Tableau Recycling

Angelika Mader *
Technische Universität München
Arcisstr. 21
W - 8000 München 2
Germany
e-mail: mader@informatik.tu-muenchen.de

Abstract

In this paper we improve a model checking algorithm based on the tableau method of Stirling and Walker. The algorithm proves whether a property expressed in the modal mu-calculus holds for a state in a finite transition system. It makes subsequent use of subtableaux which were calculated earlier in the proof run. These subtableaux are reduced to expressions. Examples show that both size of tableaux and execution time of the algorithm are reduced.

1 Introduction

The modal mu-calculus is an active area of research. It stands in the tradition of Hoare logic, Dynamic logic, Process logic, and linear and branching time logics [S]. Model checking in the modal mu-calculus plays a part in verification of parallel processes with both finite [CS1] and infinite state spaces [BS], and finds application in preorder models [CS2] and in Petri nets [B].
The main approaches are symbolic model checking [BC] [EFT], model checkers based on the fixpoint induction principle [EL], and tableau based model checkers as in [SW] [C]. An advantage of the latter is its ability to deal also with infinite state spaces. In comparison to the approximation techniques they work locally, i.e. they do not determine the set of all states satisfying a property, but prove a modal formula only for one state. Unfortunately the attractiveness of tableau methods suffers by their complexity. A main reason for this is that these model checkers do not make subsequent use of subresults.
This paper presents a method whereby a tableau based model checker for the full modal mu-calculus can recycle subtableaux, which have been calculated earlier in the model checking algorithm. An implementation of these ideas has confirmed an impressive improvement in execution speed in a variety of examples.
The following section introduces briefly the modal mu-calculus and its semantics. In section 3 the underlying standard tableau model checker is described. The motivating

*supported by Siemens AG, Corporate Research and Development

ideas of tableau recycling and the necessary notions are contained in section 4. Sections 5 and 6 present the algorithm and the proofs of its correctness and completeness. Some surprising examples can be found in section 7. Section 8 concludes this paper.

2 The Modal Mu-Calculus

This section gives a brief introduction to the modal mu-calculus. For more details see [S].

The syntax of the modal mu-calculus is defined with respect to a set $\mathcal{Q}$ of atomic propositions including *true* and $false$, a finite set $\mathcal{L}$ of action labels and a denumerable set $\mathcal{Z}$ of propositional variables. A formula of the modal mu-calculus is an expression of the form:

$$A ::= Z \mid Q \mid \neg A \mid A \wedge A \mid [a]A \mid \nu Z.A$$

where $Z \in \mathcal{Z}$, $Q \in \mathcal{Q}$ and $a \in \mathcal{L}$, and where in $\nu Z.A$ every free occurence of Z in A falls under an even number of negations. The standard conventions for the derived operators are:

$A_1 \vee A_2 := \neg(\neg A_1 \wedge \neg A_2)$
$\langle a \rangle A \quad := \neg[a]\neg A$
$\mu Z.A \quad := \neg\nu Z.\neg A[\neg Z/Z]$.

Formulae of the modal mu-calculus with the set $\mathcal{L}$ of action labels are interpreted relative to a *labelled transition system* $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{L}\})$, where $\mathcal{S}$ is a finite set of states and $\xrightarrow{a} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ for every $a \in \mathcal{L}$ a binary relation on states. A ***valuation function*** $\mathcal{V}$ assigns to every atomic proposition Q in $\mathcal{Q}$ (and propositional variable Z in $\mathcal{Z}$) a set of states $\mathcal{V}(Q) \subseteq \mathcal{S}$ ($\mathcal{V}(Z) \subseteq \mathcal{S}$) meaning that the proposition Q (variable Z) holds for every state in $\mathcal{V}(Q)$ ($\mathcal{V}(Z)$). The pair $\mathcal{T}$ and $\mathcal{V}$ is called a ***model*** of the mu-calculus. The semantics of each mu-calculus formula A is the set of states $\|A\|^{\mathcal{T}}_{\mathcal{V}}$ defined inductively as follows:

$\|Z\|^{\mathcal{T}}_{\mathcal{V}} = \mathcal{V}(Z)$
$\|Q\|^{\mathcal{T}}_{\mathcal{V}} = \mathcal{V}(Q)$
$\|\neg A\|^{\mathcal{T}}_{\mathcal{V}} = \mathrm{S} - \|A\|^{\mathcal{T}}_{\mathcal{V}}$
$\|A_1 \wedge A_2\|^{\mathcal{T}}_{\mathcal{V}} = \|A_1\|^{\mathcal{T}}_{\mathcal{V}} \cap \|A_2\|^{\mathcal{T}}_{\mathcal{V}}$
$\|[a]A\|^{\mathcal{T}}_{\mathcal{V}} = \{s \in \mathcal{S} \mid \forall s'.\ if\ s \xrightarrow{a} s'\ \ then\ \ s' \in \|A\|^{\mathcal{T}}_{\mathcal{V}}\}$
$\|\nu Z.A\|^{\mathcal{T}}_{\mathcal{V}} = \bigcup\{\mathcal{S}' \subseteq \mathcal{S} \mid \mathcal{S}' \subseteq \|A\|^{\mathcal{T}}_{\mathcal{V}[\mathcal{S}'/Z]}\}$

3 A Standard Tableau Model Checker

This section sketches a standard tableau method based on the model checker of Stirling & Walker [SW]. The notation used here is a mixture of the notations of Cleaveland [C] and Stirling & Walker [SW]. Some additional notions are necessary.
The modal mu-calculus is extended by a set of propositional constant symbols. Let

$U, U_1 \ldots$ range over these symbols. A *definition* is a declaration $U = A$, where U is a constant symbol and A a formula of the modal mu-calculus which may contain constant symbols. A *definition list* Δ consists of a sequence of definitions $(U_1 = A_1) \ldots (U_n = A_n)$. It fulfills the requirement that every U_j appearing in A_i is defined before U_i, i.e. $j \leq i$, and that $U_i \neq U_j$ for $i \neq j$. For Δ being such a definition list the function $\Delta(U_i) = A_i$ is declared. A *hypothesis* is an expression of the form $s \in U$, where s is a state of a transition system and U a constant symbol. Hypotheses are collected in a *hypothesis set* H. A *sequent* $H \vdash_\Delta s \in A$ expresses that the formula A is valid at the state s with respect to the hypotheses of H. We drop empty Δs and Hs.

The model checker here is tableau based. This corresponds to a top-down proof method, starting with the intended conclusion and reducing it stepwise to (atomic) premisses. The rules for a tableau method are inverse to the usual rules of natural deduction. Here we take the conclusions and premisses to be sequents. The root sequent $\vdash s \in A$ contains the state s and the modal property A, which we want to prove for this state. The root sequent has an empty hypothesis set and an empty definition list. The rules of the tableau system are:

$$1)\frac{H \vdash_\Delta s \in \neg\neg A}{H \vdash_\Delta s \in A} \qquad 2)\frac{H \vdash_\Delta s \in A \wedge B}{H \vdash_\Delta s \in A \quad H \vdash_\Delta s \in B}$$

$$3)\frac{H \vdash_\Delta s \in \neg(A \wedge B)}{H \vdash_\Delta s \in \neg A} \qquad 4)\frac{H \vdash_\Delta s \in \neg(A \wedge B)}{H \vdash_\Delta s \in \neg B}$$

$$5)\frac{H \vdash_\Delta s \in [a]A}{H \vdash_\Delta s_1 \in A \ldots H \vdash_\Delta s_n \in A} \quad \{s_1, \ldots, s_n\} = \{s' \mid s \xrightarrow{a} s'\}$$

$$6)\frac{H \vdash_\Delta s \in \neg[a]A}{H \vdash_\Delta s' \in \neg A} \quad s \xrightarrow{a} s'$$

$$7)\frac{H \vdash_\Delta s \in \nu Z.A}{H \vdash_{\Delta'} s \in U} \quad \Delta' = \Delta \cdot (U = \nu Z.A)$$

$$8)\frac{H \vdash_\Delta s \in \neg\nu Z.A}{H \vdash_{\Delta'} s \in U} \quad \Delta' = \Delta \cdot (U = \neg\nu Z.A)$$

$$9)\frac{H \vdash_\Delta s \in U}{H' \vdash_\Delta s \in A[Z := U]} \quad (s \in U) \notin H,\ \Delta(U) = \nu Z.A,\ H' = H \cup \{s \in U\}$$

$$10)\frac{H \vdash_\Delta s \in U}{H' \vdash_\Delta s \in \neg A[Z := \neg U]} \quad (s \in U) \notin H,\ \Delta(U) = \neg\nu Z.A,\ H' = H \cup \{s \in U\}$$

A *proof tree* is constructed by applying the rules to the root sequent, and then to its successors etc. The proof tree is *maximal* if no rule is applicable to a leaf sequent. Such a maximal proof tree is called a *tableau*. A tableau is *successful*, if all its leaves are successful. A leaf sequent $H \vdash_\Delta s \in B$ is successful, if it satisfies one of the properties (i)-(iv):

(i) $B = Q$ and $s \in \mathcal{V}(Q)$

(ii) $B = \neg Q$ and $s \notin \mathcal{V}(Q)$

(iii) $B = [a]C$

(iv) $B = U$ and $\Delta(U) = \nu Z.C$ and $(s \in U) \in H$

(i') $B = Q$ and $s \notin \mathcal{V}(Q)$

(ii') $B = \neg Q$ and $s \in \mathcal{V}(Q)$

(iii') $B = \langle a \rangle C$

(iv') $B = U$ and $\Delta(U) = \neg \nu Z.C$ and $(s \in U) \in H$

If one of the dual forms (i')-(iv') of these requirements holds for a leaf sequent, then it is not successful. In an *unsuccessful* tableau there is at least one leaf, which is not successful.

For later considerations the definition of a *computation tree* is also neccessary. A *model checker algorithm* based on the tableau rules builds a tree starting with $\vdash s \in A$ as root sequent and applying nondeterministically the rules. When a leaf fails the algorithm has to use backtracking techniques to try other paths. It will build up this tree until it is sure that a sequent has a successful subtableau or not. In the first case the tree includes a successful tableau, in the second case it is not necessary that the tree contains any maximal proof tree. We call a tree created by such a model checker algorithm a *computation tree*, if it sufficient to decide, whether there exists a successful tableau or not.

Note that such a computation tree determines an "and-or-tree", if the successful leaves are identified with *true*, the unsuccessful ones with *false*, the nondeterministic branching as disjunction and the deterministic branching as conjunction. It can be evaluated to *true*, iff the computation tree contains a successful tableau.

For simplicity of notation in this paper the definition lists Δ, Δ' are considered to be *global* for all different subtrees of one computation tree, i.e. there are no two different definition lists (in two different subtrees) Δ_1, Δ_2 and for any U, $\Delta_1(U) \neq \Delta_2(U)$.

4 Tableau Recycling

This section starts by describing a *basic source of inefficiency* in the standard model checker. Then it considers *how the efficiency might be improved*, and finally *means for an algorithm* are presented that gives a significant gain in efficiency.

The basic problem with the standard model checker is that it does not store any intermediate results (subresults). It is possible that it proves the same formula for the same state arbitrarily often, as the following example will show:

$\vdash s_1 \in \Phi$
$\Phi = \nu X.[a]\langle b \rangle X$

informal meaning: "every a-successor has a b-successor for which this property holds,recursively."

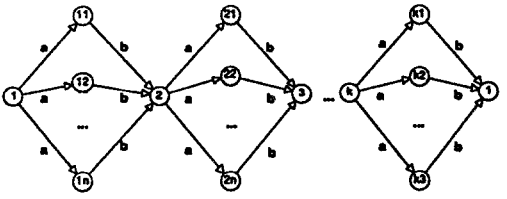


Fig. 1. model with tableau of exponentional size

Here the formula Φ is proved n times for the state s_2, n^2 times for the state s_3, ..., n^{k-1} times for the state s_k. The number of nodes in the computation tree is exponential with respect to the length k of this transition system.
Moreover, a look at computation trees shows that in many cases, different subtableaux of a state and a formula are very *similar*.

The most obvious way to improve efficiency would be just to store the information *"the sequent $H \vdash s \in \Phi$ has a successful (or no successful) subtableau"* , and use it whenever the same state and formula appear again in the computation tree.
Unfortunately this idea is too simple:

- The first obstacle is that the constants used in different subtableaus have different names. It therefore does not happen that exactly the same sequent appears twice. A notion of equivalence will help to solve this problem.

- Secondly the hypothesis sets can differ, even when formulae and states are identical. It is obvious that the shapes of the computation subtrees differ accordingly, since they depend on the hypothesis sets. Therefore in order to recycle a computation subtree of a similar sequent with even a slightly different hypothesis set, one has to store also the shape of the computation subtree. It turns out that the shape of such a subtree can be reduced to an expression which is sufficient for the derivation of all useful information.

A cornerstone of the tableau recycling algorithm is a notion of equivalence, which allows different sequents in a computation tree to be compared. The basis for this is the definition of equivalent constants, which have different names but identify syntactically the same formula. In the following let for $Z_1, \ldots, Z_n$ being the free variables in Φ, denote $\Phi(U_{j_1}, \ldots U_{j_n}) = \Phi[U_{j_1}/Z_1, \ldots U_{j_n}/Z_n]$, meaning that every occurence of Z_i in Φ is substituted by U_{j_i}.

DEFINITION 1 (Equivalence)

(1) A formula $\Phi(U_{j_1}, \ldots U_{j_n})$ *is* equivalent *to the formula* $\Phi(U_{k_1}, \ldots U_{k_n})$, *denoted* $\Phi(U_{j_1}, \ldots U_{j_n}) \sim \Phi(U_{k_1}, \ldots U_{k_n})$, *iff for* $1 \leq i \leq n$ *all constants* $U_{j_i} \sim U_{k_i}$.

(2) A constant U_1 *is* equivalent *to the constant* U_2, *denoted* $U_1 \sim U_2$, *iff* $U_1 = U_2$ *or* $\Delta(U_1) \sim \Delta(U_2)$.

Note that this equivalence is essentially alpha-conversion with respect to constants.
The main insight now is that equivalent sequents can have identical computation subtrees.
We will now show how the shape of a computation tree can be reduced to an expression.

DEFINITION 2 (Hypothesis Tree)
A hypothesis tree is an expression of the form:

$$HT ::= true \mid false \mid unknown \mid Y \mid Y; HT \mid \bigvee_{i \in I} HT_i \mid \bigwedge_{i \in I} HT_i$$

where I is a finite index set and Y is a hypothesis, e.g. $s \in U$.

If the termination behavior of all paths of the computation tree is known, it is determined whether there is a successful subtableau or not. Some of the paths terminate with the rules (i)-(iii) and their dual forms. This kind of termination is independent of the hypothesis sets. The other paths terminate with rule (iv) and its dual form. In this case the kind of termination depends on the hypothesis set and whether for the terminal sequent $H' \vdash_\Delta s \in U$ the constant U stands for a maximal or negated maximal (minimal) fixpoint formula.
The reduction of a computation tree to a hypothesis tree reflects this idea:
transform the computation tree to an "and-or tree", but leave all hypotheses in it.

DEFINITION 3 (Reduction to a Hypothesis Tree)
Reduce a computation tree to a hypothesis tree in the following way:

- *Replace every leaf terminating with rules (i)-(iii) or their dual forms by true or false respectively.*
- *Replace every leaf terminating with a hypothesis which is not contained in the root hypothesis set by true if the leaf is successful or false otherwise.*
- *Replace every nondeterministic branching by a disjunction. If not all of the nondeterministic rules were applied, extend this disjunction by a leaf unknown. The number of all possible nondeterministic rules is called the arity of the disjunction.*
- *Replace every deterministic branching by a conjunction. If not all of the deterministic rules were applied, extend this conjunction by a leaf unknown. The number of all possible deterministic rules is called the arity of the conjunction.*
- *Drop all sequents which are not of the form $H \vdash_\Delta s \in U$, and drop all hypothesis sets.*

DEFINITION 4 (Evaluation of a Hypothesis Tree)
The function eval evaluates a hypothesis tree HT together with a hypothesis set H to $eval(HT, H) \in \{true, false, unknown\}$ in the following way:

- *substitute every hypothesis $s \in U$ in HT which is contained in H by true if $\Delta(U) = \nu X.A$, else by false .*
- *substitute every hypothesis which is not contained in H by unknown.*
- *Evaluate this tree with the following rules which are extended in the obvious way for indexed conjunction and disjunction:*

$\vee$	*true*	*false*	*unknown*
true	*true*	*true*	*true*
false	*true*	*false*	*unknown*
unknown	*true*	*unknown*	*unknown*

$\wedge$	*true*	*false*	*unknown*
true	*true*	*false*	*unknown*
false	*false*	*false*	*false*
unknown	*unknown*	*false*	*unknown*

The operator ";" is treated as follows:

$$eval(\ unknown\ ;\ HT\ ,\ H\ ) = eval(HT\ ,\ H)$$
$$eval(\ true\ ;\ HT\ ,\ H\ ) = true$$
$$eval(\ false\ ;\ HT\ ,\ H\ ) = false$$

The following definition reflects the idea that hypothesis trees with equivalent roots particulary have a common structure. Combining two hypothesis trees then means that the common structure is identified and extended by both non common structures parts.

DEFINITION 5 (Combination of Hypothesis Trees)
Consider two sequents $H_1 \vdash_\Delta s \in U_1$ and $H_2 \vdash_\Delta s \in U_2$, $U_1 \sim U_2$. τ_1 is the hypothesis tree constructed from a computation tree of the first sequent, τ_2 from the second one. Let $\tau_3 = \tau_1 \circ \tau_2$ be the combination of τ_1 and τ_2 such that

- *The root of τ_3 is the root of τ_1.*
- *If $t \in U_1'$ is a successor of $s \in U_1$ with the subtree τ_1' and $t \in U_2'$ is a successor of $s \in U_2$ with the subtree τ_2' such that $U_1' \sim U_2'$, then $\tau_1' \circ \tau_2'$ is a direct subtree of the root of τ_3.*
- *If $t \in U_1'$ is a successor of $s \in U_1$ with the subtree τ_1' and there is no successor $t \in U_2'$ of $s \in U_2$ such that $U_1' \sim U_2'$, then τ_1' is a direct subtree of the root of τ_3. (symmetrically for $t \in U_2'$)*
- *If the successors of $s \in U_1$ and $s \in U_2$ are disjunctions (neccessarily with the same arity) then the successor of the root of τ_3 is also a disjunction with the same arity and those branches are combined pairwise which correspond to the same rule applied at this place in the computation tree. Branches of one disjunction which have no corresponding branch of the other disjunction appear directly in the combined disjunction. If the arity of the disjunction and the number of branches are equal all leaves unknown in this disjunction are dropped. (analogously for conjunction)*

5 The Tableau Recycling Model Checker

The standard tableau model checker is extended by a set of hypothesis trees $\mathcal{HT}$. In the beginning $\mathcal{HT}$ is initialized with the empty set.

Every time, when a computation tree is built up for a sequent $H \vdash_\Delta s \in U$, the hypothesis tree τ is derived and inserted in $\mathcal{HT}$. If there already exists a hypothesis tree τ_1 in $\mathcal{HT}$ with a equivalent root $s \in U'$, $U \sim U'$, the combined hypothesis tree $\tau \circ \tau_1$ is added to $\mathcal{HT}$ replacing τ_1.
The tableau rules 9) and 10) are extended by a further requirement:

9')

$$\frac{H \vdash_\Delta s \in U}{H' \vdash_\Delta s \in A[Z := U]} \; (s \in U) \notin H, \;\; \Delta(U) = \nu Z.A, \;\; H' = H \cup \{s \in U\}$$

and there is no $\tau \in \mathcal{HT}$ such that the root $s \in U'$, $U \sim U'$ and $eval(\tau, H) \in \{true, false\}$;

10')

$$\frac{H \vdash_\Delta s \in U}{H' \vdash_\Delta s \in \neg A[Z := \neg U]} \; (s \in U) \notin H, \;\; \Delta(U) = \neg \nu Z.A, \;\; H' = H \cup \{s \in U\}$$

and there is no $\tau \in \mathcal{HT}$ such that the root $s \in U'$, $U \sim U'$ and $eval(\tau, H) \in \{true, false\}$;

In addition to the conditions (i)-(iv) stated in section 3 a leaf sequent $H \vdash_\Delta s \in U$ of a tableau constructed by by rules 1)-8), 9') and 10') is also successful if it fulfills the following property:

(v) There exists a $\tau \in \mathcal{HT}$ such that the root $s \in U'$, $U \sim U'$ and $eval(\tau, H) = true$.

The number of elements in the set $\mathcal{HT}$ is bounded: for every state in the transition system $\mathcal{T}$ and every fixpoint operator in the root formula there is at most one hypothesis tree contained in $\mathcal{HT}$.

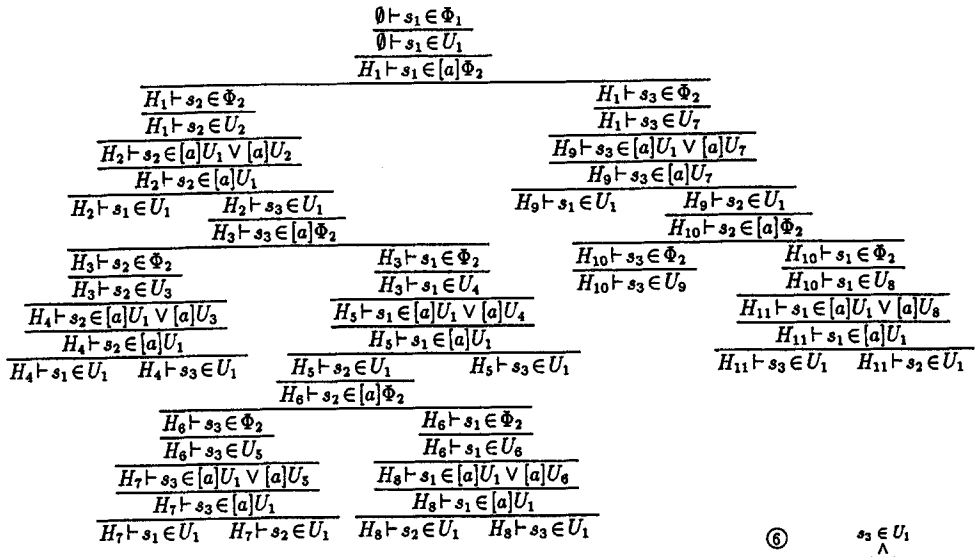
The example on the next page should clarify the algorithm.

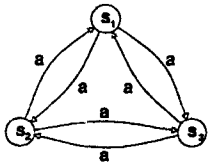
6 Correctness and Completeness

In this section it will be shown that the recycling model checker produces the same results as the model checker from Stirling & Walker [SW]. Then correctness and completeness of the model checker presented here follow from the correctness and completeness proved in [SW].

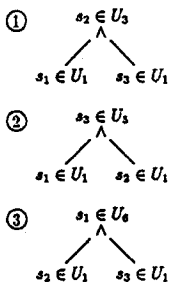
The way of argumentation is as follows: the propositions here are valid for both versions, the standard tableau model checker and the recycling one. First some properties of tableaux and computation trees with similar root sequents are stated. In the remainder it is shown that the evaluation of a hypothesis tree corresponds to the result derived from a computation tree.

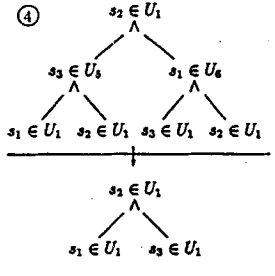
As a first step the notion of equivalence given in definition 1 must be extended to hypothesis sets and sequents.

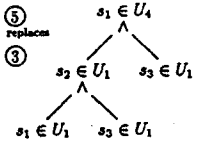


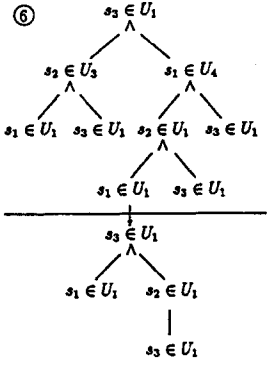


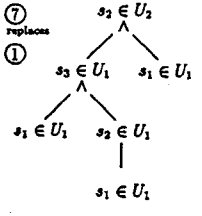
$\Phi_1 \equiv \nu Z.[a]\mu X.[a]Z \vee [a]X$
$\Phi_2 \equiv \mu X.[a]U_1 \vee [a]X$
Δ: $U_1 = \Phi_1$
for $i > 1$: $U_i = \Phi_2$
Δ-list omitted in the tableau;
the hypothesis sets H are
derivable from the rules.











⑧ : ② evaluates to *unknown*
⑨ : ④ evaluates to *unknown*
⑩ : ② evaluates to *true*
⑪ : ⑤ evaluates to *unknown*
⑫ : ⑥ evaluates to *true*

Example 1 : Transitionsystem, Tableau and Hypothesis Trees

DEFINITION 6 (Equivalence)

(3) A hypothesis set H_1 *is* smaller *than a hypotheses set* H_2 *with respect to the equivalent constants* U_1, U_2, *denoted* $H_1 \leq_{U_1,U_2} H_2$, *iff*

- *for every hypothesis* $(s \in U_1) \in H_1$ *the hypothesis* $(s \in U_2)$ *is contained in* H_2, *and*
- *with* $\Delta(U_1) = \Phi(U_{j_1}, \ldots U_{j_n})$ *and* $\Delta(U_2) = \Phi(U_{k_1}, \ldots U_{k_n})$ *for every* $1 \leq i \leq n$ *holds* $H_1 \leq_{U_{j_i},U_{k_i}} H_2$.

(4) Two hypothesis sets H_1 *and* H_2 *are* equivalent *with respect to the equivalent constants* U_1, U_2, *denoted* $H_1 \sim_{U_1,U_2} H_2$,
iff $H_1 \leq_{U_1,U_2} H_2$ *and* $H_2 \leq_{U_1,U_2} H_1$.

(5) Two sequents $H_1 \vdash_\Delta s \in \Phi(U_{j_1}, \ldots U_{j_n})$ *and* $H_2 \vdash_\Delta s \in \Phi(U_{k_1}, \ldots U_{k_n})$ *are* equivalent, *iff for* $1 \leq i \leq n$ $U_{j_i} \sim U_{k_i}$, *and* $H_1 \sim_{U_{j_i},U_{k_i}} H_2$.

PROPOSITION 1 (Equivalent Sequents have the same Subtableaux)
Suppose τ_1 *is a computation tree with the root sequent* $H_1 \vdash_{\Delta_1} s \in \Phi_1$, *and* $H_2 \vdash_{\Delta_2} s \in \Phi_2$ *is an equivalent sequent. Then there exists a computation tree* τ_2 *of* $H_2 \vdash s \in \Phi_2$ *with the same branching structure as* τ_1 *such that every node of* τ_2 *is labelled by a sequent which is equivalent to the sequent of corresponding node of* τ_1.

Proof: by induction in the structure of τ_1

Induction hypothesis: $seq_1 \equiv H_i \vdash_{\Delta_i} t \in \Phi_i$ in τ_1 and $seq_2 \equiv H_j \vdash_{\Delta_j} t \in \Phi_j$ in τ_2 are equivalent sequents.

Base case: the induction hypothesis is true for the root sequents of τ_1 and τ_2 by assumption.

Induction step: argumentation about the applicable rules

- leaf sequents
  If seq_1 is a leaf sequent and fulfills one of the requirements (i) - (iii) or their dual forms, then $\Phi_i = \Phi_j$ and there is no rule applicable to seq_2.
  The more interesting case is if the leaf sequent seq_1 fulfills (iv) or its dual form. Here for $\Phi_i = U_i$ the hypothesis $t \in U_i$ is contained in H_i. Since for $\Phi_j = U_j$ $U_i \sim U_j$ and $H_i \sim_{U_i,U_j} H_j$ the hypothesis $t \in U_j$ must be contained in H_j. Therefore also seq_2 is also a leaf sequent.

- One of the rules 1) - 4) is applicable to the sequent seq_1.
  As equivalent formulae have equivalent structure, the same rule can be applied to seq_2. As equivalent formulae have also equivalent subformulae the successor seq_2' of seq_2 is equivalent to the successor seq_1' of seq_1.

- Rule 5) or 6) is applicable to seq_1.
  $\Phi_i = [a]\Phi_i'$ (or $\Phi_i = \neg[a]\Phi_i'$), hence also $\Phi_j = [a]\Phi_j'$ (or $\Phi_j = \neg[a]\Phi_j'$) and $\Phi_i' \sim \Phi_j'$. All a-successors of the state s depend only on the transition system. Therefore the same rule can be applied to seq_2 and the successor sequents of seq_2 contain the same states, equivalent hypothesis sets and equivalent formulae as the sequent successors of seq_1.

- One of the rules 7) or 8) is applicable to seq_1.
  A new constant U_i' is generated in τ_1. As the same rule must be applicable to seq_2, also a new constant U_j' is generated in τ_2 with $\Delta_i'(U_i') \sim \Delta_j'(U_j')$. Therefore the successor sequent $H_i \vdash_{\Delta_i'} t \in U_i'$ is equivalent to $H_j \vdash_{\Delta_j'} t \in U_j'$.

- Rule 9) or 10) is applied to seq_1.
  Here $H_i \vdash_{\Delta_i} t \in U_i$ and $(t \in U_i) \notin H_i$. For $H_j \vdash_{\Delta_j} t \in U_j$ holds by induction hypothesis $U_i \sim U_j$ and $H_i \sim H_j$. Therefore $(t \in U_j) \notin H_j$. For the successor sequent holds
  $H_i \cup \{t \in U_i\} = H_i' \sim H_j' = H_j \cup \{t \in U_j\}$ and as
  $\sigma X.A(X, U_{i_1}, \ldots, U_{i_n}) = \Delta_i(U_i) \sim \Delta_j(U_j) = \sigma X.A(X, U_{j_1}, \ldots, U_{j_n})$ the successor sequents are equal.

□

PROPOSITION 2 (Size of the Computation Trees)
Consider a sequent $H_1 \vdash_{\Delta_1} s \in \Phi(U_1, \ldots U_n)$ having τ_1 as computation tree , and a sequent $H_2 \vdash_{\Delta_1} s \in \Phi(U_1, \ldots U_n)$ with $H_1 \leq_{U_i, U_i} H_2$ for all $1 \leq i \leq n$. Then $H_2 \vdash_{\Delta_2} s \in \Phi(U_1, \ldots U_n)$ has a computation tree τ_2 which s contained in τ_1 as subtree, such that (up to the hypothesis sets).

Proof: by induction in the structure of τ_1
omitted in this version □

PROPOSITION 3 (Combination of Hypothesis Trees)
Let τ_1 be the hypothesis tree derived from a computation tree of $H_1 \vdash_\Delta s \in U$ and τ_2 be the hypothesis tree derived from a computation tree of $H_2 \vdash_\Delta s \in U$. Then there exists a hypothesis set H_3 with $H_1 \cap H_2 \subseteq H_3 \subset H_1 \cup H_2$, and $\tau_1 \circ \tau_2$ as in definition 5 is the hypothesis tree derived from a computation tree of $H_3 \vdash_\Delta s \in U$.

Proof: by induction in the structure of τ_1 and τ_2
omitted in this version □

PROPOSITION 4 (Correctness I)
Let τ_1 be a computation tree of $H \vdash_\Delta s \in U$ and HB its hypothesis tree.
The hypothesis tree HB is evaluated with the hypothesis set H to true, iff $H \vdash_\Delta s \in U$ has a successful subtableau.
The hypothesis tree HB is evaluated with the hypothesis set H to false, iff $H \vdash_\Delta s \in U$ has no successful subtableau.

Proof: omitted in this version □

PROPOSITION 5 (Correctness II)
Let HB be the hypothesis tree of a sequent $H_1 \vdash_\Delta s \in U$, and H_2 a hypothesis set.
If HB together with H_2 evaluates to true, then the sequent $H_2 \vdash_\Delta s \in U$ has a successful subtableau.
If HB together with H_2 evaluates to false, then the sequent $H_2 \vdash_\Delta s \in U$ has no successful subtableau.

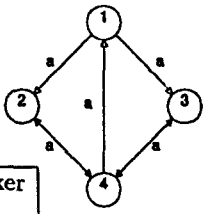
Proof: omitted in this version □

7 Benchmarks

The presented algorithm is implemented in *QUINTUS-PROLOG* on a *SUN/SPARC* system.
In the following examples the standard tableau model checker is compared to the tableau recycling model checker. As units of measurement we took the number of nodes in the computation tree and the system time which the model checker took to solve the task.

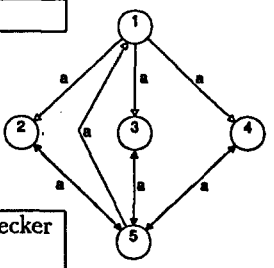
Examples 2 to 5 from section 4, *Fig.1*:

	standard tableau model checker		tableau recycling model checker	
	number of nodes	time	number of nodes	time
n=3, k=3	105	< 1s	25	< 1s
n=4, k=3	211	1s	31	< 1s
n=3, k=4	321	1s	33	< 1s
n=4, k=4	851	2s	41	< 1s



Example 6 : $\vdash 1 \in \nu Z.\langle a\rangle\mu X.\langle a\rangle\langle a\rangle X \wedge \langle a\rangle\langle a\rangle Z$

standard tableau model checker		tableau recycling model checker	
number of nodes	time	number of nodes	time
32766	74s	139	4s



Example 7 : $\vdash 1 \in \nu Z.\langle a\rangle\mu X.\langle a\rangle\langle a\rangle X \wedge \langle a\rangle\langle a\rangle Z$

standard tableau model checker		tableau recycling model checker	
number of nodes	time	number of nodes	time
> 22100000	> 1.5h	218	1326s

8 Conclusion

A tableau based model checker for the full modal mu-calculus was presented, which profits from the idea to recycle subtableaux which have been calculated earlier in the model checker algorithm. The information contained in a subtableau is reduced to a much smaller expression. An implementation of this algorithm showed in several examples an impressive acceleration.

Future work will include the following aspects:
We continue to get more experience with real world transition systems and relevant modal properties when verified with the tableau recycling model checker and different model checking approaches.
Secondly in this paper an idea was worked out how the maximal information can be

preserved during a proof run. Heuristic methods could help to do it without *maximal* information in order to reduce memory expense.
Finally we will continue in investigating the complexity of the model checker algorithm.

Acknowledgement I thank Dirk Taubner for many motivating discussions. Florian Mengedoht implemented the algorithm.

9 References

[B] Julian Bradfield, Verifying Temporal Properties of Systems, Birkhäuser,1992.

[BC] J.R.Burch, E.M.Clarke, K.L.McMillan, D.L.Dill and L.J.Hwang, Symbolic model checking: 10^{20} states and beyond, in: *Information and Computation*, Vol 98, Num 2, June 1992, (141-170).

[BS] Julian Bradfield and Colin Stirling, Verifying Temporal Properties of Processes, CONCUR 1990, in: LNCS 458, Springer Verlag, Berlin, 1991, (115-125).

[C] Rance Cleaveland, Tableau Based Model Checking in the Propositional Mu-Calculus, in: *Acta Informatica*,1990,(725-747).

[CS1] Rance Cleaveland and Bernhard Steffen, A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus, in: *Proc. of the Third Workshop on Computer Aided Verification* ,LNCS 575, 1992,(48-58).

[CS2] Rance Cleaveland and Bernhard Steffen, Computing Behavourial Relations, Logically, in: *Proc. ICALP '91*, 1991.

[EFT] Reinhard Enders, Thomas Filkorn, Dirk Taubner, Generating BDDs for Symbolic Model Checking in CCS, in: *Proc. of the Third Workshop on Computer Aided Verification* ,LNCS 575, 1992, (203-213).

[EL] E.Allen Emerson and Ching-Luang Lei, Efficient model checking in fragments of the propositional mu-calculus, in: *Proc. of Symposium on Logic in Computer Science*,IEEE, 1986, (267-278).

[S] Colin Stirling, Modal and Temporal Logics, in: S.Abramsky, D.Gabbay, and T.Maibaum, editors, *Handbook of Logic in Computer Science*, Oxford University Press.

[SW] Colin Stirling and David Walker, Local model checking in the modal mu-calculus, in: *Proc. International Conference on Theory and Practice of Software Development*, LNCS 351, Springer Verlag, Berlin,1989,(369-382).