Formula dependent model reduction through elimination of invisible transitions for checking fragments of CTL

Alexander Kick*

Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler,
Universität Karlsruhe, Am Fasanengarten 5,D-76128 Karlsruhe, Germany
Email: kick@ira.uka.de

Abstract

We present a reduction algorithm which reduces Kripke structures by eliminating transitions from the model which do not affect the visible components of the model. These are exactly the variables contained in the specification formula. The reduction algorithm preserves the truth of special CTL formulae. In contrast to formula-dependent reduction algorithms presented so far, which are mostly computationally expensive, our algorithm needs only one pass through the reachable states of the model. Nevertheless, preliminary results show that models are reduced considerably, which is plausible because, in general, the number of visible components of a reactive system is small compared to the number of internal components.

1 Introduction

Model checking [CGL93] has been successfully applied to the verification of large and complex systems. This has been made possible mainly by the introduction of OBDD based techniques [Bry86]. Researchers have tried to make model checking applicable to even larger systems by using abstraction and reduction algorithms. Data abstraction [Lon93] is used to reduce the number of different data values, thus allowing to collapse several states into a class of states. Reduction algorithms, on the other hand, try to cut down the length or number of execution sequences by eliminating states and transitions or whole execution paths.

Several formula independent reduction algorithms have been proposed. Some of them are based on collapsing states which are bisimulation equivalent [BFH$^+$92], [BdS92]. Other reduction methods are partial order approaches that allow parts of the state graph caused by different interleavings of independent, parallel actions to be ignored [Val90], [GW91], [Pel93].

By considering the properties to be verified, however, much better reductions should be achievable. In [ASSSV94], a state equivalence is defined with respect to a given CTL formula. According to this equivalence, the size of each component finite

*Supported by DFG Vo 287/5-2

state machine is reduced in dependance on all other components. In [DGG93], the reduction algorithm, which works for formulae in ACTL, is based on the successive refinement of a model by splitting states with respect to formulae. In contrast to [ASSSV94] where only equivalent paths are combined, the model is maximally reduced with respect to an (ACTL-) formula in [DGG93].

The algorithms in both of these papers are computationally expensive since states and transitions of the model have to be inspected several times. In the case of [DGG93], for instance, the splitting subprocedure, which inspects all states of the model, is called several times, especially if there are subformulae of type AU. Efficiency of the reduction algorithm, however, is important to achieve a faster overall performance of prior reduction and subsequent checking of the reduced model compared to just model checking the original model.

In our paper, we present a reduction algorithm which is efficient, since it needs just one pass through the reachable states of the model. Preliminary results show that our reduction algorithm will nevertheless cut down the size of the model considerably. Thus, our reduction algorithm trades off efficiency of the reduction algorithm with the size of the reduction to maximize the overall performance.

Our approach to reduction which is also formula-dependent is motivated by the fact that the number of visible state components – one can look upon the variables appearing in the specification formula as being the 'visible' components of the system – is in general small compared to the components involved in the internal working of the system. Exploiting this fact one can cut long subpaths from the state-transition-graph where the 'visible' components do not change their state. This is in contrast to most other reduction algorithms where only equivalences between paths are considered. Only in [DGG93] superfluous transitions – with respect to the formula – within paths are also cut from the model.

The reduction algorithm proposed in this paper preserves the truth of fair CTL formulae that fulfill certain restrictions which are not important in practice.

The rest of the paper is structured as follows. In Section 2 we define the subset of CTL* for which our reduction algorithm preserves the truth of the specification formula. In Section 3 the reduction transformation is presented and an efficient algorithm is developed from it. In Section 4 we justify why model checking with prior reduction by our reduction algorithm is in general faster than just model checking the original model. In Section 5 we draw some conclusions and point out further work.

2 Preliminaries – Computation tree logics

Most of this section (except the definition of RCTL and FRCTL) can be found in more detail in [CGL93].

2.1 CTL*, CTL, fair CTL and ACTL

Let AP be the set of atomic proposition names. State and path formulae of the temporal logic CTL* are constructed as follows.

- state formulae (SF)
  - $p \in AP \Rightarrow p \in SF$

  - $f, g \in SF \Rightarrow \neg f, f \vee g \in SF$
  - $f \in PF \Rightarrow E(f) \in SF$

- path formulae (PF)
  - $f \in SF \Rightarrow f \in PF$
  - $f, g \in PF \Rightarrow \neg f, f \vee g, Xf, fUg \in PF$

CTL* is the set of state formulae generated by the above rules.

In the rest of the paper we use the usual abbreviations, e.g. $Ff = trueUf$, $Gf = \neg F \neg f$, $A = \neg E \neg$, $\overset{\infty}{F}f = GFf$.

CTL is a restricted subset of CTL* that is obtained if the following two rules are used to specify the syntax of path formulae.

- $f, g \in SF \Rightarrow \neg Xf, fUg \in PF$
- $f \in PF \Rightarrow \neg f \in PF$

ACTL is a further restriction of CTL. A formula in CTL is also in ACTL if driving the negations in to the literals results in a formula without the E path quantifier.

In fair CTL as it is defined in [CGL93] the path quantifiers are restricted to fair paths. A path is fair with respect to a set of fairness constraints which can be an arbitrary set of states, usually described by a formula of the logic, if each constraint holds infinitely often along the path.

2.2 RCTL, FRCTL

Let AP be the set of atomic proposition names. Formulae of the temporal logic RCTL are constructed as follows.

- Propositional formulae (PF)
  - $p \in AP \Rightarrow p \in PF$
  - $f_1, f_2 \in PF \Rightarrow \neg f_1, f_1 \vee f_2 \in PF$
- Formulae (F)
  - $g_1, g_2 \in F \Rightarrow g_1 \vee g_2, g_1 \wedge g_2 \in F$
  - $f \in PF,\ g \in F \Rightarrow AgUf \in F,\ AGg \in F,\ AGEGg \in F,\ AGE[gUf] \in F$

We define RRCTL to be RCTL with the restriction that the boolean $\wedge$ is only applied to either (several) propositional formulae or formulae of the form AGg, $AGEGg$ or $AGE[gUf]$.

FRCTL is RCTL with fairness constraints defined similar to fair CTL. Fairness constraints in FRCTL are allowed to be only formulae of RRCTL.

RCTL, which is a subset of CTL, contains most of the formulae needed in practice, such as $AG\ p$, AFp, $AGAFp$, $AG(p_1 \rightarrow AFp_2)$, and $AG(p_1 \rightarrow A(p_2Up_3))$ where p, p_1, p_2 are propositional formulae. RCTL also contains formulae of the type $AGEFp$. RCTL thus comprises all formulae but one which were verified for the

IEEE Futurebus+ cache coherence protocol in [Lon93] and the Ethernet protocol in [NS94].

On the one hand, RCTL is less expressive than ACTL because we do not allow the next operator X and allow AU formulae only of a special type. On the other hand, RCTL is more expressive than ACTL since it allows existential path quantifiers of a certain type.

In general, the transformation presented in Section 3 preserves only formulae which do not talk about a special subset of states in combination with existential path quantification. Consequently, formulae of the type $AG(p \rightarrow EFb)$ are not preserved by the transformation. This is the reason for the restricted definition of RCTL above.

2.3 Semantics – Kripke structures

The semantics of the above logics is defined with respect to a Kripke structure.

- A Kripke structure is a 3-tuple $M = (S, R, L)$ where
  - S is a set of states,
  - $R \subseteq S \times S$ is a total transition relation, and
  - $L : S \rightarrow \mathcal{P}(AP)$ labels each state with a set of atomic propositions true in that state.
- A path in M is an infinite sequence of states: $\pi = s_0 s_1 \ldots$ such that $\forall i \geq 0 : (s_i, s_{i+1}) \in R$. The set of paths is denoted by P.
- $\pi^i = s_i s_{i+1} \ldots$

We assume that the Kripke structures we deal with in the following are finite.

S_M, R_M, L_M denote the set of states, the transition relation and the labelling function of the Kripke structure M, respectively.

If f is a state formula, $s \models_M f$ means that f holds at state s in the Kripke structure M. If f is a path formula, $\pi \models_M f$ means that f holds along the path π in the Kripke structure M.

$\models$ is defined inductively as follows (assuming that p is an atomic proposition, f_1 and f_2 are state formulae and g_1 and g_2 are path formulae), where we write $\models$ instead of $\models_M$ for simplicity:

- $s \models p \Leftrightarrow p \in L(s)$
- $s \models \neg f_1 \Leftrightarrow s \not\models f_1$
- $s \models f_1 \vee f_2 \Leftrightarrow s \models f_1 \quad \text{or} \quad s \models f_2$
- $s \models E(g_1) \Leftrightarrow \exists \pi = s \ldots \in P \quad \text{and} \quad \pi \models g_1$
- $\pi \models f_1 \Leftrightarrow \quad \text{if} \quad \pi = s \ldots \quad \text{then} \quad s \models f_1$
- $\pi \models \neg g_1 \Leftrightarrow \pi \not\models g_1$
- $\pi \models g_1 \vee g_2 \Leftrightarrow \pi \models g_1 \quad \text{or} \quad \pi \models g_2$
- $\pi \models X g_1 \Leftrightarrow \pi^1 \models g_1$
- $\pi \models g_1 U g_2 \Leftrightarrow \exists k \geq 0 : \pi^k \models g_2 \wedge \forall 0 \leq j < k : \pi^j \models g_1$

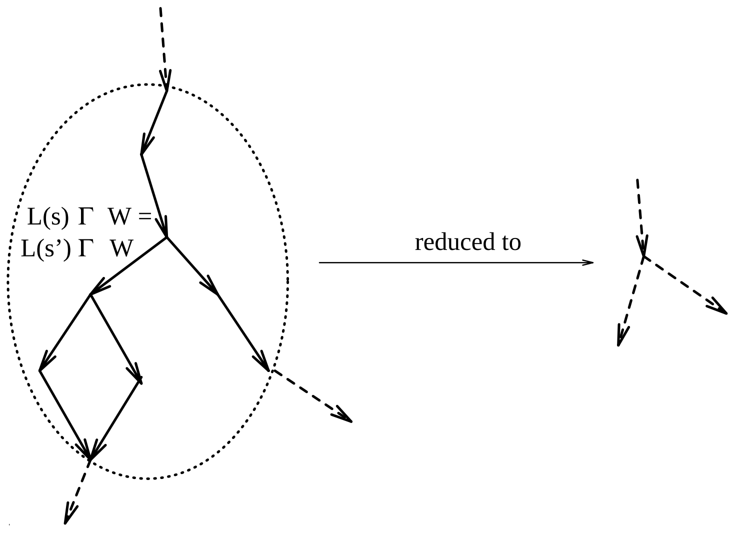

Figure 1: Superfluous subpaths

3 The Reduction

In this section we describe our reduction which transforms the original model into a smaller one. We first explain the idea behind the transformation. Then, we define the transformation precisely, discuss the correctness of the transformation, and finally present the complete algorithm.

3.1 The Idea

A specification formula usually depends only on some small subset of the variables representing the states of the Kripke model. As a consequence, the checking algorithm unnecessarily runs through states which do not affect the variables in the formula at all. By combining such states the model can be reduced without affecting the outcome of the model checking.

Figure 1 shows how our reduction algorithm cuts subsequences of the model where variables (W) contained in the specification formula are left unchanged. If the number of variables in the specification is small compared to the whole number of variables in the model (V) it can be expected that the reduced model is significantly smaller.

Figure 2 shows an example for the full reduction of the original model. The labels below each circle (= state) denote the variables which are true in that state. If only variable p appears in the specification (e.g. $A[trueU\neg p]$ as specification), the two intermediate states can be cut from the original model since they do not change p.

The transformation T described in detail below takes two states s_1 and s_2 which have the same valuations for variables in W and performs the transformations as depicted in Figure 3. To achieve much better reduced models this transformation can be applied several times.

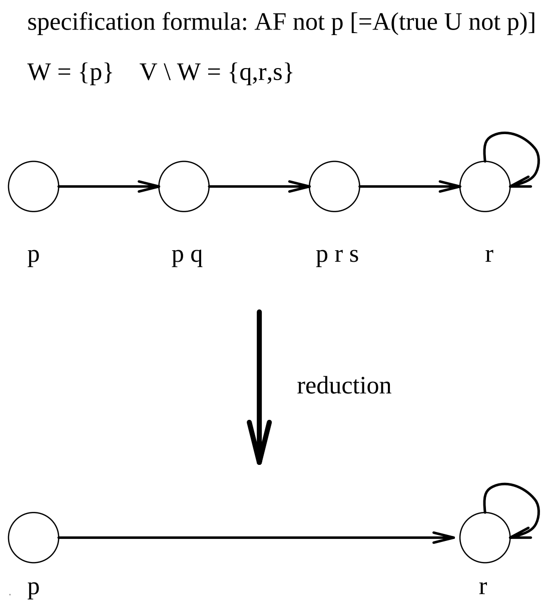
specification formula: AF not p [=A(true U not p)]
W = {p}   V \ W = {q,r,s}
p
p q
p r s
r
reduction
p
r

Figure 2: Example reduction

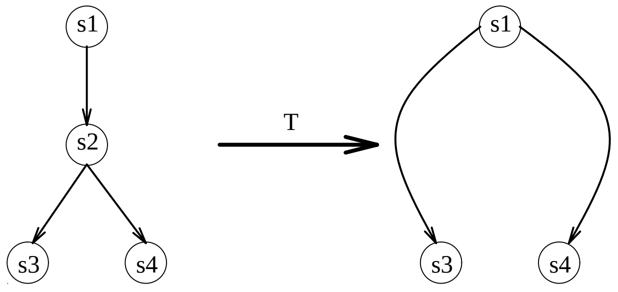

Figure 3: Transformation for the case that $L(s_1) \cap W = L(s_2) \cap W$ and that there is no loop from s_2 to s_2

3.2 The transformation

Let $\mathcal{M}$ denote the set of Kripke models and M the original Kripke model (S, R, L). Let V be the set of variables used to represent the transition relation R. W $(\subset V)$ denotes the set of variables contained in the formula and fairness constraints.

If a model (S, R, L) contains two states s_1 and s_2 which fulfill the condition

$$s_1 R s_2 \wedge \neg s_2 R s_2 \wedge (L(s_1) \cap W = L(s_2) \cap W) \tag{1}$$

its reduced model with respect to these two states is defined as follows:

$$T : S \times S \times \mathcal{M} \to \mathcal{M}$$

$$T(s_1, s_2, (S, R, L)) = (S', R', L')$$

where S', R' and L' are defined below with respect to the two special states s_1 and s_2

$$S' = \begin{cases} S \setminus \{s_2\} & \text{if} \quad \forall s : sRs_2 \to s = s_1 \\ S & \text{otherwise} \end{cases}$$

$$R' = \begin{cases} \begin{array}{c}(R \setminus (\{(s_1, s_2)\} \cup \{(s_2, s) | s_2 R s\})) \\ \cup \{(s_1, s) | s_2 R s\}\end{array} & \text{if} \quad \forall s : sRs_2 \to s = s_1 \\ \\ (R \setminus \{(s_1, s_2)\}) \cup \{(s_1, s) | s_2 R s\} & \text{otherwise} \end{cases}$$

$$L' = L \upharpoonright S' \quad (= L \text{ restricted to } S')$$

In the following, let s_1 and s_2 fulfill condition (1).

3.3 Correspondence between paths

In this subsection we describe how the transformation T affects the paths of the original model M.

Let P_M denote all possible paths in the model M. For all start states $s \in S_M$ we then define:

$$\Pi_s = \{\pi | \pi = s \ldots \wedge \pi \in P_M\}$$

$$\Pi'_s = \{\pi' | \pi' = s \ldots \wedge \pi' \in P_{T(s_1, s_2, M)}\}$$

$$P_{M_r} = \bigcup_{s \in S_{T(s_1, s_2, M)}} \Pi_s$$

Note that P_{M_r} may not contain the paths in M starting with s_2 whereas P_M contains all paths in M.

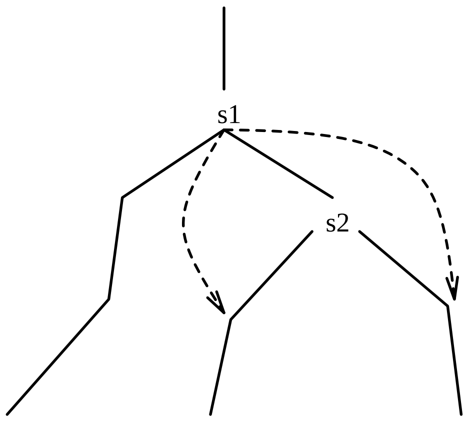

Figure 4: Interrelation between paths in the original and the transformed model

$$U^{succ}_{s_1,s_2} : P_M \to P_{T(s_1,s_2,M)}$$

$$U^{succ}_{s_1,s_2}(ss'x) = \begin{cases} sU^{succ}_{s_1,s_2}(x) & \text{if } s = s_1 \wedge s' = s_2, \\ sU^{succ}_{s_1,s_2}(s'x) & \text{otherwise.} \end{cases}$$

$$U_{s_1,s_2} : P_{M_r} \to P_{T(s_1,s_2,M)}$$

$$U_{s_1,s_2}(ss'x) = \begin{cases} sU^{succ}_{s_1,s_2}(x) & \text{if } s = s_1 \wedge s' = s_2, \\ sU^{succ}_{s_1,s_2}(s'x) & \text{otherwise.} \end{cases}$$

U_{s_1,s_2} induces its inverse $U^{-1}_{s_1,s_2}$ which we describe below for ease of understanding.

$$U^{-1}_{s_1,s_2} : P_{T(s_1,s_2,M)} \to \mathcal{P}(P_{M_r})$$

$$U^{-1}_{s_1,s_2}(srx) = \begin{cases} \{s_1U^{-1}_{s_1,s_2}(rx)\} \cup \{s_1s_2U^{-1}_{s_1,s_2}(rx)\} & \text{if } s = s_1 \wedge s_1R_Mr \wedge s_2R_Mr, \\ s_1s_2U^{-1}_{s_1,s_2}(x) & \text{if } s = s_1 \wedge \neg s_1R_Mr, \\ sU^{-1}_{s_1,s_2}(rx) & \text{otherwise.} \end{cases}$$

U_{s_1,s_2} and $U^{-1}_{s_1,s_2}$ describe how T transforms paths in M into paths in $T(s_1, s_2, M)$ and which paths in M correspond to paths in $T(s_1, s_2, M)$, respectively. Their definition should become clear from Figure 4.

$U_{s_1,s_2}(\pi)$ is the same as π but with some s_2 lost. If s_2 is cut from the model then T also cuts the paths starting at s_2. $U^{-1}_{s_1,s_2}(\pi')$ is exactly the set of those paths in M that are mapped by U_{s_1,s_2} to π' in $T(s_1, s_2, M)$.

Note that T is surjective on the paths. The latter follows from the fact that only the paths in M which go through s_1 are modified and paths starting at s_2 can be lost in the reduced model. I.e., if π is a path in the model M and π does not start with s_2 or s_2 was not cut from the model then $U_{s_1,s_2}(\pi)$ is a path in $T(s_1, s_2, M)$ and if π' is a path in $T(s_1, s_2, M)$ then there exists a path π in M such that $U_{s_1,s_2}(\pi) = \pi'$. T is not injective on the paths, i.e., $\pi \neq \sigma \not\Rightarrow U_{s_1,s_2}(\pi) \neq U_{s_1,s_2}(\sigma)$. This, however, is the case only for paths $\ldots s_1s_2r\ldots$ and $\ldots s_1r\ldots$, i.e., if a successor of s_2 was reachable by a direct arc from s_1 in the original model M.

Lemma 3.1 $\forall s \in S_{T(s_1,s_2,M)}$:

1. $\Pi'_s = U_{s_1,s_2}(\Pi_s)$
2. $\Pi_s = U^{-1}_{s_1,s_2}(\Pi'_s)$

Proof:

1. "$\subseteq$"
   We have to show: $\forall \pi' \in \Pi'_s : \exists \pi \in \Pi_s : U_{s_1,s_2}(\pi) = \pi'$.

   For a given π' simply choose a π out of the nonempty set $U^{-1}_{s_1,s_2}(\pi')$.

   "$\supseteq$"
   We have to show: $\forall \pi \in \Pi_s : U_{s_1,s_2}(\pi) \in \Pi'_s$. This is trivially fulfilled by definition of U_{s_1,s_2} since this is exaclty how the transformation affects the paths of the original model.

2. "$\subseteq$"
   For an arbitrary $\pi \in \Pi_s$, clearly $\pi \in U^{-1}_{s_1,s_2}(U_{s_1,s_2}(\pi))$ by definition of $U^{-1}_{s_1,s_2}$ and U_{s_1,s_2}. By part 1) we have $U_{s_1,s_2}(\pi) \in \Pi'_s$ and thus $\pi \in U^{-1}_{s_1,s_2}(\Pi'_s)$.

   "$\supseteq$"
   For an arbitrary $\pi' \in \Pi'_s$, $U^{-1}_{s_1,s_2}(\pi')$ are paths also starting with s in M. Note that the only change between R_M and $R_{T(s_1,s_2,M)}$ are the arcs starting at s_1 and possibly the arcs starting at s_2. Exactly this change is taken into account by $U^{-1}_{s_1,s_2}$.

■

This lemma allows us to write $U_{s_1,s_2}(\pi)(\pi \in P_M)$ when we want to prove something about paths π' in $T(s_1, s_2, M)$. Similarly, we can restrict our attention to $U^{-1}_{s_1,s_2}(\pi')(\pi' \in P_{T(s_1,s_2,M)})$ when we want to show properties about paths π in M.

3.4 Preservation of the truth of RCTL specification formulae

In this subsection we prove that the transformation preserves the truth of RCTL specification formulae.

T does not preserve the truth for certain CTL* formulae. This is because s_1 and s_2 can be distinguished by the type of paths starting at them. This is illustrated in Figure 5. The formula $p\ U\ (EF\ 1 \wedge EF\ 2 \wedge \neg EF\ 0)$ is true at the path starting at i on the left-hand side but wrong on the right-hand side if the proposition p is true up to s_1 and if there are no other states where 1 and 2 are true.

For this reason we defined RCTL in Section 2 as a subset of CTL which excludes such types of formulae.

Lemma 3.2 *If s_1 and s_2 fulfill condition (1) then:*
$\forall$ propositional formulae $\phi \in RCTL : s_1 \models_M \phi \Leftrightarrow s_2 \models_M \phi$

Proof: The proof is by induction on the length of the propositional formula. Let $p \in AP$ and W be the set of atomic propositions (AP) in the specification formula ϕ.

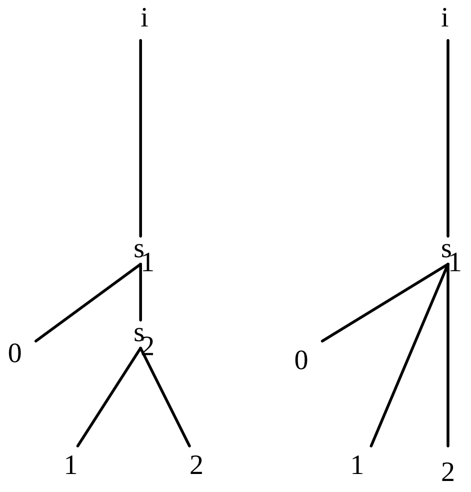

Figure 5: The formula must not distinguish between s_1 and s_2

1. induction base

$$s_1 \models_M p \Leftrightarrow p \in L_M(s_1) \Leftrightarrow p \in L_M(s_2) \Leftrightarrow s_2 \models_M p$$

   because $p \in W \wedge L_M(s_1) \restriction W = L_M(s_2) \restriction W$ (condition (1))

2. The induction step for the boolean connectives $\neg$ and $\vee$ is trivial.

■

Lemma 3.3 *If s_1 and s_2 fulfill condition (1) then:*
$\forall$ propositional formulae $\phi \in RCTL : \forall s \in S_{T(s_1,s_2,M)}$:
$s \models_M \phi \Leftrightarrow s \models_{T(s_1,s_2,M)} \phi$

Proof: The proof is by induction on the structure of the propositional formula. Let $p \in AP$.

1. induction base

$$s \models_M p \Leftrightarrow p \in L_M(s) \Leftrightarrow p \in L_{T(s_1,s_2,M)}(s) \Leftrightarrow s \models_{T(s_1,s_2,M)} p$$

   This is clear since T only restricts L_M to the states in $T(s_1, s_2, M)$.

2. The induction step for the boolean connectives $\neg$ and $\vee$ is trivial.

■

Corollary 3.1 *If s_1 and s_2 fulfill condition (1) then:*
$\forall$ propositional formulae $\phi \in RCTL : s_2 \models_M \phi \Leftrightarrow s_1 \models_{T(s_1,s_2,M)} \phi$

Proof:

$$s_1 \models_{T(s_1,s_2,M)} \phi \Leftrightarrow s_1 \models_M \phi \Leftrightarrow s_2 \models_M \phi$$

by Lemmata 3.3 and 3.2 ■

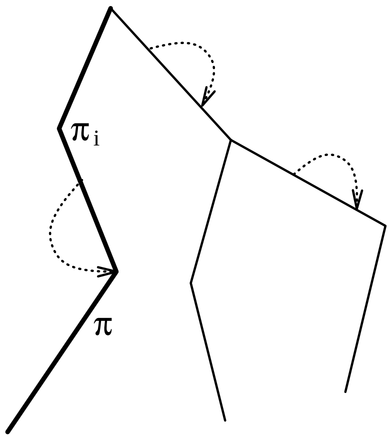

Figure 6: Case AGg

Proposition 3.1 *If s_1 and s_2 fulfill condition (1) then:*
$\forall \phi \in RCTL\colon \forall s \in S_{T(s_1,s_2,M)}\colon s \models_M \phi \Rightarrow s \models_{T(s_1,s_2,M)} \phi$

Proof:

The proof is by structural induction. Let f be a propositional formula and g, g_1, g_2 be formulae.

1. induction base for propositional formulae by Lemma 3.3

2.
$$s \models_M g_1 \vee g_2 \Leftrightarrow s \models_M g_1 \text{ or } s \models_M g_2 \Rightarrow$$
$$s \models_{T(s_1,s_2,M)} g_1 \text{ or } s \models_{T(s_1,s_2,M)} g_2 \Leftrightarrow s \models_{T(s_1,s_2,M)} g_1 \vee g_2$$

3.
$$s \models_M g_1 \wedge g_2 \Leftrightarrow s \models_M g_1 \text{ and } s \models_M g_2 \Rightarrow$$
$$s \models_{T(s_1,s_2,M)} g_1 \text{ and } s \models_{T(s_1,s_2,M)} g_2 \Leftrightarrow s \models_{T(s_1,s_2,M)} g_1 \wedge g_2$$

4.
$$s \models_M AGg \Leftrightarrow$$
$$\forall \pi = s \ldots \forall i \geq 0 : \pi_i \models_M g$$

   Figure 6 illustrates this case. For an arbitrary chosen path $\pi \in P_M$ (drawn bold in the figure) the corresponding path $U_{s_1,s_2}(\pi)$ in $T(s_1, s_2, M)$ is obtained by jumping some s_2 (dotted arrows) – T only cuts some states. By the induction hypothesis, all states in $U_{s_1,s_2}(\pi)$ fulfill g. The last iff is true because by Lemma 3.1 the $U_{s_1,s_2}(\pi)$ are all paths in $T(s_1, s_2, M)$ that start with s.

$$\forall U_{s_1,s_2}(\pi) \forall i \geq 0 : U_{s_1,s_2}(\pi)_i \models_{T(s_1,s_2,M)} g \Leftrightarrow s \models_{T(s_1,s_2,M)} AGg$$

5.

$$s \models_M A[gUf] \Leftrightarrow$$
$$\forall \pi = s \ldots \exists k \geq 0 : \pi_k \models_M f \wedge \forall 0 \leq j < k : \pi_j \models_M g$$

The proof for $A[gUf]$ is similar to the proof for AGg, the only difference lying in the fact that for a fixed path π in M, the state which fulfills f is not cut by T, as becomes clear from the following paragraph.

Let $\pi \in P_M$ be chosen arbitrarily. We suppose that k is the smallest k such that $\pi_k \models_M f$. In that case T may cut some $\pi_j \neq \pi_k$ but not π_k from π. Otherwise, $\pi_{k-1} = s_1$ which would mean that $\pi_{k-1} \models_M f$ since $s_2 \models_M f \Leftrightarrow s_1 \models_M f$ because of Lemma 3.2, in contradiction to the fact that k is the smallest.

By the induction hypothesis $\pi_k \models_{T(s_1,s_2,M)} f$. Since T just cuts some states from a path π (s_2 is jumped if it was preceded by s_1), we can use the induction hypothesis to conclude that all intermediate states up to this state π_k on $U_{s_1,s_2}(\pi)$ fulfill g.

$$\Rightarrow \forall U_{s_1,s_2}(\pi) \exists l \leq k : U_{s_1,s_2}(\pi)_l = \pi_k \models_{T(s_1,s_2,M)} f \wedge$$
$$\forall 0 \leq i < l : U_{s_1,s_2}(\pi)_i = \pi_{q(i)} \models_{T(s_1,s_2,M)} g$$

where $q(0) = 0, q(l) = k$
and $q(i+1) = \begin{cases} q(i) + 2 & \pi^{q(i)} = s_1 s_2 x \\ q(i) + 1 & \text{otherwise} \end{cases}$

The following iff is true because by Lemma 3.1.1 the $U_{s_1,s_2}(\pi)$ are all paths in $T(s_1, s_2, M)$ that start with s – T does not cut or add any paths starting at a state $s \in S_{T(s_1,s_2,M)}$.

$$\Leftrightarrow s \models_{T(s_1,s_2,M)} A[gUf]$$

6.

$$s \models_M AGEGg \Leftrightarrow$$
$$\forall \pi = s \ldots \forall i \geq 0 : \pi_i \models_M EGg \Leftrightarrow$$
$$\forall \pi = s \ldots \forall i \geq 0 : \exists \sigma = \pi_i \ldots \forall j \geq 0 : \sigma_j \models_M g$$

Figure 7 illustrates this case. For an arbitrary chosen path $\pi \in P_M$ the corresponding path $U_{s_1,s_2}(\pi)$ in $T(s_1, s_2, M)$ is obtained by only cutting some states from π. Therefore, for any state s' on the transformed path $U_{s_1,s_2}(\pi)$ there also exists a path on which g is globally true in $T(s_1, s_2, M)$, namely $U_{s_1,s_2}(\sigma)$, the path in $T(s_1, s_2, M)$ corresponding to the path σ in M starting at s' on which g is always true. The existence of $U_{s_1,s_2}(\sigma)$ is guaranteed by Lemma 3.1, and that $U_{s_1,s_2}(\sigma)$ fulfills Gg by the induction hypothesis (note again that only some states are cut from σ to obtain $U_{s_1,s_2}(\sigma)$).

$$\forall U_{s_1,s_2}(\pi) = s \ldots \forall i \geq 0 : \exists U_{s_1,s_2}(\sigma) = U_{s_1,s_2}(\pi)_i \ldots$$
$$\forall j \geq 0 : U_{s_1,s_2}(\sigma)_j \models_{T(s_1,s_2,M)} g$$

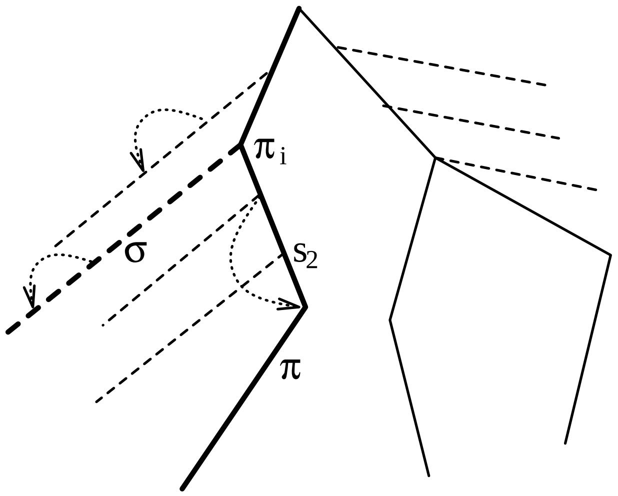

Figure 7: Case $AGEGg$

The last iff is true because by Lemma 3.1 the $U_{s_1,s_2}(\pi)$ are all paths in $T(s_1, s_2, M)$ that start with s.

$$\Leftrightarrow s \models_{T(s_1,s_2,M)} AGEGg$$

7.

$$s \models_M AGE[gUf] \Leftrightarrow$$
$$\forall \pi = s \ldots \forall i \geq 0 : \pi_i \models_M E[gUf] \Leftrightarrow$$
$$\forall \pi = s \ldots \forall i \geq 0 : \exists \sigma = \pi_i \ldots \wedge \exists k \geq 0 : \sigma_k \models_M f \wedge \forall 0 \leq j < k : \sigma_j \models_M g \Rightarrow$$

For each state on the path $U_{s_1,s_2}(\pi)$, which in fact is one of the π_i, since only some states are cut by the transformation T, we can simply take the corresponding $U_{s_1,s_2}(\sigma)$ as the path which models gUf. $U_{s_1,s_2}(\sigma)$ exists by Lemma 3.1. Indeed, reasoning similar to case 5) shows that $\exists l \leq k : U_{s_1,s_2}(\sigma)_l \models_{T(s_1,s_2,M)} f \wedge \forall 0 \leq j < l : U_{s_1,s_2}(\sigma)_j \models_{T(s_1,s_2,M)} g$. Since the $U_{s_1,s_2}(\pi)$ are all paths starting at s (Lemma 3.1) we have:

$$\forall U_{s_1,s_2}(\pi) = s \ldots \forall i \geq 0 : \exists U_{s_1,s_2}(\sigma) = U_{s_1,s_2}(\pi)_i \ldots \wedge$$
$$\exists 0 \leq l \leq k : U_{s_1,s_2}(\sigma)_l \models_{T(s_1,s_2,M)} f \wedge \forall 0 \leq j < l : U_{s_1,s_2}(\sigma)_j \models_{T(s_1,s_2,M)} g \Leftrightarrow$$
$$\forall U_{s_1,s_2}(\pi) = s \ldots \forall i \geq 0 : U_{s_1,s_2}(\pi)_i \models_{T(s_1,s_2,M)} E[gUf] \Leftrightarrow$$
$$s \models_{T(s_1,s_2,M)} AGE[gUf]$$

■

Proposition 3.2 *If s_1 and s_2 fulfill condition (1) then:*
$\forall \phi \in RCTL$:
(a) $\forall s \in S_{T(s_1,s_2,M)} : s \models_{T(s_1,s_2,M)} \phi \Rightarrow s \models_M \phi \wedge$
(b) $s_1 \models_{T(s_1,s_2,M)} \phi \Rightarrow s_2 \models_M \phi$.

Proof:

The proof is by structural induction. Let f be a propositional formula and g be a formula.

1. induction base for propositional formulae by Lemma 3.3 and Corollary 3.1.
2. In an easy way the induction hypothesis can be used to prove the induction step for the boolean connectives $\wedge, \vee$.
3. $$s \models_{T(s_1,s_2,M)} AGg \Leftrightarrow \forall \pi' = s \ldots \forall i \geq 0 : \pi'_i \models_{T(s_1,s_2,M)} g$$

   We consider an arbitrary path $\pi = s \ldots \in U^{-1}_{s_1,s_2}(\Pi'_s)$, with $\pi \in U^{-1}_{s_1,s_2}(\pi')$, $\pi' \in \Pi'_s$.

   (a) i. $\pi_i \neq s_2$ By induction hypothesis, $\pi_i \models_M g$.

   ii. $\pi_i = s_2$
   If s_2 was already on π', then we can reason as in the above case i). Otherwise, s_2 can only be added to π' by $U^{-1}_{s_1,s_2}$ to obtain π if s_2 is preceded by s_1 which already appeared in $\pi' \in P_{T(s_1,s_2,M)}$, where $s_1 \models_{T(s_1,s_2,M)} g$. By induction hypothesis, $s_2 \models_M g$. Therefore:

   $$s \models_M AGg$$

   since the $\pi \in U^{-1}_{s_1,s_2}(\Pi'_s)$ are all paths starting at s in M by Lemma 3.1.

   (b) $s_1 \models_{T(s_1,s_2,M)} AGg \Rightarrow s_1 \models_M AGg$ by case a). This in turn implies that all successors of s_1 in M have to fulfill AGg as well. Therefore:

   $$s_1 \models_{T(s_1,s_2,M)} AGg \Rightarrow s_2 \models_M AGg$$

4. $$s \models_{T(s_1,s_2,M)} A[gUf] \Leftrightarrow$$
   $$\forall \pi' = s \ldots \exists l \geq 0 : \pi'_l \models_{T(s_1,s_2,M)} f \wedge \forall 0 \leq i < l : \pi'_i \models_{T(s_1,s_2,M)} g$$

   We consider an arbitrary path $\pi = s \ldots \in U^{-1}_{s_1,s_2}(\Pi'_s)$. Let $\pi \in U^{-1}_{s_1,s_2}(\pi')$, $\pi' \in \Pi'_s$.

   (a) Of course the first state $q \models_{T(s_1,s_2,M)} f$ in π' is also in π and by induction hypothesis $q \models_M f$. The subpath in π up to q is the same as the one in π' but possibly with some s_2 added. The latter can happen only if s_2 was preceded by s_1.
   By the induction hypothesis and Lemma 3.1 we conclude:

   $$\Rightarrow \forall \pi = s \ldots \exists k \geq l : \pi_k = \pi'_l \models_M f \wedge \forall 0 \leq i < k : \pi_i \models_M g \Leftrightarrow$$
   $$s \models_M A[gUf]$$

   where π_i is either a π'_j or s_2.

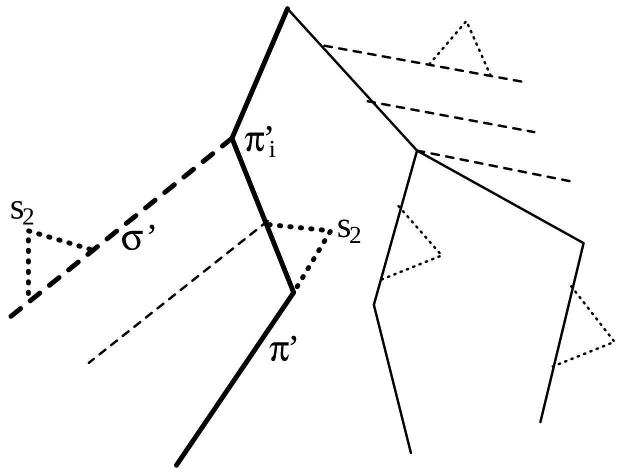

Figure 8: Case $AGEGg$

(b) $s_1 \models_{T(s_1,s_2,M)} A[gUf] \Rightarrow s_1 \models_M A[gUf]$ by case a). If $s_1 \models_M f$, then $s_2 \models_M f$ by the induction hypothesis and thus trivially $s_2 \models_M A[gUf]$. If $s_1 \not\models_M f$, then all successors of s_1 have to fulfill $A[gUf]$ and thus $s_2 \models_M A[gUf]$.

5.

$$s \models_{T(s_1,s_2,M)} AGEGg \Leftrightarrow$$
$$\forall \pi' = s \ldots \forall i \geq 0 : \exists \sigma' = \pi'_i \ldots \forall j \geq 0 : \sigma'_j \models_{T(s_1,s_2,M)} g$$

We consider an arbitrary path $\pi = s \ldots \in U^{-1}_{s_1,s_2}(\Pi'_s)$. Let $\pi \in U^{-1}_{s_1,s_2}(\pi')$, $\pi' \in \Pi'_s$ (Figure 8).

(a) i. $\pi_i \neq s_2$
$\sigma = U^{-1}_{s_1,s_2}(\sigma')$ exists by Lemma 3.1 and can contain some additional s_2 (indicated by dotted triangles in Figure 8), but only if they are preceded by s_1. By induction hypothesis, $\forall j \geq 0 : \sigma_j \models_M g$ and therfore $\pi_i \models_M EGg$.

ii. $\pi_i = s_2$
If s_2 was already on π', then we can reason as in the above case i). Otherwise, s_2 can only be added if it is preceded by s_1 which already appeared in $\pi' \in P_{T(s_1,s_2,M)}$, where $s_1 \models_{T(s_1,s_2,M)} EGg \Rightarrow s_1 \models_{T(s_1,s_2,M)} g$. By the induction hypothesis, $s_2 \models_M g$. The state following π_i on π, which cannot be equal to s_2 by condition (1), models EGg by case i) above. Thus, $s_2 \models_M EGg$. Therefore:

$$s \models_M AGEGg$$

since the $\pi \in U^{-1}_{s_1,s_2}(\Pi'_s)$ are all paths starting at s in M by Lemma 3.1.

(b) $s_1 \models_{T(s_1,s_2,M)} AGEGg \Rightarrow s_1 \models_M AGEGg$ by case a). All states reachable from s_1 in M fulfill EGg. As a consequence, $s_2 \models_M AGEGg$.

6.

$$s \models_{T(s_1,s_2,M)} AGE[gUf] \Leftrightarrow$$
$$\forall \pi' = s\ldots \forall i \geq 0 : \pi'_i \models_{T(s_1,s_2,M)} E[gUf] \Leftrightarrow$$
$$\forall \pi' = s\ldots \forall i \geq 0 : \exists \sigma' = \pi'_i \ldots$$
$$\wedge \exists l \geq 0 : \sigma'_l \models_{T(s_1,s_2,M)} f \wedge \forall 0 \leq j < l : \sigma'_j \models_{T(s_1,s_2,M)} g$$

We consider an arbitrary path $\pi = s\ldots \in U^{-1}_{s_1,s_2}(\Pi'_s)$. Let $\pi \in U^{-1}_{s_1,s_2}(\pi')$, $\pi' \in \Pi'_s$.

(a) i. $\pi_i \neq s_2$
Let $\sigma = U^{-1}_{s_1,s_2}(\sigma')$. σ exists by Lemma 3.1 and can contain some additional s_2, but only if they are preceded by s_1. Argumentation similar to case 4.a) allows us to conclude that $\sigma \models_M gUf$ and thus $\pi_i \models_M E[gUf]$.

ii. $\pi_i = s_2$
If s_2 already was on π', then we can reason as in the above case i). Otherwise, s_2 can only be added if it is preceded by s_1 which already appeared in $\pi' \in P_{T(s_1,s_2,M)}$, where $s_1 \models_{T(s_1,s_2,M)} E[gUf] \Rightarrow s_1 \models_{T(s_1,s_2,M)} g$ or $s_1 \models_{T(s_1,s_2,M)} f$. In the first case, by induction hypothesis, $s_2 \models_M g$. The state following π_i on π models $E[gUf]$ (case a.i) above). Thus, $s_2 \models_M E[gUf]$. The second case is clear.

$$s \models_M AGE[gUf]$$

since the $\pi \in U^{-1}_{s_1,s_2}(\Pi'_s)$ are all paths starting at s in M by Lemma 3.1.

(b) $s_1 \models_{T(s_1,s_2,M)} AGE[gUf] \Rightarrow s_1 \models_M AGE[gUf]$ by case a). This means that all states reachable from s_1 in M fulfill $E[gUf]$. As a consequence, $s_2 \models_M AGE[gUf]$.

■

The following theorem follows immediately from the two propositions.

Theorem 3.1 *If s_1 and s_2 fulfill condition (1) then:*
$\forall \phi \in RCTL : \forall s \in S_{T(s_1,s_2,M)} : s \models_M \phi \Leftrightarrow s \models_{T(s_1,s_2,M)} \phi$

A similar theorem can be proven for FRCTL. Let $H = \{h_0, \ldots, h_n\}$ be the set of fairness constraints. We define Δ_s as the set of fair paths in M starting with s and Δ'_s as the set of fair paths in $T(s_1,s_2,M)$ starting with s.

$$\Delta_s = \{\pi \in \Pi_s | \pi \models_M \bigwedge_i \overset{\infty}{F} h_i\}$$

$$\Delta'_s = \{\pi \in \Pi'_s | \pi' \models_{T(s_1,s_2,M)} \bigwedge_i \overset{\infty}{F} h_i\}$$

Lemma 3.4 *If s_1 and s_2 fulfill condition (1) then:*
$\forall \phi \in RRCTL : s_2 \models_M \phi \Rightarrow (s_1 \models_M \phi \vee \exists r : s_2 R r \wedge r \neq s_2 \wedge r \models_M \phi)$

Proof: By induction.

1. induction base for propositional formulae by Lemma 3.2
2. Since for any state s: $s \models_M g_1 \Rightarrow s \models_M g_1 \vee g_2$, we can conclude, using the induction hypothesis:

$$s_2 \models_M g_1 \vee g_2 \Rightarrow$$
$$(s_1 \models_M g_1 \vee g_2) \vee (\exists r : s_2 R r \wedge r \neq s_2 \wedge r \models_M g_1 \vee g_2)$$

   In the remaining cases the induction hypothesis is not needed.

3. $$s_2 \models_M A[gUf]$$

   If $s_2 \models_M f$ then $s_1 \models_M A[gUf]$ by Lemma 3.2. Otherwise, every successor of s_2 models $A[gUf]$. Therefore:

$$s_2 \models_M A[gUf] \Rightarrow s_1 \models_M A[gUf] \vee \exists r : s_2 R r \wedge r \neq s_2 \wedge r \models_M A[gUf]$$

   The existence of such an r is guaranteed by condition (1) and totality of R_M.

4. For $\phi = AGg$ or $\phi = AGEGg$ or $\phi = AGE[gUf]$ we can argue similarly. If $s_2 \models_M \phi$ then for every successor r of s_2: $r \models_M \phi$. A successor $r \neq s_2$ exists by condition (1) and totality of R_M.
5. Let g_i be one of AGg, $AGEGg$ or $AGE[gUf]$. If $s_2 \models_M \bigwedge_i g_i$ then, $\forall i$: $s_2 \models_M g_i$, and as in the previous case, for every successor r of s_2: $r \models_M g_i$ and consequently, $\exists r : s_2 R r \wedge r \neq s_2 \wedge r \models_M \bigwedge_i g_i$.

■

Lemma 3.5 $\forall s \in S_{T(s_1,s_2,M)}$:

1. $\Delta'_s = U_{s_1,s_2}(\Delta_s)$

2. $\Delta_s = U^{-1}_{s_1,s_2}(\Delta'_s)$

Proof: Since $\Pi'_s = U_{s_1,s_2}(\Pi_s)$ and $\Pi_s = U^{-1}_{s_1,s_2}(\Pi'_s)$ by Lemma 3.1 it suffices to show that

- $\pi \in \Delta_s \Rightarrow U_{s_1,s_2}(\pi)$ is fair
- $\pi' \in \Delta'_s \Rightarrow U^{-1}_{s_1,s_2}(\pi')$ are fair paths

Let $h_i \in H$ $(0 \leq i \leq n)$ be an arbitrary fairness constraint.

1. $\pi \in \Delta_s \Rightarrow U_{s_1,s_2}(\pi)$ is fair
   If π is fair then there is at least one state $s \models_M h_i$ which appears infinitely often on π since M is finite. If $s \neq s_2$ then clearly $U_{s_1,s_2}(\pi)$ is also fair since $s \models_{T(s_1,s_2,M)} h_i$ by Theorem 3.1 and s also appears infinitely often on $U_{s_1,s_2}(\pi)$. The only case where fairness might be lost in $U_{s_1,s_2}(\pi)$ is when $s = s_2$ and s_2

is the only such state and s_2 is cut so often from π that it appears just finitely many times in $U_{s_1,s_2}(\pi)$. We show, however, that this can never be the case.

Whenever s_2 is cut from π its predecessor is s_1 and it has a successor $r \neq s_2$ on that path π. Together with Lemma 3.4 we can conclude that if s_2 can be cut infinitely often then h_i is also fulfilled infinitely often in M by a set of states which does not contain these cuttable s_2.

2. $\pi' \in \Delta'_s \Rightarrow U^{-1}_{s_1,s_2}(\pi')$ is fair
The only state added to π' is s_2 and s_2 can only be added if it is preceded by s_1. Therefore, every state on π' other than s_2 appears equally often on $U^{-1}_{s_1,s_2}(\pi')$. Let s with $s \models_{T(s_1,s_2,M)} h_i$ and appear infinitely often on π', then both $s \models_M h_i$ by Theorem 3.1 and s appears infinitely often on $U^{-1}_{s_1,s_2}(\pi')$.

■

Lemma 3.6 $s_1\Delta_{s_2} = \Delta_{s_1} \cap s_1 s_2 S^\omega_M$

Proof: If $\pi = s_1 s_2 \rho$ is a fair path then clearly $s_2\rho$ is also a fair path. If $s_2\rho$ is a fair path then clearly $\pi = s_1 s_2 \rho$ is a fair path. ■

Theorem 3.2 *If s_1 and s_2 fulfill condition (1) then:*
$\forall \phi \in FRCTL : \forall s \in S_{T(s_1,s_2,M)} : s \models_M \phi \Leftrightarrow s \models_{T(s_1,s_2,M)} \phi$

Proof: In the same way as the proof of Theorem 3.1, i.e., the proofs of Lemmata 3.2 and 3.3, Corollary 3.1 and Propositions 3.1 and 3.2, if we substitute $\phi \in RCTL$ by $\phi \in FRCTL$, P_M by Q_M which shall denote the set of all fair paths in M, Π_s by Δ_s, Π'_s by Δ'_s, always choose $\pi \in \Delta_s$ and $\pi' \in \Delta'_s$ for some s and argue with Lemma 3.5 instead of Lemma 3.1.

In the b) parts of the proof of Proposition 3.2 we can argue with Lemma 3.6. If for all fair paths $\pi = s_1 r \ldots$ starting at s_1 follows that $r \neq s_2$ then any formula starting with the universal path quantifier which talks about fair paths starting at s_2 is trivially true since there are no such paths.

■

Of course, the application of T can be iterated.

Definition 3.1 *A sequence of finite models*
$(S_0, R_0, L_0), (S_1, R_1, L_1), \ldots (S_n, R_n, L_n)$ *where*

$$\forall 0 \leq i < n :$$
$$(\exists s_1, s_2 \in S_i : (s_1 R_i s_2 \wedge \neg s_2 R_i s_2 \wedge (L_i(s_1) \cap W = L_i(s_2) \cap W))$$
$$\wedge (S_{i+1}, R_{i+1}, L_{i+1}) = T(s_1, s_2, (S_i, R_i, L_i)))$$

i.e. a subsequent model is obtained by an application of T at two states in the predecessor model that fulfill condition (1), is called a reduction sequence.

Clearly, the reduction sequence preserves the truth of FRCTL formulae.

Theorem 3.3 *Let (S_0, R_0, L_0) be the first, and (S_n, R_n, L_n) be the last model in a reduction sequence. Then $\forall \phi \in FRCTL : \forall s \in S_n : s \models_{(S_0,R_0,L_0)} \phi \Leftrightarrow s \models_{(S_n,R_n,L_n)} \phi$.*

Proof: Directly by subsequent application of Theorem 3.2.

■

Note that we can determine the truth of a specification formula only for states which are not cut from the original model. Therefore, if we want to determine the truth of a formula with respect to a certain state we have to make sure that this state does not serve as an s_2 in an application of T.

3.5 The algorithm

In this subsection we present a reduction algorithm based on the transformation T.

3.5.1 Description of algorithm

The algorithm is based on one breadth-first search through the Kripke model. The algorithm together with the meaning of the used identifiers is shown in Figure 9. The algorithm is described in the form of relations. This makes it easily implementable by OBDDs, the data structure on which the fastest model checkers are built.

In the beginning, the set of reached states (G^0) is the set of initial states (I), the transition relation H^i representing the reduced transition relation is empty, and P^0, which contains the frontier transitions, contains transitions from I to their successor states. In an incremental way, the reduced model (H^i) is built, new states are reached (G^i) and new frontier transitions (P^i) are considered.

In each iteration, the frontier transitions are considered (P^i). The algorithm differentiates between three different types of such frontier transitions P^i: PA^i, M^i and N^i. Transitions to states (PA^i) which have already been reached are simply added to H^{i+1}. This ensures that the algorithm does not consider already reached states again. Transitions between states which fulfill condition (1) and the next state of which do not have any other ingoing arcs (M^i) are modified according to Figure 3. The modified transitions are added to the new frontier transitions P^{i+1}. The reason why we have to make sure that there are no other ingoing arcs is that loops in the original model have to be maintained (to preserve the truth of the FRCTL formula). The remaining frontier transitions N^i are added to H^{i+1} and follow-up transitions are added to P^{i+1}. Thus, P^{i+1} contains the next slice of frontier transitions.

The algorithm stops when no new states are reached. In this case, H^i will contain the reduced model and G^i the reachable states of the original model. The reachable states of the reduced model could also be calculated without special overhead at the same time.

3.5.2 Correctness of the algorithm

In the following, non indexed names correspond to the names after the last iteration.

Theorem 3.4 *The algorithm terminates and constructs a reduced model (S_H, H, L_H) from the original model (S, R, L) and preserves the truth of the specification formulae in FRCTL, i.e.*

$$\forall \phi \in FRCTL : \forall s \in S_H : s \models_{(S,R,L)} \phi \Leftrightarrow s \models_{(S_H,H,L_H)} \phi$$

where S_H are the states reachable by transitions in H, which are a subset of the states reachable by transitions in R, and L_H is L restricted to these states.

```
$i = 0$
$G^0 = I$
$P^0 = \{(s,s') | sRs' \wedge s \in I\}$
$H^0 = \emptyset$
repeat
        $F^i = \{s' | \exists s : sP^is'\}$
        $B^i = F^i \setminus G^i$
        $PB^i = \{(s,s') | sP^is' \wedge s' \in B^i\}$
        $PA^i = P^i \setminus PB^i$
        $M^i = \{(s,s') | sPB^is' \wedge \neg s'Rs' \wedge (L(s) \cap W = L(s') \cap W) \wedge$
                        $(\neg \exists s'' \in \overline{G^i} : s'' \neq s \wedge s''Rs')\}$
        $N^i = PB^i \setminus M^i$
        $G^{i+1} = G^i \cup B^i$
        $H^{i+1} = H^i \cup PA^i \cup N^i$
        $P^{i+1} = \{(s,s') | \exists s'' : sM^is''Rs'\} \cup \{(s,s') | \exists s'' : s''N^isRs'\}$
        $i = i + 1$
until $B^i = \emptyset$
```

R	transition relation of original model
I	initial states
G	states reached so far (in the original model)
F	frontier states
B	frontier states not in G (not yet reached)
H	current transition relation of reduced model
P	frontier transitions
PA	frontier transitions which lead back to already reached states
PB	frontier transitions to new states
M	frontier transitions between two different states which fulfill condition (1) and the next state of which do not have other ingoing arcs
N	transitions in PB but not in M

Figure 9: The reduction algorithm together with an explanation of the identifiers

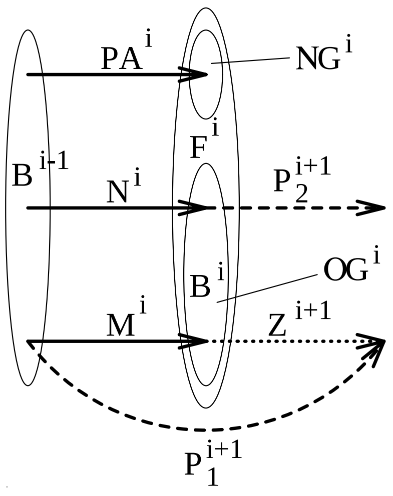

Figure 10: Successors (F^i) of newly reached states (B^{i-1}) are considered in each iteration of the algorithm

Proof:

Part I: termination

The number of reached states, i.e. the cardinality of G, strictly increases except for the last iteration when $B = \emptyset$. Since the number of states in the model is finite G is limited and B, the set of newly reached states will be empty after a finite number of iterations. As a consequence, the algorithm terminates.

Part II: truth preservation

For ease of notation, we write $R \models \phi$ and mean $s \models_{(S,R,L)} \phi$ for any state $s \in S_H$ since S is understood from R and L is only restricted when R is modified.

For the states s reachable in (S, R, L) all arcs leading to or starting from s are considered and directly added to H or added after modification according to the transformation T.

Let $P_1^i = \{(s, s')|\exists s'' : sM^i s'' Rs'\}$ and $P_2^i = \{(s, s')|\exists s'' : s'' N^i sRs'\}$.

We show that the following invariant holds after each iteration:
$H^{i+1} \cup P_1^{i+1} \cup (R \cap (S \setminus G^i \times S)) \models \phi \Leftrightarrow H^i \cup P_1^i \cup (R \cap (S \setminus G^{i-1} \times S)) \models \phi$.

Before the first iteration we have $H^0 \cup P_1^0 \cup R \cap (S \setminus G^{-1} \times S) = R$ if we take $G^{-1} = \emptyset$, $P_1^0 = \emptyset$ and $H^0 = \emptyset$.

In each iteration, the successors (F^i) of newly reached states (B^{i-1}) are considered (Figure 10). $B^{-1} = I$ in the first iteration. All arcs to these successors from states already reached are contained in P^i. This is clear before the first iteration. In each iteration P^{i+1} is correctly updated. Transitions from B^i are either added directly via P_2^{i+1} or the transformation T is applied to them via P_1^{i+1} (states in B^i serve as states s_2). P^{i+1} therefore contains all arcs to successors of newly reached states from states already reached.

We now describe what happens to the arcs (marked bold in Figure 10) to these successors of newly reached states. These arcs were either contained in $Z^i = \{(s, s')|\exists s'' \colon s'' M^i sRs'\}$ or P_2^i of the previous iteration. These arcs are cut from $R \cap (S \setminus G^{i-1} \times S)$ because $R \cap (S \setminus G^i \times S) = (R \cap (S \setminus G^{i-1} \times S)) \setminus B^{i-1} \times S$. At the same time, the P^i arcs, in which the P_2^i arcs are contained, are directly added to H^i via PA^i or N^i or modified according to the transformation T in P_1^{i+1} (on the left-hand side of the union of P^{i+1}). In this calculation several arcs are modified simultaneously. The result of the simultaneous modifications is the same as when these modifications are performed in sequence since any s_1 does never serve as an s_2 since all s_2 are newly reached states and all s_1 are 'old' states. The latter follows from $M^i \subseteq PB^i$. In spite of this transformation in P_1^{i+1} the transitions in Z^{i+1} are still in R. However, these transitions do not play any role since the starting points of these arcs are not reachable any longer in $R \cap (S \setminus G^i \times S)$. This is the reason why Z^i did not need to be added to the new transition relation.

Using Theorem 3.3 we can therefore conclude that $H^{i+1} \cup P_1^{i+1} \cup R \cap (S \setminus G^i \times S) \models \phi \Leftrightarrow H^i \cup P_1^i \cup R \cap (S \setminus G^{i-1} \times S) \models \phi$ after each iteration.

In the last iteration, B^i is empty, and thus also P_1^{i+1}, consequently $H^{i+1} \cup P_1^{i+1} = H^{i+1}$. Since G^i contains all reachable states in (S, R, L) (the frontier states are added to G^i in each iteration), $R \cap (S \setminus G^i \times S)$ contains only arcs from non-reachable states. Since these non-reachable states do not have any effect on the truth of ϕ for a given $s \in S_H$ we can conclude:

$$\begin{gathered}
\Rightarrow \forall s \in S_H : s \models_{(S_H, H, L_H)} \phi \Leftrightarrow \\
s \models_{(S_H, H \cup R \cap (S \setminus G \times S), L_H)} \phi \Leftrightarrow \\
\ldots \Leftrightarrow \\
s \models_{(S,R,L)} \phi
\end{gathered}$$

■

3.5.3 Better reduction by special treating of fairness

An even better reduction can be obtained if the formula is out of FRCTL and the fairness constraints consist only of propositional logic. In this case, the variables in the fairness constraints are not added to W. Instead, we add the following constraint to condition (1): if s_2 fulfills any of the fairness constraints then s_1 fulfills this fairness constraint. In the algorithm we substitute $M^i = \ldots$ by

$$\begin{aligned}
M^i = \{(s, s') | sPBs' \wedge \neg s'Rs' \wedge (L(s) \cap W = L(s') \cap W) \wedge \\
(\neg \exists s'' \in \overline{G^i} : s'' \neq s \wedge s''Rs') \wedge (\forall 0 \leq i \leq n : s' \models h_i \rightarrow s \models h_i)\}
\end{aligned}$$

In this way, the transformation will leave fair paths fair. At the same time a better reduction is possible since W is smaller.

4 Evaluation

It is difficult to justify why the overall performance of our reduction algorithm with subsequent model checking of the reduced model is better than just model checking the original model if OBDDs are used to represent the transition relation of the model. We therefore analyze the overall performance for the case that hash tables are used to represent the transitions between states. Furthermore, we substantiate our claim by a practical example.

4.1 Analysis of the overall performance

In the case of hash tables, CTL model checking has time complexity $O(|f| \cdot (|S|+|R|)$ where $|f|$ represents the length of the formula f [CES86]. Our reduction algorithm has time complexity $O(|S|+|R|)$, i.e., it has linear time complexity in the number of reachable states and transitions (due to just one breadth-first search through the reachable states of the model). Thus, checking an RCTL formula on the reduced model together with prior reduction has time complexity $O(|S|+|R|+|f| \cdot (|S'|+|R'|))$ where (S, R, L) is the original Kripke model and (S', R', L') the reduced model.

There are three reasons why model checking with prior reduction by our reduction algorithm will in general be faster:

- In general, $|W| << |V|$, i.e., there are only few (visible) components (compared to the total number of components) about which the specification talks, and as a consequence the reduction will be considerable, i.e., S', R' are small compared to S and R.
- It is often the case that $|f| \geq 2$ and thus the reachable states of the model have to be inspected several times by the model checking algorithm.
- In most cases reachability analysis is advantageous. The time needed for the reachability analysis would need to be added to the time complexity of the model checking. Our reduction algorithm is combined with reachability analysis and thus eliminates the need for an extra reachability analysis and does not contribute any additional cost to the time complexity above.

If a model is checked for an RCTL formula with fairness constraints, the gain will be even bigger since checking CTL formulae with fairness constraints as in [CGL93] has complexity $O(|f|^2 \cdot (|S|+|R|))$.

4.2 Example: the alternating bit protocol

Based on David Long's OBDD package a μ-calculus evaluator [Bie95] has been implemented. Various input languages (e.g. state charts, process algebra) will later be on top of this μ-calculus evaluator[†]. This general approach allows easy extensions to other specification languages so that no special model checker has to be written for every specification language, optimizations on the μ-calculus level and easy experimentation to speed up model checking. Our system also allows the experimentation with different representations for transition relations, not just OBDDs.

Our reduction algorithm has been implemented on the level of the μ-calculus evaluator, also using Long's OBDD package. We applied our reduction algorithm to an interleaving model of the alternating bit protocol as it is described in [BK95] in Sections 4.1 and 4.2. The original model has 22 *reachable* states (out of 1728 possible states) and 26 transitions among them. The formula to be checked is an RCTL formula (stating that all messages are received in the correct order) with fairness constraints.

After reduction with our algorithm only 7 states and 10 transitions remained. Although the model is extremely small our reduction algorithm with subsequent

[†]This is done in cooperation with Forschungszentrum Informatik, Karlsruhe.

model checking needs almost only half of the time (53 %) compared to model checking the original model. This result is very promising since we can expect much better reductions on large models.

When the compilers from various description languages to our μ-calculus evaluator have been implemented larger test examples will be available. This will allow us to draw conclusions about when our reduction algorithm should be applied and to which extent the reduction speeds up model checking.

5 Conclusions and future work

Formula-dependent model reduction algorithms can achieve much better reductions than reduction algorithms which do not consider the properties to be verified. The formula-dependent reduction algorithms presented so far [ASSSV94], [DGG93], however, are computationally expensive. In this paper we have presented a formula-dependent reduction algorithm which is computationally efficient because it needs only one pass through the reachable states of the model but allows nevertheless that long subpaths are cut from the original model if they do not affect the 'visible' variables. Although no complete reduction with respect to the equivalence relation induced by the formula is guaranteed, it is reasonable to expect that the reduction achieved by our algorithm will be close to it if the components work synchronously. Thus our reduction algorithm will often reduce both space and time needed for model checking real systems.

Our reduction algorithm requires the prior construction of the global transition relation of the system to be model checked. This is not necessary if the parallel components of a distributed system do not work synchronously. In this case, the reduction algorithm can be modified so that it reduces the components before the global transition relation is constructed. We can already cut out transitions which are neither visible actions nor actions participating in a communication in the automata to be composed. So we subsequently reduce and compose automata. In this way, at least the space complexity due to the space needed by the global transition relation of the original model can be reduced.

After application of our reduction algorithm the number of variables needed to represent the reachable states might be much smaller than before. The reduced transition relation could therefore be mapped onto a smaller representation with fewer variables. That only the variables in W might not be sufficient to represent the states of the fully reduced model becomes clear from Figure 11 where both s_1 and s_2 are in the reduced model although they have the same valuations. The number of variables needed can thus be determined from the number of states which can not be distinguished by variables in W (s_1 and s_2 in Figure 11).

When checking a formula on the reduced model there might be errors in the implementation. The counterexamples will contain only the variables in the reduced model. This might have disadvantages when searching for errors in the original implementation. On the other hand, however, counterexamples which contain only important information (visible variables) might be easier to understand and thus help to find errors in the original model. If the former is the case, one could add variables considered important for error finding to the variables in the specification and doing the reduction on these variables to produce more understandable counterexamples.

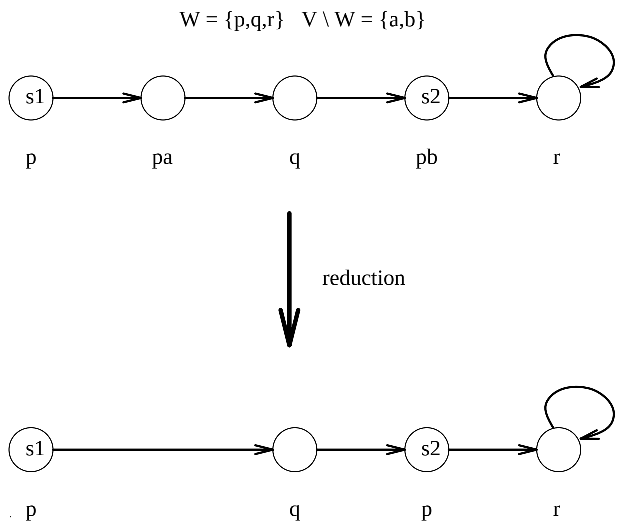

Figure 11: Variables needed to represent the reduced model.

Acknowledgements

We are grateful to A. Biere and A. Zundel for their careful reading and useful comments on earlier drafts of this paper.

References

[ASSSV94] A. Aziz, T. R. Shiple, V. Singhal, and A. L. Sangiovanni-Vincentelli. Formula-dependent equivalence for compositional CTL model checking. In D. L. Dill, editor, *Computer Aided Verification*, volume 818 of *LNCS*, pages 324 – 337. Springer, 1994.

[BdS92] A. Bouali and R. de Simone. Symbolic bisimulation minimization. In *Computer Aided Verification*, volume 663 of *LNCS*, 1992.

[BFH$^+$92] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18(3):247 – 271, 1992.

[Bie95] A. Biere. μcke – an evaluator of μ-calculus formulae. Technical report, University of Karlsruhe, 1995.

[BK95] A. Biere and A. Kick. A case study on different modelling approaches based on model checking - verifying numerous versions of the alternating bit protocol with SMV. Technical Report 5, University of Karlsruhe, 1995.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[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, April 1986.

[CGL93] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In de Bakker, editor, *A Decade of Concurrency, REX School/Symposium*, volume 803 of *LNCS*, pages 124 – 175. Springer, 1993.

[DGG93] D. Dams, O. Grumberg, and R. Gerth. Generation of reduced models for checking fragments of CTL. In C. Courcoubetis, editor, *Computer Aided Verification*, volume 697 of *LNCS*, pages 479 – 490. Springer, 1993.

[GW91] P. Godefroid and P. Wolper. A partial approach to model checking. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science (LICS)*, 1991.

[Lon93] D. E. Long. *Model Checking, Abstraction, and Compositional Verification.* PhD thesis, Carnegie Mellon University, July 1993.

[NS94] V. G. Naik and A. P. Sistla. Modeling and verification of a real life protocol using symbolic model checking. In D. L. Dill, editor, *Computer Aided Verification*, volume 818 of *LNCS*, pages 194 – 206. Springer, 1994.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In C. Courcoubetis, editor, *Computer Aided Verification*, volume 697 of *LNCS*, pages 409 – 423, 1993.

[Val90] A. Valmari. A stubborn attack on the state explosion. In *Computer Aided Verification*, LNCS, 1990.