Fair Model Checking of Abstractions

(Extended Abstract)

Dennis Dams* Rob Gerth† Orna Grumberg‡

Abstract

Model checking temporal logic over abstracted transition systems under fairness constraints needs to be done with some care. This paper discusses why and how.

1 Introduction

This section introduces the concepts of abstraction in model checking and fairness at an informal level and presents a motivating example. For technical definitions, we refer to [3, 2] and [6, 9].

1.1 Abstraction

Figure 1 shows a (concrete) transition system consisting of states s_i (s_1 and s_2 are initial[1]) and transitions (thin arrows). Superimposed on it is an *abstraction*. Each abstract state represents a set of concrete states, indicated by drawing each abstract state a_j as a dashed ellipse around the concrete states it represents. The abstract transitions (dashed and fat arrows) are explained further on.

A transition system models the behaviour of some discrete system. Every state is labelled with a set of propositions formalising what is observable about that state. The transitions capture the possible changes of state. Thus, the state labels and the transitions represent, resp., the "static" and "dynamic" of a system. Model checking is answering the question whether a certain correctness requirement, formalised as a temporal logic formula interpreted over such transition systems, is satisfied by a given model. In this paper we consider the logic CTL* — we refer to [5] for its definition. As an example, consider the formula $\forall \mathsf{F} p$ expressing that along all (infinite) paths, p is true at some point. Clearly, it does not hold in the concrete system of Figure 1: The only path satisfying $\mathsf{F}p$ is $s_2, s_5, s_5, \ldots$; the other two paths do not.

An abstraction collapses sets of concrete states into single abstract states, thus indicating that any differences between the concrete states within a single abstract state are ignored. As such it is a method to reduce the size of a model by ignoring certain static aspects. Such a reduction is usually necessary to render model checking feasible. In order to define the state

*Dept. of Electrical Engineering & Dept. of Math. & Comp. Science, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven. E-mail: `d.dams@tue.nl`. Web: `http://www.ics.ele.tue.nl/~dennis`

†Intel Architecture Business Group, SCL, 5200 NE Elam Young Parkway, JFT-104, Hillsboro, OR 97124-6497. E-mail: `robgerth@ichips.intel.com`

‡Computer Science Dept., Technion, Haifa, Israel. E-mail: `orna@cs.technion.ac.il`

[1] The treatment of initiality of states and related issues are suppressed in this extended abstract.

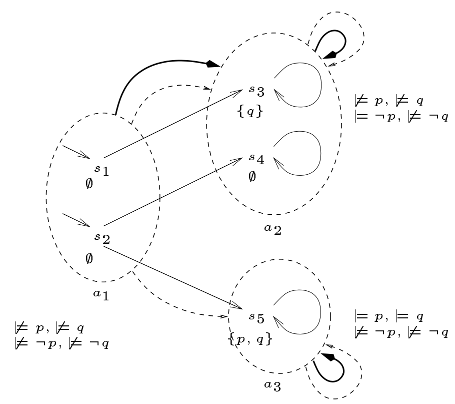

Figure 1: A simple example

labels and the transitions on the abstract level, we must be more precise about the purpose of abstraction. What we require from it, is that temporal logic formulas can be interpreted at the abstract level in such a way that they are *preserved*: If a formula is concluded to hold at the abstract level, it must be true that it also holds on the underlying concrete transition system. Regarding the static aspect, this requirement leads to what can be seen as a 3-valued logic for the valuation of propositions in abstract states: A proposition may be true, false, or unknown. For the abstract states in Figure 1, this is captured by giving truth values to *literals*; a literal is a proposition or its negation. Formally, the value of a literal q in an abstract state a is defined as follows: $a \models q$ iff $\forall_{c \in a}\ c \models q$ (we write $c \in a$ to denote that c is in the set of concrete states that a represents). For example, in state a_2, the literal $\neg p$ is true (meaning $\neg p$ is true in all concrete states in a_2), and consequently the literal p is false. On the other hand, both q and $\neg q$ are false, reflecting that the value of q is indeed unknown if concrete states s_3 and s_4 are identified.

In such a 3-valued setting, one could say that positive and negative information are present independently from each other. This also holds for the transition relation at the abstract level: In order to guarantee the preservation of all CTL* formulas, it turns out that we need to introduce *two* abstract transition relations, called the *free* and the *constrained* relations. *Universal* properties, i.e. properties of the form $\forall\psi$, then have to be interpreted along *free paths*, these are paths formed by repeating free transitions. *Existential* properties ($\exists\psi$) are interpreted along constrained paths. As a result, all formulas that can be expressed in CTL* are preserved from abstract to concrete models. However, formulas need to be in *positive normal form*, i.e. negations may only occur directly in front of propositions. The reason is that the usual definition of negation, $\mathcal{T} \models \neg\varphi$ iff $\mathcal{T} \not\models \varphi$ (for a model $\mathcal{T}$ and a formula φ), fails in the context of preservation.

Technically, the definitions of the abstract transition relations are refinements of the notion of *simulation*. Let R be the concrete transition relation, ${}_\alpha\mathrm{R}^F$ and ${}_\alpha\mathrm{R}^C$ be the free and constrained abstract transition relations resp., s be a concrete state, and a an abstract state containing s. We have that if $\mathrm{R}(s, s')$ for some s', then ${}_\alpha\mathrm{R}^F(a, a')$ for some a' containing s'. For the constrained relation, we have that if ${}_\alpha\mathrm{R}^C(a, a')$ for some a', then $\mathrm{R}(s, s')$ for some s' contained in a'. We stress that these are *consequences* of the definitions of ${}_\alpha\mathrm{R}^F$ and ${}_\alpha\mathrm{R}^C$; for the definitions themselves see [3]. As an example, in Figure 1 the free transitions are shown as dashed arrows and the constrained as thick arrows.

1.2 Fairness

Although every execution run of a system is obtained by iterating state transitions from the transition relation R, one often does not want to consider just any sequence of steps from R as a valid execution. For example, assumptions about scheduling strategies exclude behaviours that forever ignore a particular process. Or, in systems that model real time by distinguishing instantaneous, discrete transitions from time-elapse steps, time should proceed after every finite number of instantaneous transitions. Such an assumption is captured by a so-called *fairness constraint*: A predicate on execution sequences that is true exactly for those sequences that are to be considered valid executions. Verification under fairness then means that path properties are evaluated over fair execution sequences only.

We follow [4] and define a fairness constraint to be any boolean combination of the operator GF ("infinitely often") applied to a literal. Given a fairness constraint, the path quantifiers of CTL* can now be *relativised* to fair paths, resulting in *Fair* CTL* (FCTL*). For example, if we take the formula $\Phi = \mathsf{GF}q$ ("infinitely often q") as fairness constraint, then the formula $\forall\mathsf{F}p$ from above is relativised so that the quantifier $\forall$ only ranges over the paths satisfying Φ. This property is still not satisfied by the concrete system in Figure 1: The path $s_1, s_3, s_3, \ldots$ complies to Φ but does not satisfy $\mathsf{F}p$.

1.3 Abstraction and fairness

Having introduced separately the concepts of abstraction and of fairness, let us now combine them. Consider the *abstract* system in Figure 1. The Φ-relativised property $\forall\mathsf{F}p$ from above is satisfied by it: The only free path complying to $\mathsf{GF}q$ is $a_1, a_3, a_3, \ldots$, and this path satisfies $\mathsf{F}p$. So, φ holds on the abstract but not on the concrete system; in other words: Preservation does not hold for FCTL*.

This outcome is not surprising. Observe that model checking the property $\forall\mathsf{F}p$ under fairness constraint $\Phi = \mathsf{GF}q$ can be restated as the problem of model checking the property $\forall(\mathsf{GF}q \to \mathsf{F}p)$ without fairness constraints. Recall from Section 1.1 that in order to interpret this formula over the abstract transition system, we need to bring it in positive normal form: $\forall(\mathsf{FG}\neg q \vee \mathsf{F}p)$. Now, this formula is *not* satisfied in a_1: There is a free abstract path, namely $a_1, a_2, a_2, \ldots$, that satisfies neither $\mathsf{FG}\neg q$ nor $\mathsf{F}p$. Nevertheless, this approach is not completely satisfactory. It requires reasoning in a logic that allows to express the kinds of fairness one is interested in. Although CTL* enjoys this property, this need in general not be the case, as e.g. for the logic Fair CTL, on which the model checker SMV is based. Furthermore, model checking tools often treat fairness separately from the temporal logic properties to be checked (also called *algorithmically*), for a number of reasons. First, it is often computationally more efficient to deal with fairness algorithmically: In order to encode fairness in the correctness formula, it is often required to introduce auxiliary variables in a model, and also the increased size of the formula may lead to an unacceptable blow-up of the state space. For example, the SPIN model checker ([7]) deals algorithmically with (weak) fairness for these reasons, even though the logic LTL that it uses is expressive enough for fairness. Second, fairness constraints often belong rather to the model than to a particular property being checked. Technically, transition system models are often ω-automata, i.e. they are accompanied by *acceptance conditions* which are a particular form of fairness constraints. As a practical consequence, the fairness constraints often remain the same while different properties are being checked.

In summary, we can say that on abstract systems, the negation in CTL*, and hence the algorithmic treatment of fairness, needs to be approached with care. The reason is the non-complementarity of positive and negative (or: definite and possible) information in abstract systems. This phenomenon occurs not only for the static aspect — the valuation of propositions in states —, but also for the dynamic — the information conveyed by the two types of abstract transitions. Requiring formulas to be in positive normal form (as in [3]), hence limiting the use of negation, is not possible if we want to consider formulas with relativised path quantifiers. Therefore, we define, in Section 2, the interpretation over abstract systems of CTL* with general negation in such a way that the central preservation result from [3] is recovered. Based on this, we can then define what it means for an abstract path to be fair, and extend the interpretation of formulas to FCTL* while maintaining preservation.

In [8], a similar result appears for the less general temporal logic LTL. The results presented below offer a more general framework, though it should immediately be said that the treatment of fairness in the context of abstraction is not the central issue of [8].

2 General Negation

It is possible to allow negations in front of arbitrary subformulas as long as we make sure that, in abstract systems, the interpretation of any path quantifier and any proposition is sensitive to the number of negations in whose scope it occurs. This is formalised by introducing annotations on path quantifiers and propositions, which we write as superscripts: $\cdot^C$ and $\cdot^F$. A proposition p^C is interpreted in the same way as p; p^F gives the dual information: it is true in every abstract state that does *not* have $\neg p$ in its label. For a path quantifier ($\exists$ and $\forall$), a C annotation indicates that it should be evaluated over constrained paths, while an F annotation prescribes the evaluation over free paths. Also, we introduce a syntactic operator t on formulas that toggles all annotations: Cs become Fs, and conversely. *In the sequel, an unannotated formula will be seen as an abbreviation of its "default" annotated form, i.e. all propositions and $\exists$s are annotated with C, and all $\forall$s with F.* Note that we then have $t(t(\varphi)) = \varphi$ for both annotated and unannotated φ.

The interpretation of CTL* formulas with arbitrary negation over abstract transition systems can now be formally defined. We give here the essential clauses; the rest is the same as in the standard case.

2.0.1 Definition *Let a be an abstract state, p a proposition, and $\varphi, \psi \in$ CTL*, where ψ is a path formula.*

(1^C) $a \models p^C$ iff p has value true in a (i.e. $a \models p$).

(1^F) $a \models p^F$ iff $\neg p$ has value false in a (i.e. $a \not\models \neg p$).

(neg) $a \models \neg\varphi$ iff $a \not\models t(\varphi)$.

(6^F) $a \models \forall^F \psi$ iff for every free a-path π, we have $\pi \models \psi$; $a \models \exists^F \psi$ iff there exists a free a-path π such that $\pi \models \psi$.

(6^C) $a \models \forall^C \psi$ iff for every constrained a-path π, we have $\pi \models \psi$; $a \models \exists^C \psi$ iff there exists a constrained a-path π such that $\pi \models \psi$.

As can be seen from clause *neg*, the syntactic negation, $\neg$, does not coincide with its semantic counterpart. As a result, certain formulas that are equivalent (over all models) in the concrete case, are not equivalent anymore over all abstract models. A typical example is the formula $\varphi \vee \neg\varphi$ (where φ is an arbitrary formula) versus the formula *true*. On the other hand, many other equivalences remain intact. The following lemma gives some of those.

2.0.2 Lemma *Let $\varphi \in$ CTL* be an unannotated formula and let φ' be any formula obtained from φ by introduction and removal of double negations and application (in any direction) of De Morgan's laws and duality rules for the path quantifiers and temporal operators. Then for all abstract states a, we have $a \models \varphi'$ iff $a \models \varphi$.*

One consequence of this lemma is that, for unannotated formulas, $\forall$ may be defined as an abbreviation for $\neg\exists\neg$ (also for relativised quantifiers). So, in retrospect we may drop the cases for $\forall^F$ and $\forall^C$ from Definition 2.0.1. Also, disjunction may be defined in terms of conjunction.

We can now extend the preservation result to formulas that are not necessarily in positive form:

2.0.3 Theorem *For every unannotated $\varphi \in$ CTL*, every abstract state a, and every $c \in a$, we have $a \models \varphi \;\Rightarrow\; c \models \varphi$.*

3 Fairness in Abstractions

We can now formalise the answer to the question what is a fair path in the context of abstracted transition systems. Like the valuation of propositions and the abstract transition relations, also the notion of fairness comes in two, dual flavours.

3.0.4 Definition *Let Φ be a fairness constraint. A path π is Φ^C-*fair *iff $\pi \models \Phi$; π is Φ^F-*fair *iff $\pi \models t(\Phi)$.*

The interpretation of relativised path quantifiers over abstract transition systems can now be defined as follows. Only the replacements for clauses 6^C and 6^F in Definition 2.0.1 are given; the others remain the same. Also, we use the fact that $\forall$ is an abbreviation for $\neg\exists\neg$, giving only the cases for $\exists$.

3.0.5 Definition *Let a be an abstract state, ψ a path formula, and Φ a fairness constraint.*

($\overline{6^F}$) *$a \models \exists^F\psi$ under fairness constraint Φ iff there exists a free Φ^F-fair a-path π such that $\pi \models \psi$.*

($\overline{6^C}$) *$a \models \exists^C\psi$ under fairness constraint Φ iff there exists a constrained Φ^C-fair a-path π such that $\pi \models \psi$.*

We can now state the main result of this paper.

3.0.6 Theorem *For every unannotated $\varphi \in$ FCTL*, every abstract state a, and every $c \in a$, we have $a \models \varphi \;\Rightarrow\; c \models \varphi$.*

We illustrate our results by applying them to our running example. The question is whether the formula $\forall \mathsf{F} p$ under fairness constraint $\Phi = \mathsf{GF} q$, holds on the concrete system. According to Theorem 3.0.6, it suffices to check it on the abstract system. As we are using $\forall$ as an abbreviation for $\neg\exists\neg$, we should replace the formula by $\neg\exists\neg\mathsf{F} p$ — as explained after Lemma 2.0.2, this should be done before annotating it. The resulting formula is in basic syntax, without any abbreviations. Now, we adorn it by the default annotations (see the beginning of Section 2), getting $\neg\exists^C\neg\mathsf{F} p^C$. We then interpret the formula over the abstract system, using Definition 2.0.1. Using clause *neg*, we need to check that *it is not the case that* $\exists^F\neg\mathsf{F} p^F$, under fairness constraint Φ. Using clause $\overline{6^F}$ (Definition 3.0.5), this amounts to checking that *it is not the case that there exists a free* Φ^F*-fair path* π *such that* $\pi \models \neg\mathsf{F} p^F$. I.e., using clause *neg* once more, *it is not the case that there exists a free* Φ^F*-fair path* π *such that it is not the case that* $\pi \models \mathsf{F} p^C$. Applying Definition 3.0.4, this rewrites to: *it is not the case that there exists a free path* π *such that: (a)* $\pi \models \mathsf{GF} q^F$*, and (b) it is not the case that* $\pi \models \mathsf{F} p^C$. Finally, by using the clauses for F, G (which have not been explicitly mentioned), and the clauses 1^C and 1^F (Definition 2.0.1), this rewrites to: *it is not the case that there exists a free path* π *such that: (a) along* π *there are infinitely many states in which* $\neg q$ *has value false* ($\not\models \neg q$)*, and (b) it is not the case that in some state along* π*,* p *has value true* ($\models p$). This does not hold for the abstract system of Figure 1, because indeed there *does* exist such a free path π, namely $a_1, a_2, a_2, \ldots$.

4 Discussion

We have extended the framework of model checking over abstractions of transition systems as developed in [2, 3], to systems with fairness constraints specified external to the correctness formula. The practical relevance of this result is that it allows to incorporate formal abstraction into model checkers that treat fairness algorithmically. Technically, the interpretation of temporal logic over abstract systems is done in a 3-valued world, and by this the naive application of the standard definitions for interpreting fair versions of temporal logics fails.

A question that follows naturally from this discussion, is whether the algorithmic approaches to model checking under fairness are correct in the context of abstraction. For example, the approach proposed in [1] is based on the observation that in a formula in which the path quantifiers are relativised to fair paths, the fairness can be "factored out", resulting in an equivalent formula in which the only relativised quantifiers are of a special form. These special cases are then dealt with by an extension of the standard model checking algorithm, which is often considered as a kind of preprocessing phase. It will be shown in the full paper that the rewrite rules used, are based on equivalences that are also valid over abstract systems.

Finally, we would like to raise a question concerning the relation between abstraction and fairness on one hand, and logic programming on the other, for discussion in the VCL workshop: Do logic programming languages have the appropriate level of abstraction to serve as a finite-state modelling formalism for verification purposes? In particular: Is it possible to express fairness constraints (e.g. à la Büchi or Streett acceptance conditions for ω-automata) within a finite-state logic program?

Acknowledgements We would like to thank Yassine Lakhnech for some useful comments.

References

[1] 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, April 1986.

[2] Dennis Dams, Orna Grumberg, and Rob Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$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, Amsterdam, June 1994. North-Holland/Elsevier. Full version available as Computing Science Note 95/16, Eindhoven University of Technology, Dept. of Math. and Comp. Sc.

[3] Dennis René Dams. *Abstract Interpretation and Partition Refinement for Model Checking.* PhD thesis, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, July 1996.

[4] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[5] 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.

[6] Nissim Francez. *Fairness.* Texts and Monographs in Computer Science. Springer-Verlag, 1986.

[7] Gerard Holzmann. Spin package. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[8] Yonit Kesten and Amir Pnueli. Verification by augmented finitary abstraction. *Information and Computation.* To appear in special issue.

[9] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, New York, 1992 (Vol. 1), 1995 (Vol. 2).