lfp, **16**
μ-calculus, 9, 18, 91, 101, 109, 143, 153
Pred, **120**
Prop, **18**
$\mathit{partit}_{\equiv}$, **155**
paths, **21**, 150, 157
post_R, $\widetilde{\mathit{post}}_R$, $\mathit{post}^\bullet_R$, **14**
pre_R, $\widetilde{\mathit{pre}}_R$, $\mathit{pre}^\bullet_R$, **14**
rel, **127**
sim, **26**
sim^n, **26**, 133
simeq, **27**
simeq^n, **27**, 133
split, **126**
subform, **136**

abstract, abstractness, 147, 150
Abstract Interpretation, 4, **30**, 33
abstract states, **59**
abstraction family, 93, **95**
abstraction function, **31**
abstraction relation, **40**

abstraction theory, 30, **34**
action system, **67**, 90
adequate, adequacy, **49**, 147
analysis, **4**
approximant, **19**, 121
approximation, 105
approximation order, **36**, 101
approximation relation, *see* approximation order

base, **40**
BDD, *see* binary decision diagram
behavioural equivalence, **25**
binary decision diagram, **11**, 28, 96, 109–111
bisimulation, **27**, 117, 146, **149**, 154
bottom (element), **15**
BPRA, *see* partition refinement algorithm, behavioural
branching bisimulation, 117, 154

characteristic predicate, **120**
companion, 122, **125**
complete partial order (cpo), **16**
concretisation function, **31**
concretisation relation, **40**
consistent, consistency, **50**, **122**
continuous, **16**

description relation, 31, **35**
  approximative, 36, **39**
difunctional, **14**, 50
dining mathematicians, **69**, 82, 86, 138
distinguishing power, **25**
downward-chain-limited, **15**
downwards-closed, **15**

embedding, **16**
equivalence
  behavioural, 117, **118**, 136
  logical, **25**, 117, **118**
expressive power, **25**
extensive, **16**

fine, fineness, **49**, 147, 149
finitely branching, *see* image-finite
fixed point (fixpoint), **16**
  least, **16**
formal methods, **2**
formula
  path, **18**
  state, **18**

Galois connection, **17**, 32, 42, 43, 93
Galois insertion, **18**, 32, 43, 103, 106
Galois-connection framework, **44**
Galois-insertion framework, **44**, 60
game, 150, 173
greatest lower bound (glb), **15**

image-finite, **14**, **21**, 158
initial state, **21**
  abstract, **61**
interpretation, **5**, 6
  abstract, **7**, 32, 68, 72, 79, 89, 96, 101
    constrained, **69**
    free, **68**
interpretation function, **21**, 51
  (concrete), 68
  abstract, 51
  non-standard, *see* abstract

Kripke structure, **22**
  Abstract, 59, **66**

labelling function, **22**
least upper bound (lub), **15**
level, **20**
literals, **18**
lower bound, **15**

maximal element, **15**
minimal element, **15**
model checking, 5, **27**, 89, 127
  symbolic, 11
monotonic, **16**

next(-state) operator, **18**, 152–155

optimal, optimality, 32, 38, **39**, 83, 105

part, **21**
partially ordered set (poset), **15**
partition refinement, 116
partition refinement algorithm, 116, 142, 163, 178
  behavioural, **142**, 146
  logical, **142**
path, **21**, 147
power construction, **44**, 60
PRA, *see* partition refinement algorithm
pre-extensive, **16**
pre-monotonic, **16**
pre-reductive, **16**
prefix, **21**, 147
preservation, 33, 34, 100
  strong, **7**, 33, 35, **49**, 62, 94, 116, 122
  weak, **7**, 35, **36**, 101
principal filter, **15**
principal ideal, **15**
property
  existential, **18**, 62, 108
  invariance, **4**
  liveness, 9, **18**, 100, 108
  safety, 4, 9, **18**, 100, 108
  universal, 9, **18**, 62, 100, 108
proposition, **18**
pseudo-simulation, **26**, 77, 79, 103, 104

reactive, **2**, 5, 58, 90, 100, 110, 112
reductive, **16**
refinement, **92**
  domain driven, **116**
  property driven, **116**
relation, **14**

safe, safety, 31, 32, **35**, 53, 76, 79, 95, 97
simulation, **26**, 103, 106
simulation equivalence, **27**, 133, 142
splitting algorithm, *see* partition refinement algorithm
stuttering equivalence, 155, **157**
  divergence blind, **156**
  divergence sensitive, **157**
  flat, **169**
  flat star, **174**
Stuttering Lemma, **161**
su x, **21**

temporal logic, **18**
top (element), **15**
total, totality, **14**, 120, 146
transition relation, **21**
  abstract, **62**
    computed, **69**
    constrained, **63**, 104
    free, **65**
transition system, **21**
  mixed, **66**

upper bound, **15**
upward-chain-limited, **15**
upwards-closed, **15**

verification, **2**

Curriculum Vitae

July 25, 1966 Born in Voorburg (ZH), The Netherlands.

1972–1977 Frans-ten-Boschschool, Lichtenvoorde (GLD).

1977–1978 School met den Bijbel, Hedel (GLD).

May 1984 Diploma gymnasium-β, Jeroen-Boschcollege, 's-Hertogenbosch.

1984–1990 Eindhoven University of Technology. Computing Science.

Jan.–July 1989, Jan. & Feb. 1990, June & July 1990 Weizmann Institute of Science, Rehovot, Israel. Traineeship and research on the topic of logic programming, under supervision of prof.dr. E. Shapiro.

Aug. 1990 Master's degree (cum laude) in computing science, Eindhoven University of Technology, under supervision of dr. R.T. Gerth, M. Codish and dr. R.P. Nederpelt. Title master's thesis: "Abstract Interpretation of Concurrent Logic Programs for Analysis of Variable Sharing".

Dec. 1990–Feb. 1995 PhD, Eindhoven University of Technology, under supervision of dr. R.T. Gerth and prof.dr. J.C.M. Baeten.

Mar. 1995–Mar. 1996 Civil service, Utrecht University. Design and implementation of a toolset for the specification language μCRL, jointly with dr. J.F. Groote.

current address:
faculty of Mathematics & Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

e-mail: `wsindd@win.tue.nl`