# A Practical Technique for Process Abstraction

Glenn Bruns

Department of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ, UK

**Abstract.** With algebraic laws a process can be simplified before verifying its equivalence with another process. Also needed are laws to allow a process to be simplified before verifying that it satisfies a temporal logic formula. Most previous work on this problem is based on property-preserving mappings between transition systems. The results presented here allow direct simplification of process terms for some important classes of temporal properties.

## 1 Introduction

In attempting to verify that a property holds of a system, a natural approach is to simplify the system in a way that preserves the property. For example, to show that a system is deadlock-free, it is common to ignore computations made by the system and focus instead on its synchronisation skeleton. Simplification can also be achieved by merging states, ignoring components, and hiding actions. Soundness of the approach is only guaranteed if one shows that the property shown of the simplified model will indeed hold of the original model, although this requirement is sometimes ignored in informal proofs.

Within concurrency theory, most work on this abstraction approach to verification (sometimes called *reduction*) has been concerned with property-preserving mappings between transition systems. In Kwong [6], reductions are defined on labelled transition systems that strongly preserve a small but important set of properties including deadlock-freedom, determinacy, and the Church-Rosser property. A more general approach [11] defines classes of mappings, which, if applied to a transition system, will preserve properties expressible in certain classes of temporal logic formulas. Recent work [1] gives specific results for the modal mu-calculus [5] and two of its sub-languages. Central here is the role of $< f, g >$-*simulations*, which are simulation relations on transition systems based on Galois connections. For example, it is shown that if a transition system $< f, g >$-simulates another transition system, then $f$ preserves formulas of a mu-calculus sub-language.

While the existing work is general, it does not provide everything needed to apply abstraction methods in practice. First, the operations are defined on transition systems, not process terms. In practice, building the full transition system may be impossible, as it may be very large or even infinite. Second, most existing work gives no particular abstraction operations, instead defining classes of operations that preserve classes of properties. In practice it can be difficult to find an abstraction operation that suits the problem at hand. Finally, property-preserving mappings are often not what is needed in verification. Rather than showing that an abstraction operation will preserve a property, one would like to show the opposite: that properties shown of the abstracted process will also hold of the original process. A compromise is to show strong preservation, in which a property holds of the original process exactly when it holds of the abstracted process.

In this paper we attempt to address these requirements for practical abstraction techniques. First, we show how process terms can be directly simplified, without first building a transition system. Thus our approach is equally suited to manual or automatic verification, and can be used even for infinite-state systems. Second, we are example-driven: abstraction operations are invented to handle real verification problems — and are therefore likely to be generally useful. Finally, our main theorem relates the original and simplified processes in a useful way. For an abstraction operation $A$ on processes we define a corresponding operation $A'$ on temporal properties such that $A(P)$ has property $\phi$ exactly when $P$ has property $A'(\phi)$.

Before describing our abstraction technique, we briefly review CCS [9], our process notation, and the modal mu-calculus [5], our property notation.

## 2 Defining Processes and Properties

The terms of CCS represent processes that perform *actions*. The set of actions Act is composed of a set of *names* $(a, b, \ldots)$, a set of *co-names* $(\overline{a}, \overline{b}, \ldots)$, and the special action $\tau$, which has no corresponding co-action.

Processes are given as *agents* having the following syntax, where $a$ ranges over actions, $L$ ranges over sets of actions, $A$ ranges over process constants, and $f$ ranges over relabelling functions (functions from Act to Act satisfying $f(\tau) = \tau$ and $f(\overline{\mathsf{a}}) = \overline{f(a)}$):

$$E ::= A \mid 0 \mid a.E \mid E_1 + E_2 \mid E_1 \mid E_2 \mid E \backslash L \mid E[f]$$

The set of all agents is denoted by $\mathcal{P}$. The meaning of agent constants is given by a set of agent definitions of the form $\{A_i \stackrel{\text{def}}{=} E_i \mid i \in I\}$, such that an agent constant in any $E_i$ belongs to $\{A_i \mid i \in I\}$.

The meaning of agents is given as a labelled transition system $(\mathcal{P}, \{\stackrel{a}{\rightarrow} : a \in Act\})$, where $\stackrel{a}{\rightarrow}$ is a transition relation $\stackrel{a}{\rightarrow} \subseteq \mathcal{P} \times \mathcal{P}$ for each $a \in Act$. We write $E \stackrel{a}{\rightarrow} E'$ if $(E, E') \in \stackrel{a}{\rightarrow}$. The relation $\stackrel{a}{\rightarrow}$ is defined to be the least relation satisfying the following:

$$a.E \stackrel{a}{\rightarrow} E$$

$$E \stackrel{a}{\rightarrow} E' \Rightarrow E{+}F \stackrel{a}{\rightarrow} E', F{+}E \stackrel{a}{\rightarrow} E', E|F \stackrel{a}{\rightarrow} E'|F, F|E \stackrel{a}{\rightarrow} F|E',$$
$$E[f] \stackrel{f(a)}{\rightarrow} E'[f]$$

$$E \stackrel{a}{\rightarrow} E', F \stackrel{\overline{a}}{\rightarrow} F' \Rightarrow E|F \stackrel{\tau}{\rightarrow} E'|F'$$

$$E \xrightarrow{a} E', a \notin L \cup \overline{L} \Rightarrow E \backslash L \xrightarrow{a} E' \backslash L$$
$$E \xrightarrow{a} E', A \stackrel{\text{def}}{=} E \Rightarrow A \xrightarrow{a} E'$$

The set of labels that a process can perform eventually is called its *sort*. For example, the sort of $A \stackrel{\text{def}}{=} a.A + b.0$ is $\{a, b\}$.

An example CCS process, demonstrating simple mutual exclusion, is the following:

$$\begin{aligned} U_i &\stackrel{\text{def}}{=} acq.\overline{\texttt{enter}}_i\,.\,\overline{\texttt{exit}}_i\,.\,\texttt{rel}\,.U_i + \overline{\texttt{wait}}_i\,.U_i \\ Sem &\stackrel{\text{def}}{=} \overline{\texttt{acq}}\,.\,\overline{\texttt{rel}}\,.Sem \\ M &\stackrel{\text{def}}{=} (U_1 \mid U_2 \mid Sem) \setminus \{acq, rel\} \end{aligned}$$

The user process $U_i$ either performs action $\texttt{acq}$ to acquire the semaphore $Sem$, or performs action $\overline{\texttt{wait}}_i$. The actions $\overline{\texttt{enter}}_i$ and $\overline{\texttt{exit}}_i$ signify entrance to and exit from the critical section. The process $M$ is a system with two users and a semaphore, having sort $\bigcup_{i \in \{1,2\}} \{\overline{\texttt{enter}}_i, \overline{\texttt{exit}}_i, \overline{\texttt{wait}}_i\}$.

We use the modal mu-calculus [5] in a slightly extended form [12] as a temporal logic to express behavioural properties. The syntax of the extended mu-calculus is as follows, where $L$ ranges over sets of actions and $Z$ ranges over variables:

$$\phi ::= Z \;\big|\; \neg\phi \;\big|\; \phi_1 \wedge \phi_2 \;\big|\; [L]\phi \;\big|\; \nu Z.\phi$$

The operator $\nu Z$ binds free occurrences of $Z$ in $\phi$, with the syntactic restriction that free occurrences of $Z$ in a formula $\phi$ lie within an even number of negations.

The meaning of formulas is given as the set of agents $\|\phi\|^{\mathcal{P}}_{\mathcal{V}}$ satisfying the formula $\phi$ relative to a valuation $\mathcal{V}$, which maps variables to sets of agents, and a fixed transition system $(\mathcal{P}, \{\xrightarrow{a} \mid a \in Act\})$. The notation $\mathcal{V}[\mathcal{P}'/Z]$ stands for the valuation $\mathcal{V}'$ which agrees with $\mathcal{V}$ except that $\mathcal{V}'(Z) = \mathcal{P}'$. Since the transition system is fixed we normally drop the agent set and write simply $\|\phi\|_{\mathcal{V}}$. The set $\|\phi\|^{\mathcal{P}}_{\mathcal{V}}$ is defined as follows:

$$\begin{aligned} \|\neg\phi\|_{\mathcal{V}} &= \mathcal{P} - \|\phi\|_{\mathcal{V}} \\ \|\phi_1 \wedge \phi_2\|_{\mathcal{V}} &= \|\phi_1\|_{\mathcal{V}} \cap \|\phi_2\|_{\mathcal{V}} \\ \|[L]\phi\|_{\mathcal{V}} &= \|[L]\|_{\mathcal{V}} \|\phi\|_{\mathcal{V}} \\ \|Z\|_{\mathcal{V}} &= \mathcal{V}(Z) \\ \|\nu Z.\phi\|_{\mathcal{V}} &= \bigcup \{\mathcal{P}' \subseteq \mathcal{P} \mid \mathcal{P}' \subseteq \|\phi\|_{\mathcal{V}[\mathcal{P}'/Z]}\} \end{aligned}$$

where $\|[L]\|^{\mathcal{P}}_{\mathcal{V}}$ is defined, for any $\mathcal{P}' \subseteq \mathcal{P}$, as the following:

$$\|[L]\|^{\mathcal{P}}_{\mathcal{V}} \mathcal{P}' \stackrel{\text{def}}{=} \{P \in \mathcal{P} \mid \forall P'.\forall a \in L.\text{if } P \xrightarrow{a} P' \text{ then } P' \in \mathcal{P}'\}$$

For a closed formula $\phi$ we write $P \models \phi$ if $P \in \|\phi\|_{\mathcal{V}}$ (the valuation is irrelevant for closed formulas).

Informally, $[L]\phi$ holds of $P$ if $\phi$ holds for all processes $P'$ that can be reached from $P$ through an action $a \in L$. A fixed point formula can be understood by keeping in mind that $\nu Z.\phi$ can be replaced by its "unfolding": the formula $\phi$ with $Z$ replaced by $\nu Z.\phi$ itself. Thus, $\nu Z.\psi \wedge [\{a\}]Z = \psi \wedge [\{a\}](\nu Z.\psi \wedge [\{a\}]Z) = \psi \wedge [\{a\}](\psi \wedge [\{a\}](\nu Z.\psi \wedge [\{a\}]Z)) = \ldots$ holds of any process for which $\psi$ holds along any execution path of $a$ actions.

The operators $\vee$, $\langle a\rangle$, and $\mu$ are defined as duals to existing operators (where $\phi[\psi/Z]$ is the property obtained by substituting $\psi$ for free occurrences of $Z$ in $\phi$):

$$\begin{aligned} \phi_1 \vee \phi_2 &\stackrel{\text{def}}{=} \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \langle L\rangle\phi &\stackrel{\text{def}}{=} \neg[L]\neg\phi \\ \mu Z.\phi &\stackrel{\text{def}}{=} \neg\nu Z.\neg\phi[\neg Z/Z] \end{aligned}$$

These additional abbreviations are also convenient (where $L$ ranges over sets of actions, and Act is the set of CCS actions):

$$\begin{aligned} [a_1,\ldots,a_n]\phi &\stackrel{\text{def}}{=} [\{a_1,\ldots,a_n\}]\phi \\ [-]\phi &\stackrel{\text{def}}{=} [Act]\phi \\ [-L]\phi &\stackrel{\text{def}}{=} [Act - L]\phi \\ \mathtt{tt} &\stackrel{\text{def}}{=} \nu Z.Z \\ \mathtt{ff} &\stackrel{\text{def}}{=} \neg\,\mathtt{tt} \end{aligned}$$

For example, consider the formalisation in the mu-calculus of some properties of our mutual exclusion example. The property "process $M$ may perform $\overline{\mathtt{wait}_1}$" can be expressed as the formula $\langle\overline{\mathtt{wait}_1}\rangle\,\mathtt{tt}$. The property "$M$ is deadlock free" can be expressed as $\nu Z.\langle -\rangle\,\mathtt{tt} \wedge [-]Z$. The property "user 1 can wait forever" can be expressed as $\nu Z.\langle\overline{\mathtt{wait}_1}\rangle Z$. The property of mutual exclusive execution of the users' critical sections can be expressed as $\nu Z.[\overline{\mathtt{enter}}](\nu Y.[\overline{\mathtt{enter}}]\,\mathtt{ff} \wedge [-\,\overline{\mathtt{exit}}]Y) \wedge [-]Z$. This formula can be read as "whenever an $\overline{\mathtt{enter}}$ action occurs, then another $\overline{\mathtt{enter}}$ action cannot occur until after an $\overline{\mathtt{exit}}$ action occurs.

## 3 Action Abstraction

The abstraction operation on processes that we use here is a generalisation of the relabelling operation of CCS. It subsumes relabelling, restriction, and hiding. Following [7], we define *action abstraction* with a single SOS-style rule.

For processes $P$ and $P'$, actions $a$ and $a'$, and function $f : Act \rightarrow Act$, $\mathcal{A}_f : \mathcal{P} \rightarrow \mathcal{P}$ is an action operator:

$$\frac{P \xrightarrow{a} P'}{\mathcal{A}_f(P) \xrightarrow{a'} \mathcal{A}_f(P')}\ \ a' = f(a)$$

The side condition is true if $f(a)$ is defined and its value is $a'$; otherwise the side condition is false. For example, hiding can be defined as an action operator by $f(a) \stackrel{\text{def}}{=}$ **if** $a \in L$ **then** $\tau$ **else** $a$.

To relate the properties of a process to the properties of an abstracted process, we introduce an operation on formulas of the mu-calculus. For an action operator $\mathcal{A}_f$, we can define a corresponding logical operator $\mathcal{A}_f$ as follows:

$$\begin{aligned}
\mathcal{A}_f(Z) &\stackrel{\text{def}}{=} Z \\
\mathcal{A}_f(\neg\phi) &\stackrel{\text{def}}{=} \neg\mathcal{A}_f(\phi) \\
\mathcal{A}_f(\phi_1 \wedge \phi_2) &\stackrel{\text{def}}{=} \mathcal{A}_f(\phi_1) \wedge \mathcal{A}_f(\phi_2) \\
\mathcal{A}_f([L]\phi) &\stackrel{\text{def}}{=} [F^{-1}(L)]\mathcal{A}_f(\phi) \\
\mathcal{A}_f(\nu Z.\phi) &\stackrel{\text{def}}{=} \nu Z.\mathcal{A}_f(\phi)
\end{aligned}$$

where $F^{-1}(L) \stackrel{\text{def}}{=} \bigcup_{a\in L} f^{-1}(a)$, and $f^{-1}(a) \stackrel{\text{def}}{=} \{b \mid f(b) = a\}$.

Our main theorem shows that the process and logical operators correspond:

**Theorem 1.** *(Action Abstraction Theorem) For all action operators $\mathcal{A}_f$, processes $P$, and closed formulas $\phi$ of the modal mu-calculus:*

$$\mathcal{A}_f(P) \models \phi \textit{ iff } P \models \mathcal{A}_f(\phi)$$

**Proof.** A positive normal form version of the mu-calculus is used, with the fixed point operators defined via ordinal approximants [5]. The proof proceeds by induction on the structure of formulas. The only interesting cases are for the modal operators. We show the $[L]\phi$ case; the $\langle L\rangle\phi$ case is similar. We want to prove that

$$\mathcal{A}_f(P) \models [L]\phi \text{ iff } P \models \mathcal{A}_f([L]\phi)$$

By the definition of $\models$, the left-hand side can be rewritten as

$$\forall P'.(\exists a' \in L.\mathcal{A}_f(P) \xrightarrow{a'} \mathcal{A}_f(P'))) \Rightarrow \mathcal{A}_f(P') \models \phi$$

According to the definition of action operators, the transition is possible just when there is some $a$ such that $P \xrightarrow{a} P'$ and $a' = f(a)$, so we have

$$\forall P'.(\exists a \in F^{-1}(L).P \xrightarrow{a} P') \Rightarrow \mathcal{A}_f(P') \models \phi$$

By induction the right-hand side can be rewritten as $P' \models \mathcal{A}_f(\phi)$, so we get

$$\forall P'.(\exists a \in F^{-1}(L).P \xrightarrow{a} P') \Rightarrow P' \models \mathcal{A}_f(\phi)$$

Using the definition of the box operator, this gives

$$P \models [F^{-1}(L)]\mathcal{A}_f(\phi)$$

which, by the definition of $\mathcal{A}_f([L]\phi)$, gives $P \models \mathcal{A}_f([L]\phi)$. □

A similar result is given in a different guise as the main theorem of [8] if the action operators are seen as contexts, and the corresponding logical operators are seen as cases of the weakest property transformer. However, the goals of [8] are different from ours, and our form of the modal mu-calculus allows sets of actions in the modal operators.

We demonstrate the technique of action abstraction on two small examples chosen for their simplicity and familiarity. The application of the technique to a larger problem is described in [3] (the technique itself is not described there).

## 4 Example – Dining Philosophers

Our formulation of the dining philosophers problem is based on that of [4]. A philosopher, having special states $T$,$H$,$E$,and $I$, for thinking, hungry, eating, and idle, respectively, is defined as follows:

$$\begin{aligned}
T_i &\stackrel{\text{def}}{=} \mathtt{hungry}_i \,.H_i \\
H_i &\stackrel{\text{def}}{=} \mathtt{sit}_i \,.\, \mathtt{fu}_i \,.\, \mathtt{fu}_{i+1} \,.\, \mathtt{eating}_i \,.E_i \\
E_i &\stackrel{\text{def}}{=} \mathtt{idle}_i \,.I_i \\
I_i &\stackrel{\text{def}}{=} \mathtt{fd}_i \,.\, \mathtt{fd}_{i+1} \,.\, \mathtt{rise}_i \,.\, \mathtt{thinking}_i \,.T_i
\end{aligned}$$

A single fork:

$$F_i \stackrel{\text{def}}{=} \overline{\mathtt{fu}}_i \,.\, \overline{\mathtt{fd}}_i \,.F_i + \overline{\mathtt{fu}}_{i-1} \,.\, \overline{\mathtt{fd}}_{i-1} \,.F_i$$

An usher, who keeps at least one philosopher from sitting at the table:

$$\begin{aligned}
U(m) \stackrel{\text{def}}{=} \text{ if } m < (n-1) \text{ then } &\sum_{i=1}^{n} \overline{\mathtt{sit}}_i \,.U(m+1) \\
&+ \sum_{i=1}^{n} \overline{\mathtt{rise}}_i \,.U(m-1)
\end{aligned}$$

A table with $n$ philosophers, $n$ forks, and a single usher:

$$DP_n \stackrel{\text{def}}{=} (\prod_{i=1}^{n} (T_i \mid F_i) \mid U(0)) \backslash \{sit, rise, fu, fd\}$$

The sort of agent $DP_n$, $\cup_{i=1}^{n}\{\mathtt{hungry}_i, \mathtt{eating}_i, \mathtt{idle}_i, \mathtt{thinking}_i\}$, will be abbreviated as $L_D$. Further, we will abbreviate the set $\{sit, rise, fu, fd\}$ of restricting actions as $K$.

The property we wish to show of the dining philosophers is that deadlock can never occur. The property of deadlock freedom is easily defined as a fixed point property:

$$DF \stackrel{\text{def}}{=} \nu Z.\langle - \rangle \, \mathtt{tt} \wedge [-] Z$$

A process is deadlock-free is it can perform an action, and no matter what action it performs, a deadlock-free state is reached.

The abstraction operation we perform is to hide all observable actions of $DP_n$, that is, the actions of $L_D$. As mentioned earlier, hiding is an special case of action abstraction, in which $f(a) \stackrel{\text{def}}{=} \textbf{if } a \in L \textbf{ then } \tau \textbf{ else } a$. We write $P \backslash\!\!\backslash L$ to denote the hiding of an action set $L$ in agent $P$.

An outline of our proof that the dining philosophers are deadlock free is as follows:

$$\begin{array}{lll} & DP_n \models DF & \\ \text{iff} & DP_n \models DF \backslash\!\!\backslash L_D & \text{def. of hiding on DF} \\ \text{iff} & DP_n \backslash\!\!\backslash L_D \models DF & \text{by the action abstraction theorem} \end{array}$$

We have thus reduced the problem of showing that $DP_n$ is deadlock-free to showing that $DP_n \backslash\!\!\backslash L_D$ is deadlock-free. The state space of $DP_n \backslash\!\!\backslash L_D$ is smaller than that of $DP_n$, but we would like to simplify the process term directly. Unfortunately, the agent cannot be simplified with the algebraic laws for observation equivalence or congruence, because these equivalences do not preserve $DF$ (and many other mu-calculus formulas). However, strong bisimulation equivalence (denoted by $\sim$) *does* preserve mu-calculus properties [13]. We begin, then, by moving the hiding operator into the process term using some laws about hiding.

**Lemma 2.**

$$\begin{aligned} (a.P)\backslash\!\!\backslash L &\sim \tau.(P\backslash\!\!\backslash L) && (a \in L) \\ (P+Q)\backslash\!\!\backslash L &\sim P\backslash\!\!\backslash L + Q\backslash\!\!\backslash L && \\ (P \mid Q)\backslash\!\!\backslash L &\sim P\backslash\!\!\backslash L \mid Q\backslash\!\!\backslash L && \\ (P\backslash K)\backslash\!\!\backslash L &\sim (P\backslash\!\!\backslash L)\backslash K && (L \cap K = \emptyset) \\ P[f]\backslash\!\!\backslash L &\sim (P\backslash\!\!\backslash f^{-1}(L))[f] && \end{aligned}$$

**Proof.** The proofs are similar to the proofs for the relabelling laws in [9].

Applying the hiding laws:

$$\begin{aligned} DP_n\backslash\!\!\backslash L_D &\sim (\prod_{i=1}^{n}(T_i \mid F_i) \mid U(0))\backslash K\backslash\!\!\backslash L_D \\ &\sim (\prod_{i=1}^{n}(T_i \mid F_i) \mid U(0))\backslash\!\!\backslash L_D\backslash K \\ &\sim (\prod_{i=1}^{n}(T_i\backslash\!\!\backslash L_D \mid F_i\backslash\!\!\backslash L_D) \mid U(0)\backslash\!\!\backslash L_D)\backslash K \end{aligned}$$

Applying the hiding laws to the components, we get

$$\begin{aligned} T_i\backslash\!\!\backslash L_D &\sim \tau.H_i\backslash\!\!\backslash L_D \\ H_i\backslash\!\!\backslash L_D &\sim sit_i.fu_i.fu_{i+1}.\tau.E_I\backslash\!\!\backslash L_D \end{aligned}$$

$$\begin{aligned} E_i \backslash\!\backslash L_D &\sim \tau.I_i \backslash\!\backslash L_D \\ I_i \backslash\!\backslash L_D &\sim fd_i.fd_{i+1}.rise_i.\tau.T_i \backslash\!\backslash L_D \\ F_i \backslash\!\backslash L_D &\sim F_i \\ U(m) \backslash\!\backslash L_D &\sim U(m) \end{aligned}$$

Renaming $T_i \backslash\!\backslash L_D$ to $T'_i$, etc., and using the fact that $\sim$ is a congruence, we get

$$DP_n \backslash\!\backslash L_D \sim (\prod_{i=1}^{n}(T'_i \mid F_i) \mid U(0)) \backslash K$$

The next simplification step is to remove $\tau$ actions from the process. We now show that, in some circumstances, if a process with its $\tau$ actions removed is deadlock-free then so is the process itself.

First, we define a *DF-relation* to be a binary relation $R$ on agents such that, if $(P, Q) \in R$, then

1. If $P$ is deadlocked then so is $Q$.
2. Whenever $P \xrightarrow{a} P'$ then there exists a sequence $s$ in $Act^*$ and a $Q'$ such that $Q \xrightarrow{s} Q'$ and $(P', Q') \in R$.

**Theorem 3.** *If $(P, Q)$ is a member of a DF-relation then $Q \models DF$ implies $P \models DF$.*

**Proof.** For every state $P'$ reachable from $P$ there exists a state $Q'$ reachable from $Q$ such that $(P', Q') \in R$. Suppose some $P'$ is deadlocked. Then some state $Q'$ would be deadlocked. But since $Q$ is deadlock-free, so is every derivative $Q'$, so $P$ must be deadlock-free. □

The statement of the next theorem uses the notion of a *context*, which is a CCS expression $C$ having a single hole, written as [ ]. We write $C[E]$ for the term obtained by filling the hole in context $C$ with agent $E$. A hole is not allowed to "move" in transitions between filled contexts. For example, the transition $a.[b.0] \xrightarrow{a} b.[0]$ is prohibited. Formally, the terms in the holes must correspond, in a sense defined inductively on the structure of proofs of transitions. A subterm $E$ of $P$ *corresponds* to a subterm $E'$ of $P'$ in a transition $P \xrightarrow{a} P'$ if there exists a proof of the transition such that either $E$ is $P$ and $E'$ is $P'$, or $E$ and $E'$ are identical and correspond due to the binding of variables in the consequent of the final rule application, or $E$ and $E'$ correspond in an antecedent of the final rule application and correspond in the consequent due to the binding of variables in the application.

Note that the empty context is allowed: $a.0 + [b.0] \xrightarrow{b} [0]$, and that contexts can lose their holes: $a.0 + [b.0] \xrightarrow{a} 0$.

**Lemma 4.** *If agent $\tau.P$ is in the scope of no $+$ operators in context $C$, then for a transition*

$$C[\tau.P] \xrightarrow{\tau} C'[P]$$

*it is the case that $C$ is syntactically identical to $C'$.*

**Proof.** The proof of the transition depends on the transition $\tau.P \xrightarrow{\tau} P$. Inspection of the CCS semantic rules shows that the proof of any transition from a $\tau$-transition cannot change a static context. For example, the inference of a $\tau$-transition by the rule for parallel composition is

$$P \xrightarrow{\tau} P' \Rightarrow P|Q \xrightarrow{\tau} P'|Q.$$

The proof of a $\tau$-transition also cannot change for a prefix context, because such a proof depends on exactly one application of the prefix rule, which arises here in the inference of $\tau.P \xrightarrow{\tau} P$. Finally, no $+$ rule will be used in the inference if $\tau.P$ is not in its scope. $\square$

**Theorem 5.** *Let $C$ be a context and $P$ be an agent, such that $P$ is in the scope of no $+$ operators in $C$. Then $C[P] \models DF$ implies $C[\tau.P] \models DF$.*

**Proof.** Let relation $R$ contain the following pairs, for all contexts $C$ and agents $P$:

$$\begin{array}{cl} (P\ ,\ P) & \\ (C[P]\ ,\ C[P]) & \\ (C[\tau.P]\ ,\ C[P]) & \quad P \text{ is in the scope of no } + \text{ operators in } C \end{array}$$

For each pair $(P, Q)$ the conditions required by a DF-relation must be shown. This is trivial for the first two kinds of pairs. So consider agent $C[\tau.P]$. If it is deadlocked then so is $C[P]$, because if a transition can be proved by any transition, then it can be proved by a $\tau$ transition. Next consider transitions from $C[\tau.P]$. If the transition does not depend on $\tau.P \xrightarrow{\tau} P$, then it can be matched by $C[P]$ such that the pair of derivatives are in $R$. If the transition is of the form $C[\tau.P] \xrightarrow{\tau} C'[P]$, then by the previous lemma $C$ and $C'$ are identical. This transition can be matched by the empty transition from $C[P]$, giving a pair of derivatives in $R$. Therefore $R$ is a DF-relation, and by Theorem 3, $C[P] \models DF$ implies $C[\tau.P] \models DF$. $\square$

Removing the $\tau$'s from agent $T_i'$ gives new agent $T_i''$:

$$\begin{aligned} T_i'' &\stackrel{\text{def}}{=} H_i'' \\ H_i'' &\stackrel{\text{def}}{=} sit_i.fu_i.fu_{i+1}.E_i'' \\ E_i'' &\stackrel{\text{def}}{=} I_i'' \\ I_i'' &\stackrel{\text{def}}{=} fd_i.fd_{i+1}.rise_i.T_i'' \end{aligned}$$

By the previous theorem we therefore know that

$$(\prod_{i=1}^{n}(T_i'' \mid F_i) \mid U(0))\backslash K \models DF \Rightarrow DP_n \backslash\!\!\backslash L_D \models DF$$

A philosopher agent now has 6 states instead of 10. This modest improvement gives a reduction from 79 states for $DP_2$ to 14 states, and from 1185 states for $DP_3$ to 169 states. The blow up in state space is still exponential in the number of philosophers, however.

## 5 Example – Scheduler

Our second example is a task scheduling system, taken directly from [9].

Each task to be scheduled is controlled by two actions: $a$ allows the task to begin, and $b$ signals that the task must terminate. The execution of each task is scheduled by a single process:

$$\begin{aligned} A &\stackrel{\text{def}}{=} a.C \\ C &\stackrel{\text{def}}{=} c.E \\ E &\stackrel{\text{def}}{=} b.D + d.B \\ B &\stackrel{\text{def}}{=} b.A \\ D &\stackrel{\text{def}}{=} d.A \end{aligned}$$

Scheduling processes are arranged in a ring, with actions $c$ and $d$ used for communication by a process with its neighbours. Relabelling the scheduling process to enable the formation of a ring, we define $A_i \stackrel{\text{def}}{=} A[f_i]$ and $D_i \stackrel{\text{def}}{=} D[f_i]$, where $f_i \stackrel{\text{def}}{=} (a_i/a, b_i/b, c_i/c, \overline{\mathtt{c}}_{i-1}\,/d)$.

An n-task scheduler can now be assembled from components:

$$\mathit{Sched} \stackrel{\text{def}}{=} (A_1 \mid D_2 \mid \ldots \mid D_n)\backslash c$$

where $c$ denotes the set $\{c_1, c_2, \ldots, c_n\}$ (and similarly for $a$ and $b$). The sort of the scheduler is $\{a_i, b_i \mid 1 \leq i \leq n\}$.

The property we wish to show of the scheduler is that the start-task actions $\{a_i \mid 1 \leq i \leq n\}$ are performed in the cyclic order $a_1, \ldots, a_n, a_1, \ldots, a_n, \ldots$. The mu-calculus formula expressing that actions $a_1, \ldots, a_k$ occur cyclically can be written as

$$\begin{aligned} \mathit{cycle}(a_1, \ldots, a_k) \stackrel{\text{def}}{=} \; & \nu X_1.[a - a_1]\,\mathtt{ff} \wedge [-a]X_1 \;\wedge\; [a_1] \\ & \nu X_2.[a - a_2]\,\mathtt{ff} \wedge [-a]X_2 \;\wedge\; [a_2] \\ & \vdots \\ & \nu X_k.[a - a_k]\,\mathtt{ff} \wedge [-a]X_k \;\wedge\; [a_k]X_1 \end{aligned}$$

Note that this property is rather weak, not requiring that any $a_i$ ever be performed.

Our strategy for showing that property $\mathit{cycle}(a_1, \ldots, a_n)$ is satisfied by the scheduler closely follows the pattern of the last example. Here, we hide the actions $\{b_i \mid 1 \leq i \leq n\}$. The proof outline is as follows:

$$\begin{array}{lll} & \mathit{Sched} & \models \mathit{cycle}(a_1, \ldots, a_n) \\ \text{iff} & \mathit{Sched} & \models \mathit{cycle}(a_1, \ldots, a_n)\backslash\!\!\backslash b \\ \text{iff} & \mathit{Sched}\backslash\!\!\backslash b & \models \mathit{cycle}(a_1, \ldots, a_n) \end{array}$$

Again, our first step is to move the hiding operator into the scheduler process using the hiding laws. We arrive at

$$\mathit{Sched}\backslash\!\!\backslash b \sim (A'_1 \mid D'_2 \mid \ldots \mid D'_n)\backslash c$$

where

$$A'_i \stackrel{\text{def}}{=} a_i.c_i.E'_i$$
$$E'_i \stackrel{\text{def}}{=} \tau.D'_i + \overline{\mathsf{c}}_{i-1}\,.\tau.A'_i$$
$$D'_i \stackrel{\text{def}}{=} \overline{\mathsf{c}}_{i-1}\,.A'_i$$

To eliminate the $\tau$'s from the process terms, we need a result analogous to the one for deadlock-freedom in the previous example. In what follows, $<$ and $\leq$ will denote the strong simulation and weak simulation, respectively.

**Theorem 6.** *Let $a_1, \ldots, a_n$ be non-$\tau$ actions and let $P \leq Q$. Then $Q \models \mathit{cycle}(a_1, \ldots, a_n)$ implies $P \models \mathit{cycle}(a_1, \ldots, a_n)$.*

**Proof.** Assume $P \leq Q$ and $Q \models \mathit{cycle}(a_1, \ldots, a_n)$. Suppose $P$ does not satisfy $\mathit{cycle}(a_1, \ldots, a_n)$. Then $P$ can perform a sequence of observable actions such that $a_1, \ldots, a_n$ are not in order. But since $P \leq Q$, the same sequence can be performed by $Q$, contrary to assumption. So $P$ must satisfy $\mathit{cycle}(a_1, \ldots, a_n)$. $\square$

**Theorem 7.** *Let $a_1, \ldots, a_n$ be non-$\tau$ actions. Then $C[P] \models \mathit{cycle}(a_1, \ldots, a_n)$ implies $C[\tau.P] \models \mathit{cycle}(a_1, \ldots, a_n)$.*

**Proof.** A weak simulation relation up to $<$ [9, 10] will be exhibited that contains $(C[\tau.P], C[P])$, proving that $C[\tau.P] \leq C[P]$. Then by the previous theorem the result is trivial.

Let relation $R$ contain the following pairs, for all contexts $C$ and agents $P$:

$$(P\ ,\ P)$$
$$(C[P]\ ,\ C[P])$$
$$(C[\tau.P]\ ,\ C[P])$$

For each pair $(P, Q)$ we need to show that every transition $P \xrightarrow{a} P'$ can be matched by a transition $Q \stackrel{a}{\Rightarrow} Q'$ such that, for some $(P_1, Q_1)$ in $R$, $P' < P_1$ and $Q_1 < Q'$. This is trivial for the first two kinds of pairs. So consider a transition of $C[\tau.P]$. If the transition does not depend on $\tau.P \xrightarrow{\tau} P$ then the transition can be matched by $C[P]$, resulting in a pair of derivatives in $R$. If the transition is $C[\tau.P] \xrightarrow{\tau} C[P]$, then the pair $(C[P], C[P])$ is in $R$. The interesting case is a transition $C[\tau.P] \xrightarrow{\tau} C'[P]$. Lemma 4 implies that $C$ and $C'$ are identical if $C$ contains no $+$ operators, so $C'$ and $C$ differ only in that $C$ may contain choices not in $C'$. Generally, if two agents $P'$ and $P$ differ only because $P$ has choices not in $P'$, then $P'$ simulates $P$. Therefore $C'[P]$ simulates $C[P]$, and $(C[P], C[P])$ is in $R$. $\square$

Applying the theorem gives

$$\mathit{Sched}\backslash\!\!\backslash b \models \mathit{cycle}(a_1, \ldots, a_n) \text{ iff } (A''_1 \mid D''_2 \mid \ldots \mid D''_n)\backslash c \models \mathit{cycle}(a_1, \ldots, a_n)$$

Eliminating the $b$ actions yields a great reduction in the state space of the scheduler. For 10 tasks, the original scheduler has about 12,000 states versus only 20 for the abstracted version. Generally, a lower bound on the state space of the original scheduler is $2^n$, while the state space for the abstracted version is exactly $2n$.

## 6 Conclusions

The techniques described here may seem familiar, as they are often informally adopted. We regard this not as a shortcoming of our approach, but as a sign that formalising and systematising the techniques is valuable. We have found hiding to be a particularly useful abstraction operation, as suggested by its use in both of the examples. Another useful operation is the relabelling of several actions not of interest to a single observable action.

A key step in simplifying the examples was the removal of $\tau$ actions. The theorems behind this step rely on a notion of context and on reasoning about proofs of transitions. It may be that existing work on generalized contexts [8] and on proved transition systems [2] could be applied or extended in finding more general theorems for simplifying process terms.

For the kinds of processes typically found in practice, such as those in concurrent normal form, the results given here could be combined to give simpler theorems allowing actions to be removed directly without first being hidden. For example, process actions are usually either strictly for synchronisation or strictly for observation. In such cases, a corollary of the results here might state that non-synchronising actions that are not relabelled can be removed, with cyclic properties being strongly preserved.

By attempting to verify properties of other systems, we hope to discover additional abstraction techniques. Since our ultimate goal is to simplify proofs of systems, we are also investigating abstraction operations on proofs of properties.

### Acknowledgements

I would like to thank Colin Stirling for suggesting this line of research and for many helpful discussions on the subject. Thanks also to Javier Esparza and the anonymous CONCUR referees for their useful comments.

## References

1. S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property preserving simulations. In *Proceedings of CAV '92, LNCS 663*, pages 260–273, 1992.
2. Gerard Boudol and Ilaria Castellani. Permutations of transitions: an event structure semantics for CCS and SCCS. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, 1989. LNCS 354.

3. Glenn Bruns. A case study in safety-critical design. In G.v. Bochmann and D.K. Probst, editors, *Proceedings of CAV '91, LNCS 575*, pages 220–233, 1991.
4. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.
5. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
6. Y. S. Kwong. On reduction of asynchronous systems. *Theoretical Computer Science*, 5:25–50, 1977.
7. Kim G. Larsen and Bent Thomsen. A modal process logic. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science*, pages 203–210. IEEE Computer Society Press, 1988.
8. Kim G. Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. In *Proceedings of ICALP '90, LNCS 443*, 1990.
9. Robin Milner. *Communication and Concurrency*. Prentice Hall International, 1989.
10. Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In *Proceedings of CONCUR '92, LNCS 630*, 1992.
11. Joseph Sifakis. Property preserving homomorphisms of transition systems. In *Logics of Programs*, pages 458–473, 1983. LNCS 164.
12. Colin Stirling. An introduction to modal and temporal logics for CCS. In A. Yonezawa and T. Ito, editors, *Concurrency: Theory, Language, and Architecture*, pages 2–20, 1989. LNCS 491.
13. Colin Stirling. Temporal logics for CCS. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, 1989. LNCS 354.