From tableaux to witnesses for the modal μ-calculus

November 3, 1995

Alexander Kick*

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

Abstract

Symbolic temporal logic model checking is an automatic verification method. One of its main features is that a counterexample can be constructed when a temporal formula does not hold for the model. Most model checkers so far have restricted the type of formulae that can be checked and for which counterexamples can be constructed to fair CTL formulae. This paper shows how counterexamples and witnesses for the whole μ-calculus can be constructed. The witness construction is derived in a formal way from the local model checking method. The witness construction presented in this paper is polynomial in the model and the formula.

1 Introduction

Complex state-transition systems occur frequently in the design of sequential circuits and protocols. Symbolic temporal logic model checking [CGL93] has shown in practice to be an extremely useful automatic verification method. In this approach, the state-transition systems are checked with respect to a propositional temporal logic specification.

If the model satisfies the specification the model checker returns true. Otherwise, a counterexample can be constructed, which helps finding the error in the design. The latter facility is one of the most important advantages of model checking over other verification approaches.

*Supported by DFG Vo 287/5-2

The symbolic model checker SMV developed at Carnegie Mellon University ([McM93]) based on OBDDs [Bry92] can check fair CTL (FCTL) ([CGL93]) formulae and construct counterexamples for these formulae. Model checkers which can check μ-calculus formulae [Koz83] have greater expressive power, since arbitrary μ-calculus formulae can be checked in contrast to the small subclass FCTL of the μ-calculus, and are more general since many problems can be translated into the μ-calculus.

In [CGMZ94], it is described how to construct counterexamples for FCTL formulae. To our knowledge, noone has yet investigated how to construct counterexamples for arbitrary μ-calculus formulae. To be able to construct counterexamples for μ-calculus formulae, however, is necessary to make a μ-calculus model checker as useful as a CTL model checker. In this paper, we therefore investigate how counterexamples for μ-calculus formulae can be computed.

The rest of the paper is structured as follows. Section 2 consists of preliminaries where the μ-calculus is repeated, some terminology is introduced and a modified model checking algorithm is given. In Section 3, we repeat tableau based model checking. In Section 4, we show how to construct a tableau by using information from prior model checking. In Section 5, we define collapsed isomorphic pseudo tableaux where isomorphic subparts in a tableau are eliminated and give a direct algorithm for constructing such collapsed tableaux. In Section 6, we further reduce the size of a collapsed tableau. In Section 7, we draw some conclusions. Note that we will not care about counterexamples for a formula f in the rest of the paper since counterexamples are simply witnesses for the negation of formula f.

2 The modal μ-calculus

In this section we remind the reader of the syntax and semantics of the modal μ-calculus, we introduce some notation and give a slightly modified model checking algorithm which suits our purposes of witness construction. We mainly follow [EL86].

2.1 Syntax and semantics

There are the following syntactic classes:

- *PropCon*, the class of propositional constants $P, Q, R, \ldots$
- *PropVar*, the class of propositional variables $X, Y, Z, \ldots$
- *ProgAt*, the class of program atoms or basic actions $A, B, C, \ldots$
- *Form*, the class of formulae L_μ of the propositional μ-calculus $p, q \ldots$, defined by

$$p ::= P|X|p \wedge q|\neg p|\mu X.p|\langle A\rangle p$$

where in $\mu X.p$, p is any formula syntactically monotone in the propositional variable X, i.e., all free occurrences of X in p fall under an even number of negations.

The other connectives are introduced as abbreviations in the usual way: $p \vee q$ abbreviates $\neg(\neg p \wedge \neg q)$, [A] p abbreviates $\neg\langle A\rangle\neg p$ and $\nu X.p(X)$ abbreviates $\neg\mu X.\neg p(\neg X)$.

The semantics of the μ-calculus is defined with respect to a model. A model is a triple $M = (S, R, L)$ where S is a set of states, $R\colon ProgAt \to \mathcal{P}(S \times S)$ is a mapping from program atoms A to a set of state transitions involving A, and $L\colon S \to \mathcal{P}(PropCon)$ labels each state with a set of atomic propositions true in that state.

In the rest of the paper, we rarely need the program atoms. Therefore, we introduce the abbreviation $R := \bigcup\{(s,t)|(s,t) \in R(A) \wedge A \in ProgAt\}$. A path in M is a sequence of states: $\pi = s_0 s_1 \ldots$ such that $\forall i \geq 0 : (s_i, s_{i+1}) \in R$. We assume that the models we deal with in the following are finite (i.e., S and $ProgAt$ are finite). The semantics for the modal μ-calculus is given via least and greatest fixpoints. For the details, the reader is referred to [EL86].

The meanings of formulae is defined relative to valuations $\rho\colon PropVar \to \mathcal{P}(S)$. The variant valution $\rho[T/X]$ is defined by

$$\rho[T/X](Y) = \begin{cases} T & Y \equiv X \\ \rho(Y) & \text{otherwise} \end{cases}$$

The set of states satisfying a formula f in a model M with valuation ρ is inductively defined as

$$[\![P]\!]\rho = \{s|P \in L(s)\}$$
$$[\![X]\!]\rho = \rho(X)$$
$$[\![p \wedge q]\!]\rho = [\![p]\!] \cap [\![q]\!]$$
$$[\![\neg p]\!]\rho = S \setminus [\![p]\!]\rho$$
$$[\![\langle A\rangle p]\!]\rho = \{s|\exists t \in S\colon (s,t) \in R(A) \wedge t \in [\![p]\!]\rho\}$$
$$[\![\mu X.p]\!]\rho = \bigcap\{S' \subseteq S|[\![p]\!]\rho[S'/X] \subseteq S'\}$$

We define

$$s, \rho \models p \Leftrightarrow s \in [\![p]\!]\rho$$

2.2 Some terminology

$\langle\rangle$ shall stand for any $\langle A\rangle$, $[\,]$ for any $[A]$. The terms *subformula, closed formula, bound* and *free variables* are used as usual. We write $p \preceq q$ if p is a subformula of q. A *μ-, ν-subformula* is a subformula whose main connective is μ and ν, respectively. A variable X is called a μ-variable or ν-variable if X occurs as $\mu X.p$ or $\nu X.p$ in a formula, respectively. *Alternation depth* $\mathcal{A}(f)$ of a formula f is defined in [EL86]. L_{μ_i} shall denote the sublanguage of L_μ with alternation depth i.

$\sigma X.p(X)$ shall stand for either $\mu X.p(X)$ or $\nu X.p(X)$, $\Box$ shall stand for either $[\,]$ or $\langle\rangle$. Let $b'(X) = p(X)$ if $\sigma X.p(X)$ appears as a subformula of an original formula f. We say that X is *in the scope of* $[\,]$, $\langle\rangle$ *in formula* f if X is a subformula of a subformula of f of the form $[\,]q$ and $\langle\rangle q$, respectively.

A formula is said to be in *propositional normal form (PNF)* provided that no variable is quantified twice and all the negations are applied to atomic propositions only. Note that every formula can be put in PNF. It can be shown by induction on the number of fixpoint iterations that each $\sigma X.p(X)$ can be transformed into a formula without σ or into $\sigma X.p(X)$, where X occurs in $p(X)$ and all occurrences of X in $p(X)$ are in the scope of $\langle\rangle$ or $[\,]$. In the rest of the paper we suppose (without loss of generality) that all μ-calculus formulae are in PNF and closed and all subformulae $\sigma X.p(X)$ fulfill the above constraint.

2.3 Model checking the modal μ-calculus

The model checking problem is: given a model M, a formula f and a state s in M, is $s \in [\![f]\!]\rho$? We do not need to care about ρ, since it can be arbitrary in the case of closed formulae which we consider only. For this reason, we also write $s \models f$ instead of $s, \rho \models f$. We give here a modified model checking algorithm where information needed for the later witness construction is saved.

$\vec{x} = (x_1, \ldots, x_m) \in \mathbb{N}_0^m$ shall denote a vector of integers. The ordering on these vectors is defined by: $(x) < (y) \Leftrightarrow x < y$, $(x_1, \ldots, x_m) < (y_1, \ldots, y_m) \Leftrightarrow x_1 < y_1 \vee x_1 = y_1 \wedge (x_2, \ldots, x_m) < (y_2, \ldots, y_m)$.

For vectors with different lengths we define

$$(x_1, \ldots, x_m) < (y_1, \ldots, y_l) \Leftrightarrow \begin{cases} (x_1, \ldots, x_m) < (y_1, \ldots, y_m) & m \leq l \\ (x_1, \ldots, x_l) < (y_1, \ldots, y_l) & \text{otherwise} \end{cases}$$

$$(x_1, \ldots, x_m) = (y_1, \ldots, y_l) \Leftrightarrow \begin{cases} \forall 1 \leq i \leq m : x_i = y_i & m \leq l \\ \forall 1 \leq i \leq l : x_i = y_i & m > l \end{cases}$$

Note that this equality on vectors is not transitive.
$\vec{x} \leq \vec{y} \Leftrightarrow \vec{x} < \vec{y} \vee \vec{x} = \vec{y}$

We write $\vec{x} \sqsubseteq \vec{y}$ if $\vec{x}$ is a prefix of $\vec{y}$.

Vectors shall denote the iteration numbers of the fixpoint iterations of subformulae of the form $\mu X.p$ in the model checking algorithm below.

Algorithm 1
For a given model M and a given formula f which contains propositional variables $X^1, \ldots, X^n$, where $X^1, \ldots X^m$ denote the μ-variables and $X^{m+1} \ldots X^n$ denote the ν-variables in f, $mc(f, ())$ determines the set of states of the model which fulfill f.

function *mc(f:Predicate,* $(x_1, \ldots, x_k)$*:*$\mathbb{N}_0^*$*): Predicate*
begin
case *f of the form*
X^j : $S' := S^j$;
P : $S' := \{s | P \in L(s)\}$;
$p \wedge q$: $S' := mc(p, (x_1, \ldots, x_k)) \cap mc(q, (x_1, \ldots, x_k))$;
$p \vee q$: $S' := mc(p, (x_1, \ldots, x_k)) \cup mc(q, (x_1, \ldots, x_k))$;
$\neg p$: $S' := S \setminus mc(p, (x_1, \ldots, x_k))$;
$\langle\rangle p$: $S' := \{s \in S | \exists t \in mc(p, (x_1, \ldots, x_k)) : (s, t) \in R\}$;
$[\,]p$: $S^* = mc(p, (x_1, \ldots, x_k))$; $S' := \{s \in S | \forall t \in S : (s, t) \in R \rightarrow t \in S^*\}$;
$\mu X^j.p_j(X)$:
begin
$S^j := \emptyset$;
$i := 0$;
repeat
$S' := S^j$;
$S^j := mc(p_j, (x_1, \ldots, x_k, i))$;
$i := i + 1$;
until $S' = S^j$;
end
$\nu X^j.p_j(X)$:
begin
$S^j := S$;
repeat
for all $g \prec \nu X^j.p_j(X)$ **for all** $\vec{y}$ **with** $(x_1, \ldots, x_k) \sqsubseteq \vec{y}$: $g_{\vec{y}} := \emptyset$;
$S' := S^j$;
$S^j := mc(p_j, (x_1, \ldots, x_k))$;
until $S' = S^j$;
end
v **esac**
$f_{(x_1, \ldots, x_k)} := S'$;
return S'
end

for all $p \preceq f$ **for all** $(x_1, \ldots, x_k)$: $p_{(x_1,\ldots,x_k)} := \emptyset$;
$mc(f, ())$;

Let f be the original formula to be model checked. Given $\sigma Y.q(Y) \prec \sigma X.p(X)$ we write $Y < X$. In the fixpoint iterations of subformulae $\mu X.q$, the values of their subformulae are saved together with the vector of iterations. Let p be a subformula of a maximal subterm of f of the form $\mu X^1 \ldots . \mu X^k.q$ with $p \preceq q$, i.e., no other μ-variables appear before p. Then $p_{(x_1,\ldots,x_k)}$ is the value of p during the model checking procedure where X^1 is in iteration $x_1, \ldots, X^k$ in iteration x_k.

Lemma 1 *If* $Y \subseteq Y'$ *then* $\sigma X.p(Y, X) \subseteq \sigma X.p(Y', X)$ *and* $p^i(Y, false) \subseteq p^i(Y', false)$. *This is also true for an arbitrary number of free variables* Y.

Proof:

- The case for μ was already explained in the article by Emerson. Since p is monotonic in Y (because the formulae are in PNF) we have: $p^i(Y, \text{false}) \subseteq p^i(Y', \text{false})$. Since $\mu X.p(Y, X) = \bigcup_i p^i(Y, X)$ the claim follows immediately.

- Let X_f be the greatest fixpoint of $\nu X.p(Y, X)$. Since p is monotonic in Y we have for $Y' \supseteq Y$: $p(Y, X_f) \subseteq p(Y', X_f)$ and since $X_f \subseteq p(Y, X_f)$ it follows that $X_f \subseteq \bigcup\{S' | S' \subseteq p(Y', S')\} = \nu X.p(Y', X)$, i.e. $\nu X.p(Y, X) \subseteq \nu X.p(Y', X)$.

■

Proposition 1 $\forall(x_1, \ldots, x_k, \ldots, x_l)\forall p \prec f$: $p_{(x_1,\ldots,x_k,\ldots,x_l)} \subseteq p_{(x_1,\ldots,x_k+1,\ldots,x_l)}$

Proof: We first consider the case $l = k$, i.e., subsequent iterations of $p(X^k)$. Certainly, for all $X^i > X^k$ the values for X^i are the same in the fixpoint iteration of $\mu X^k.p(X^k)$. However, the value for X^k itself has increased in iteration x_k (otherwise there would not be another iteration x_{k+1}), i.e., $X^k_{(x_1,\ldots,x_k-1,\ldots,x_j)} \subset X^k_{(x_1,\ldots,x_k,\ldots,x_j)}$.

For top-level subformulae $\sigma Z.q(X^k, Z)$ of $\mu X^k.p(X^k)$ we have:

$$\sigma Z.q(X^k_{(x_1,\ldots,x_k-1,\ldots,x_j)}, Z) \subseteq \sigma Z.q(X^k_{(x_1,\ldots,x_k,\ldots,x_j)}, Z)$$

This follows from Lemma 1. Since all other variables remain the same, and of course also $P, \neg P$, and the connectives $\wedge, \vee, \langle\rangle, [\,]$ are monotonic the claim follows immediately for $p \not\prec \sigma Z.q(X^k, Z)$.

In the case $p \prec \nu Z.q(X^k, Z)$ the value for Z is greater in the deeper iterations, i.e., iterations of $\sigma W.r$ with $W < Z$, in iterations $(x_1, \ldots, x_{k+1}, \ldots, x_j)$ than in iterations $(x_1, \ldots, x_k, \ldots, x_j)$. Therefore, for all subformulae of $\nu Z.q$ of the form $p_{(x_1,\ldots,x_k)}$ it also holds that $p_{(x_1,\ldots,x_k)} \subseteq p_{(x_1,\ldots,x_k+1)}$.

The proof for $l > k$ goes in the same way as above for the case $l = k$ except that $\forall X^i > X^l \colon X^i_{(x_1,\dots,x_k-1,\dots,x_j)} \subseteq X^i_{(x_1,\dots,x_k,\dots,x_j)}$.

■

This proposition allows us to conclude that if there is a vector $\vec{x}$ with $s \in p_{\vec{x}}$ then after the model checking stops then $s \in p\theta$ where θ substitutes the propositional variables in p by their last fixpoints.

Note that $X^{(x_1,\dots,x_{k-1},x_k,\dots,x_l)} = X^{(x_1,\dots,x_{k-1},x_k,\dots,x_j)}$ if $\mu X.p(X)$ is labeled $(\mu X.p(X))^{(x_1,\dots,x_{k-1})}$ in the model checking algorithm and therefore we define $X^{(x_1,\dots,x_{k-1},x_k)} = X^{(x_1,\dots,x_{k-1},x_k,\dots,x_l)}$. Similarly, if $(\nu X.p(X))^{(x_1,\dots,x_k)}$ we have $X^{(x_1,\dots,x_k,\dots,x_l)} = X^{(x_1,\dots,x_k,\dots,x_j)}$ and we define $X^{(x_1,\dots,x_k)} = X^{(x_1,\dots,x_k,\dots,x_l)}$. In the rest of the paper we use these abbreviations.

Definition 1
In the following, let $p \preceq f$, $p_{\vec{x}}$ obtained by $mc(f,())$ where the model is $M = (S,R,L)$, $s \in S$ and $\vec{x} \in \mathbb{N}_0^*$.

- $\forall p \forall \vec{x}$:
  $(p^{(x_1,\dots,x_j+1)} = p_{(x_1,\dots,x_j+1)} \setminus p_{(x_1,\dots,x_j)}) \wedge (p^{(x_1,\dots,x_j-1,0)} = p_{(x_1,\dots,x_j-1,0)})$
- $\forall p \preceq f \colon l(s,p) = (s \in \bigvee_{\vec{x}} p^{\vec{x}})$
- $min \colon S \times L_\mu \to L_\mu \times (\mathbb{N}_0^* \cup \{\bot\})$
  $$min(s,p) = \begin{cases} p^{min\{\vec{y} \mid s \in p^{\vec{y}}\}} & l(s,p) = true \\ \bot & otherwise \end{cases}$$
- $v \colon L_\mu \times (\mathbb{N}_0^* \cup \{\bot\}) \to (\mathbb{N}_0^* \cup \{\infty\})$
  $$v(g) = \begin{cases} \vec{x} & g = p^{\vec{x}} \\ \infty & g = \bot \end{cases}$$
  In the following, let $\forall \vec{x} \in \mathbb{N}_0^* \colon \vec{x} < \infty$.

During model checking, states s are marked with subformulae p of f which are true in s together with the iteration depths during which s is added to the set of states fulfilling p: $p^{\vec{x}}$. $s \in p^{\vec{x}}$ means that s is added to the states fulfilling p in iteration $\vec{x}$. This labeling is firmly recorded only in the last iteration of ν-variables X for $p \prec \nu X.q$. Only the iterations of the μ-variables are important in the following, so the iteration depths of the ν-variables are not recorded.

Lemma 2 *Let $p \wedge q, p \vee q, \langle\rangle p, [\,]p, \nu X.p(X)$ be subformulae of formula f model checked by the above algorithm and $s \in S$ arbitrary with $l(s, p \wedge q) = true, l(s, p \vee q) = true, \dots$, respectively, in the items below. Then*

- $v(min(s,p)) \leq v(min(s,p \wedge q)) \wedge v(min(s,q)) \leq v(min(s,p \wedge q))$
- $v(min(s,p)) \leq v(min(s,p \vee q)) \vee v(min(s,q)) \leq v(min(s,p \vee q))$

- $\exists s' \in S : (s,s') \in R \wedge v(min(s',p)) \leq v(min(s,\langle\rangle p))$
- $\forall s' \in S : (s,s') \in R \rightarrow v(min(s',p)) \leq v(min(s,[\,]p))$
- $v(min(s,\nu X.p(X))) = v(min(s,X)) = v(min(s,p(X)))$

Proof: The model checking algorithm decides upon the truth of a formula in a state s only *after* the truth of the subformulae in state s has been decided. ■

Fact 1 *From Algorithm 1 it is clear that for* $\mu X^j.p(X^j)$*:*

$$(\forall i \in \mathbb{N}: (X^j)^{(x_1,\ldots,x_{k-1},i)} = (p(X^j))^{(x_1,\ldots,x_{k-1},i-1)}) \wedge (X^j)^{(x_1,\ldots,x_{k-1},0)} = false$$

and in particular

$$(p(X^j))^{(x_1,\ldots,x_{k-1},0)} = (X^j)^{(x_1,\ldots,x_{k-1},1))} = p(false)$$

As a consequence, if $l(s,\mu X^j.p(X^j)) = true$*, then*

$$v(min(s,\mu X^j.p(X^j))) = v(min(s,X^j)) > v(min(s,p(X^j)))$$

Note that the saving of information does not change the space complexity of the algorithm which is still $O(|f| \cdot |M|)$ (and also not the time complexity). Since only $min(s,p)$ for $p \preceq f$ is needed later for witness construction a state s with $l(s,p) =$ true needs to be labeled only with $min(s,p)$ and with no other $p^{\vec{x}}$.

In [EL86] an improved algorithm for model checking is presented on which the following theorem is based.

Theorem 1 (Emerson,Lei) *Model checking can be done in time* $O((|M| \cdot |f|)^{\mathcal{A}(f)+1})$ *where* $|M| = |S| + |R|$ *and* $|f|$ *is the length of formula f.*

3 Model checking by tableaux

Local model checking ([SW91], [Cle90]) was devised as a procedure to determine the truth of a formula for a state in a model for the case that the property can be determined in a small circumference of a state (locality condition). In this case, local model checking should have advantages over model checking algorithms which explore the whole state space to determine the truth of the formula.

A constructed successful tableau can at the same time be viewed as a witness for the truth of a formula in a model. However, there are two problems which prevent us from directly taking a tableau as a witness if the locality condition does not hold. One problem with local model checking in its present form is that OBDDs can not be used and thus it is slower than symbolic model checking. Another problem is that the size of a successful tableau can be exponential in

the model. This would make error finding even worse.

We present here the tableau construction described in [SW91].

The syntax of the μ-calculus is extended to embrace a family of propositional constant symbols. Associated with a constant U is a declaration of the form $U = A$ where A is a closed formula. A definition list is a sequence Δ of declarations $U_1 = A_1, \ldots, U_n = A_n$ such that $U_i \neq U_j$ whenever $i \neq j$ and such that each constant occurring in A_i is one of $U_1, \ldots, U_{i-1}$. Let $dom(\Delta) = \{U_1, \ldots, U_n\}$ and $\Delta(U_i) = A_i$. $\Delta.(U = A)$ means appending $U = A$ to the definition list Δ. A definition list Δ is *admissible for* B if every constant occurring in B is declared in Δ. In the following, $t\langle u \leftarrow v\rangle$ shall stand for syntactic substitution of u by v in t.

Definition lists are used to keep track of the "dynamically changing" subformulae as fixpoints are unrolled.

Definition 2
If $\Delta : U_1 = A_1, \ldots, U_n = A_n$ *is admissible for* B *then* $[\![B_\Delta]\!]\rho =_{df} [\![B]\!]\rho_n$ *where* $\rho_0 = \rho$ *and* $\rho_{i+1} = \rho_i[[\![A_{i+1}]\!]\rho_i / U_{i+1}]$.

Lemma 3 $[\![B_{\Delta.U=A}]\!]\rho = [\![(B\langle U \leftarrow A\rangle)_\Delta]\!]\rho$

Definition 3 (Tableau rules TR)

$$\frac{s \vdash_\Delta p \wedge q}{s \vdash_\Delta p \quad s \vdash_\Delta q}$$

$$\frac{s \vdash_\Delta p \vee q}{s \vdash_\Delta p} \qquad \frac{s \vdash_\Delta p \vee q}{s \vdash_\Delta q}$$

$$\frac{s \vdash_\Delta \langle\rangle p}{s' \vdash_\Delta p} \quad (s, s') \in R$$

$$\frac{s \vdash_\Delta [\,]p}{s_1 \vdash_\Delta p \ldots s_n \vdash_\Delta p} \quad \{s_1, \ldots, s_n\} = \{s' | (s, s') \in R\}$$

$$\frac{s \vdash_\Delta \sigma Z.p}{s \vdash_{\Delta'} U} \quad \mathcal{B} \text{ and } \Delta' = \Delta.U = \sigma Z.p$$

$$\frac{s \vdash_\Delta U}{s \vdash_\Delta p\langle Z \leftarrow U\rangle} \quad \mathcal{C} \text{ and } \Delta(U) = \sigma Z.p$$

The condition $\mathcal{B}$ is that the new U must be different from any U' where there is a $t \vdash_{\Delta''} U'$ for some Δ'', t, appearing in the proof tree as a node above the current premise $s \vdash_{\Delta} \sigma Z.p$. The condition $\mathcal{C}$ is that no node above the current premise, $s \vdash_{\Delta} U$, in the proof tree is labelled $s \vdash_{\Delta'} U$ for some Δ'.

A tableau for $s \vdash f$ is a maximal proof tree whose root is labelled with the sequent $s \vdash f$. The sequents labelling the immediate successors of a node are determined by application of one of the rules. Maximality means that no rule applies to a sequent labelling a leaf of a tableau. We give here a more formal definition of tableau which we need later to define isomorphism in tableaux.

Definition 4 (Tableau)
A tableau for $s \vdash f$ is a triple (V, E, v) where $V \subset \mathbb{N}$, $E = V \times V$, and v labels the vertices: $v : V \to Seq$ where $Seq = \{s \vdash f | s \in S, f \in L_\mu\}$. V, E and v are inductively defined:

1. *$1 \in V$, $v(1) = s \vdash f$*
2. *Vertices and edges can be added according to the tableau rules, i.e., if $\dfrac{s \vdash A}{s_1 \vdash A_1 \ldots s_k \vdash A_k}$ a tableau rule then we can add new vertices $\{u_1, \ldots, u_k\}$ not already in V to V, with $v(u_1) = s_1 \vdash A_1, \ldots, v(u_k) = s_k \vdash A_k$, and edges $(u_0, u_1), \ldots, (u_0, u_k)$ to E if $v(u_0) = s \vdash A$.*

A vertex is called a leaf if it does not have any outgoing edges. A tableau is further required to be maximal, i.e., no rule applies to $v(l)$ where l is an arbitrary leaf.

Theorem 2 *Every tableau for $s \vdash f$ is finite if $M = (S, R, L)$ is finite.*

Proof: This was already proved in [SW91]. Here we give a much shorter and more easily understandable proof.

All rules of TR decrease the length of the formula except the last one. Let $\sigma X.p$ be a top-level subformula of f. Then the sequent $s \vdash_{\Delta} U$ with $\Delta(U) = \sigma X.p$ can occur at most $|S|+1$ times. This is because of the finiteness of M and because no other variables can cause another sequent $t \vdash \sigma X.p$ (since it is top-level). This U can have spawned at most $|S|$ similar tableaux for top-level σ-subformulae of $\sigma X.p$.

We can repeat this argument for these σ-subformulae of $\sigma X.p$ and their σ-subformulae until the smallest σ-subformula has been reached.

As a consequence, there can be only finitely many vertices in the tableau. ■

Definition 5 (Successful tableau)
A successful tableau for $s \vdash f$ is a finite tableau in which every leaf is labelled by a sequent $t \vdash_{\Delta} p$ fulfilling one of the following requirements:

1. $p = P$ *and* $P \in L(t)$
2. $p = \neg P$ *and* $P \notin L(t)$
3. $p = [\,]q$
4. $p = U$ *and* $\Delta(U) = \nu Z.r$

An unsuccessful tableau has at least one false leaf. An interesting failure is when a leaf is labelled $t \vdash_\Delta U$ where $\Delta(U) = \mu Z.p$ and above it is a node labelled $t \vdash_{\Delta'} U$.

The tableau rules work according to the semantic definition of the operators. The only interesting case is $\sigma X.p(X)$. A variable is created which is different from all other variables created so far. This variable keeps track of the path described by $\sigma X.p(X)$. In the case of $\nu X.p(X)$, the tracking of the path can successfully terminate when a state marked with $s \vdash X$ is reached again. In the case of $\mu X.p(X)$ exactly this must not happen. Instead, the path must dissolve by reaching p(false) when running along that path.

Theorem 3 (Stirling, Walker) *$s \vdash f$ has a successful tableau if and only if $s \models f$.*

The proof of this theorem consists of two parts. The authors, however, make it themselves too easy when they prove the direction "If $s \models f$ then $s \vdash f$ has a successful tableau.". It does not suffice to know the iteration when a state is added to $\mu X.p$. It is also important to make the right choices when at a state $s \vdash p \vee q$ or $s \vdash \langle\rangle p$. If making the wrong choice, the tableau can contain a loop in the case of $\mu X.p$, i.e., the tableau can contain a leaf $t \vdash_\Delta U$ with $\Delta(U) = \mu X.p$, thus making it unsuccessful. This can be avoided by using the information saved at prior model checking as in Algorithm 1. The definitions and proofs for the pseudo tableau in the next section correct the proof for the mentioned direction.

4 Constructing a tableau from information from prior symbolic model checking

Definition 6 (Reverse substitution)
Let $\Delta = (U_1 = \ldots) \ldots (U_n = \ldots)$. Then

- $\Theta(U) = Z$ *if* $\Delta(U) = \sigma Z.p$
- $\overline{f} = ((f\langle U_n \leftarrow \Theta(U_n)\rangle) \ldots \langle U_1 \leftarrow \Theta(U_1)\rangle)$
  $\overline{f}$ is f where the declaration constants are substituted by the original variables in the formula.
- $\overline{s \vdash f} = s \vdash \overline{f}$

- $\overline{f^{\vec{x}}} = \overline{f}^{\vec{x}}$

We extend the definition of min and $s \in p$ to formulae p containing declaration constants:

$$min(s,p) = \begin{cases} p^{v(min(s,\overline{p}))} & l(s,\overline{p}) = \text{true} \\ \bot & \text{otherwise} \end{cases}$$

$$s \in p^{\vec{x}} \Leftrightarrow s \in \overline{p}^{\vec{x}}$$

Definition 7 (Tableau rules PTR)

$$\frac{s \vdash_\Delta (p \wedge q)^{\vec{x}}}{s \vdash_\Delta min(s,p) \quad s \vdash_\Delta min(s,q)}$$

$$\frac{s \vdash_\Delta (p \vee q)^{\vec{x}}}{choose(s \vdash_\Delta (p \vee q)^{\vec{x}})}$$

$$\frac{s \vdash_\Delta (\langle\rangle p)^{\vec{x}}}{choose(s \vdash_\Delta (\langle\rangle p)^{\vec{x}})}$$

$$\frac{s \vdash_\Delta ([\,]p)^{\vec{x}}}{s_1 \vdash_\Delta min(s_1,p) \ldots s_n \vdash_\Delta min(s_n,p)} \quad \{s_1,\ldots,s_n\} = \{s' | (s,s') \in R\}$$

$$\frac{s \vdash_\Delta (\sigma Z.p)^{\vec{x}}}{s \vdash_{\Delta'} U^{\vec{y}}} \quad \mathcal{B} \text{ and } \Delta' = \Delta.U = \sigma Z.p \wedge \vec{y} = v(min(s,Z))$$

$$\frac{s \vdash_\Delta U^{\vec{x}}}{s \vdash_\Delta (p\langle Z \leftarrow U\rangle)^{\vec{y}}} \quad \mathcal{C} \text{ and } \Delta(U) = \sigma Z.p \wedge \vec{y} = min(s,p)$$

The condition $\mathcal{B}$ is that the new U must be different from any U' where there is a $t \vdash_{\Delta''} (U')^{\vec{z}}$ for some $\Delta'', t, \vec{z}$, appearing in the proof tree as a node above the current premise $s \vdash_\Delta (\sigma Z.p)^{\vec{x}}$. The condition $\mathcal{C}$ is that no node above the current premise, $s \vdash_\Delta U^{\vec{x}}$, in the proof tree is labelled $s \vdash_{\Delta'} U^{\vec{x}}$ for some Δ'.

$choose(s \vdash_\Delta (p \vee q)^{\vec{x}}) =$
  choose $u \in \{s \vdash_\Delta min(s,p) | v(min(s,p)) \leq \vec{x}\} \cup \{s \vdash_\Delta min(s,q) | v(min(s,q)) \leq \vec{x}\}$;
  return u;

$choose(s \vdash_\Delta (\langle\rangle p)^{\vec{x}}) =$
  choose $s' \in \{s'' | (s,s'') \in R \wedge s'' \in p^{\vec{z}} \wedge \vec{z} \leq \vec{x}\}$;
  return $s' \vdash_\Delta min(s',p)$;

Definition 8 (Pseudo tableau)
A pseudo tableau for $s \vdash f$ is a tableau for $s \vdash min(s, f)$ where the rules PTR are used instead of TR.

Theorem 4 *Every pseudo tableau for $s \vdash f$ is finite if M is finite.*

Proof: In the same way as the proof for the finiteness of a tableau.

■

Definition 9 (Successful pseudo tableau)
A successful pseudo tableau for $s \vdash f$ is a finite pseudo tableau for $s \vdash f$ in which every leaf is labelled by a sequent $t \vdash_\Delta p^{\vec{x}}$ fulfilling the same requirements as in the successful tableau.

Theorem 5 *If $s \in [\![f]\!]\rho$ then $s \vdash f$ has a successful pseudo tableau.*

Proof: The tableau rules PTR guarantee that for the successors $t \vdash g$ also $t \in [\![g]\!]\rho$. Therefore, all nodes in the tableau are true since the tableau is started with a true root. It is clear from the semantics and the model checking algorithm that there are always such successors except for nodes which do not fulfill the side conditions $\{s_1, \ldots, s_n\} \neq \emptyset$ or $\mathcal{C}$ of the fourth and sixth rule, respectively.

The leaves of the maximal pseudo tableau will therefore be of the types $s \vdash p^{\vec{x}}$ where $p = P, p = \neg P, p = [\,]q, p = U$. All that remains to be shown in order for the pseudo tableau to be successful is that if $p = U$ then $\Delta(U) = \nu Z.r$. This is done in the following argument.

All tableau rules PTR do not increase $\vec{x}$. This follows from Lemma 2 and Fact 1. Fact 1 implies that the last rule, actually decreases $\vec{x}$ if $\Delta(U) = \mu Z.p$. Furthermore, the last rule has to be applied before any new $t \vdash_\Delta U^{\vec{y}}$ can be reached. As a consequence, if $\Delta(U) = \mu Z.p$ then for $s \vdash U^{\vec{x}}$ and $t \vdash U^{\vec{y}}$ lying on a path in the pseudo tableau where s appears before t it holds that $\vec{y} < \vec{x}$. Therefore, s and t must be different since there can be at most one $\vec{x}$ with $v(min(s, Z)) = \vec{x}$ (The unique minimum is always chosen.). It follows that there can not be a leaf $u \vdash_\Delta U$ with $\Delta(U) = (\mu Z.p)^{\vec{x}}$.

■

Theorem 6 *If $s \in [\![f]\!]\rho$ then $s \vdash f$ has a successful tableau.*

Proof: A successful tableau can be easily obtained from a successful pseudo tableau by stripping off the $\vec{x}$ from all formulae in the pseudo tableau.

■

5 Exploiting isomorphism in pseudo tableaux

5.1 Collapsed isomorphic pseudo tableaux

Definition 10 (Isomorphic edges in a tableau)
Let (V, E, v) be a tableau. Then, two edges in the tableau $(i, j) \in E$ and $(k, l) \in E$ are called isomorphic ($(i, j) \sim (k, l)$) iff $\overline{v(i)} = \overline{v(k)} \wedge \overline{v(j)} = \overline{v(l)}$.

Since isomorphic edges have the same structure it is enough to show just one of them to the user who wants to find an error. In order to make the reduced tableau as small as possible it is advantageous to have as many isomorphic edges as possible. This leads to the following definition.

Definition 11 (Isomorphic pseudo tableau (IPT))
An isomorphic pseudo tableau for $s \vdash f$ is defined in the same way as the pseudo tableau except that the construction has to proceed according to an additional constraint: whenever at a state $s \vdash f_2$ and there is a sequent $s \vdash f_1$ already reached with $\overline{f_1} = \overline{f_2}$ then the same choices have to be made at $s \vdash f_2$ as were made at $s \vdash f_1$. I.e., if $s \vdash f_2$ and $s \vdash f_1$ are not leaves and $\dfrac{s \vdash f_1}{s_1 \vdash A_1 \ldots s_k \vdash A_k}$ was applied to $s \vdash f_1$ then $\dfrac{s \vdash f_2}{s_1 \vdash B_1 \ldots s_k \vdash B_k}$ where $\overline{A_i} = \overline{B_i}$ is applied at $s \vdash f_2$. I.e., $\overline{choose(s \vdash_\Delta f_2^{\vec{x}})} = \overline{choose(s \vdash_{\Delta'} f_1^{\vec{x}})}$ for all such f_1, f_2.

Definition 12 (Collapsed isomorphic pseudo tableau (CIPT))
Let $T = (V, E, v)$ be a isomorphic pseudo tableau. We define the following equivalence relation on V:

$$\forall i, j \in V : i \equiv j \Leftrightarrow \overline{v(i)} = \overline{v(j)}$$

$[i] = \{j | i \equiv j\}$ denotes the equivalence class induced by this equivalence relation. The collapsed isomorphic pseudo tableau $T_\equiv = (V_\equiv, E_\equiv, v_\equiv)$ is then defined as

$$V_\equiv = \{[i] | i \in V\}$$

$$E_\equiv = \{([i], [j]) | \exists k, l \in V : k \in [i] \wedge l \in [j] \wedge (k, l) \in E\} = \{([i], [j]) | (i, j) \in E\}$$

$$v_\equiv : V_\equiv \rightarrow Seq$$

$$v_\equiv([i]) = \overline{v(i)}$$

Note that a collapsed isomorphic pseudo tableau is no longer a tree.

Lemma 4 *Let $T_\equiv = (V_\equiv, E_\equiv, v_\equiv)$ be the collapsed isomorphic pseudo tableau for $s \vdash f$ in the model $M = (S, R, L)$. Then $|V_\equiv| \leq |f| \cdot |S|$.*

5.2 Direct construction of collapsed isomorphic pseudo tableaux

First constructing an isomorphic pseudo tableau and then collapsing it is time consuming. The following algorithm constructs a collapsed isomorphic pseudo tableau directly.

Algorithm 2 (Direct CIPT construction)
function *choose*($s \vdash f^{\vec{x}}$: *Seq*):*Seq*
begin
  case f *of the form*
    $p \vee q$: **if** $(v(min(s,p)) \leq \vec{x}) \wedge (v(min(s,q)) \leq \vec{x})$ **then**
        begin
          choose $u := s \vdash min(s,p)$ *or* $u := s \vdash min(s,q)$
          return u;
        end
      else if $(v(min(s,p)) \leq \vec{x})$ **then return** $s \vdash min(s,p)$
      else return $s \vdash min(s,q)$
    $\langle\rangle p$: *choose* s' *such that* $s' \in \{s''|(s,s'') \in R \wedge s'' \in p^{\vec{z}} \wedge \vec{z} \leq \vec{x}\}$
      return $s' \vdash min(s',p)$
end

procedure *newnode*($s \vdash f^{\vec{x}}$: *Seq*,*k*: *node*)
begin
  if $(s \vdash f^{\vec{x}}) \in v(V)$ **then** $E := E \cup \{(k,i)\}$ **where** $v(i) = s \vdash f^{\vec{x}}$
  else
    begin
      create node j *with* $j \notin V$;
      $V := V \cup \{j\}; v(j) := s \vdash f^{\vec{x}}; E := E \cup \{(k,j)\}$;
      $c(s \vdash f^{\vec{x}}, j)$
    end
end

procedure *c*($s \vdash f^{\vec{x}}$: *Seq*, *k*: *node*)
begin
  case f *of the form*
    $P, \neg P$: **return**;
    $p \wedge q$: *newnode*($s \vdash min(s,p)$,*k*);
        newnode($s \vdash min(s,q)$,*k*);
    $p \vee q$: $u :=$ *choose*$(s \vdash (p \vee q)^{\vec{x}})$;
        newnode(*u*,*k*);
    $\langle\rangle p$: $u :=$ *choose*$(s \vdash (\langle\rangle p)^{\vec{x}})$;
        newnode(*u*,*k*);

$[\,]p$: **for all** s' **with** $(s,s') \in R$ **do**
newnode(s' ⊢ min(s',p),k);
$\sigma X.p(X)$: *newnode(s ⊢ min(s,X),k);*
X : *newnode(s ⊢ min(s,b'(X)),k);*
esac
end

$V := \{1\}; v(1) := s \vdash min(s,f); E := \emptyset$; $c(s \vdash min(s,f),1)$;
for all $k \in V$: *strip off* $\vec{x}$ *from* $v(k)$;

Theorem 7 *The tableau W constructed for* $s \vdash f$ *by Algorithm 2 is identical to the collapsed isomorphic pseudo tableau CIPT for* $s \vdash f$ *provided that* choose *chooses in the same way.*

Proof: We need to show that

$$\frac{s \vdash f_1}{t \vdash f_2} \in CIPT \Leftrightarrow \frac{s \vdash f_1}{t \vdash f_2} \in W$$

Let $CIPT$ be obtained by collapsing the isomorphic pseudo tableau IPT. Since

$$\frac{s \vdash f_1}{t \vdash f_2} \in IPT \Leftrightarrow \frac{s \vdash \overline{f_1}}{t \vdash \overline{f_2}} \in CIPT$$

it suffices to show that

$$\frac{s \vdash f_1}{t \vdash f_2} \in IPT \Leftrightarrow \frac{s \vdash \overline{f_1}}{t \vdash \overline{f_2}} \in W$$

We use the following induction hypothesis: $c(s \vdash f)$ constructs a proof tree which contains all proof trees for $s \vdash f_i$ with $\overline{f_i} = \overline{f}$ as isomorphic subtrees.

The rules for constructing IPT and W are the same. Also the choices are made in the same way. As a consequence, the only difference can occur when $c(s \vdash f)$ reaches a vertex with the label $t \vdash g$ again. c would stop in this case whereas the construction for some of the $t \vdash g_i$ with $\overline{g_i} = g$ in IPT might go on. However, $t \vdash g$ could only have been added by $c(t \vdash g)$ which already ensures by the induction hypothesis that all proof trees for $t \vdash g_i$ are contained as isomorphic subtrees in W.

■

Theorem 8 *Algorithm 2 has time complexity* $O(|f| \cdot |M|)$.

Proof: Obvious.

■

Note, however, that information from prior symbolic model checking is needed for Algorithm 2 to work. Therefore, the total complexity of constructing a collapsed IPT is the sum of these two complexities.

6 Super-collapsed isomorphic pseudo tableau

Definition 13 (Super-collapsed isomorphic pseudo tableau (SCIPT)) *Let $T_{\equiv} = (V_{\equiv}, E_{\equiv}, v_{\equiv})$ be a collapsed isomorphic pseudo tableau. We define the following equivalence relation on $V_{\equiv}$:*

$$\forall [i], [j] \in V_{\equiv} : [i] \rightleftharpoons [j] \Leftrightarrow (v([i]) = s \vdash f) \wedge (v[j] = t \vdash g) \wedge (s = t)$$

$\lfloor i \rfloor = \{[j] | [i] \rightleftharpoons [j]\}$ denotes the equivalence class induced by $\rightleftharpoons$. Then $T_{\rightleftharpoons} = (V_{\rightleftharpoons}, E_{\rightleftharpoons}, v_{\rightleftharpoons})$ is the corresponding super-collapsed isomorphic pseudo tableau where

$$V_{\rightleftharpoons} = \{\lfloor i \rfloor | [i] \in V_{\equiv}\}$$

$$E_{\rightleftharpoons} = \{(\lfloor i \rfloor, \lfloor j \rfloor) | \exists [k], [l] \in V_{\equiv} : [k] \in \lfloor i \rfloor \wedge [l] \in \lfloor j \rfloor \wedge ([k], [l]) \in E_{\equiv}\}$$

$$v_{\rightleftharpoons} : V_{\rightleftharpoons} \rightarrow \mathcal{P}(Seq)$$

$$v_{\rightleftharpoons}(\lfloor i \rfloor) = \{v_{\equiv}([j]) | [j] \in \lfloor i \rfloor\}$$

Lemma 5 *Let $T_{\rightleftharpoons} = (V_{\rightleftharpoons}, E_{\rightleftharpoons}, v_{\rightleftharpoons})$ be the super-collapsed isomorphic pseudo tableau for $s \vdash f$ in the model $M = (S, R, L)$. Then $|V_{\rightleftharpoons}| \leq |S|$.*

We can easily adapt the direct algorithm to one which constructs a super-collapsed IPT. In the following algorithm, we identify vertices with states.

Algorithm 3 (Direct SCIPT construction)
procedure $c(s \vdash f^{\vec{x}})$
begin
  if $s \vdash f^{\vec{x}} \in v(s)$ **then return**
  else
    begin
      $v(s) := v(s) \cup \{s \vdash f^{\vec{x}}\}$
      case f *of the form*
        $P, \neg P$: **return**;
        $p \wedge q$: $c(s \vdash min(s, p)); c(s \vdash min(s, q))$;
        $p \vee q$: $u :=$ *choose*$(s \vdash f^{\vec{x}})$; $c(u)$;
        $\langle\rangle p$: *let* $s' \vdash p^{\vec{y}} =$*choose*$(s \vdash f^{\vec{x}})$;
            if $s \neq s'$ **then begin** $V := V \cup \{s'\}; E := E \cup \{(s, s')\}$ **end**;
            $c(s' \vdash p^{\vec{y}})$;
        $[\,]p$: **for all** s' *with* $(s, s') \in R$ **do**
            begin
              if $s \neq s'$ **then begin** $V := V \cup \{s'\}; E := E \cup \{(s, s')\}$ **end**;
              $c(s' \vdash min(s, p))$
            end;
        $\sigma X.p(X)$: $c(s \vdash min(s, X))$;

$X : c(s \vdash min(s, b'(X)))$;
esac
end
end

$V := \{s\}; E := \emptyset$; **for all** $s \in S$ **do** $v(s) := \emptyset$; $c(s \vdash min(s, f))$;
for all $s \in V$ **for all** $p^{\vec{x}} \in v(s)$: *strip off* $\vec{x}$ *from* $p^{\vec{x}}$;

Theorem 9 *The tableau W constructed for* $s \vdash f$ *by Algorithm 3 is identical to the super collapsed isomorphic pseudo tableau* $SCIPT$ *for* $s \vdash f$ *provided that* choose *chooses in the same way.*

Proof: Let $SCIPT$ be obtained by super-collapsing the isomorphic pseudo tableau IPT. It suffices to show that

$$\frac{s \vdash f_1}{t \vdash f_2} \in IPT \Leftrightarrow \frac{s \vdash \overline{f_1}}{t \vdash \overline{f_2}} \in W$$

if $s \neq t$ and

$$\frac{s \vdash f_1}{s \vdash f_2} \in IPT \Leftrightarrow \{s \vdash \overline{f_1}, s \vdash \overline{f_2}\} \in v(s)$$

We use the following induction hypothesis: $c(s \vdash f)$ constructs a proof tree which contains all proof trees for $s \vdash f_i$ with $\overline{f_i} = \overline{f}$ as subtrees isomorphic to these except that subsequent vertices i, j with $v(i) = s \vdash p$ and $v(j) = s \vdash q$ are collapsed into one vertex.

The rules for constructing IPT and W are the same except for the wanted difference just described. Also the choices are made in the same way. As a consequence, the only difference can occur when a call $c(t \vdash g)$ reaches a vertex t with $(t \vdash g) \in v(t)$ again. c would stop in this case whereas the construction for some of the $t \vdash g_i$ with $\overline{g_i} = g$ in IPT might go on. However, $t \vdash g$ could only have been added by $c(t \vdash g)$ which already ensures by the induction hypothesis that all proof trees for $t \vdash g_i$ are contained as super-collapsed subtrees in W.

■

Theorem 10 *Algorithm 2 has time complexity* $O(|f| \cdot |M|)$.

Proof: Obvious.

■

In fact, this algorithm is more or less equivalent to the algorithm presented in [Kic95].

It also turns out that the definition of super-collapsed IPT is equivalent to the direct definition of witness in [Kic95].

7 Conclusions

In this paper, we have derived in a formal way a definition of witness for the truth of μ-calculus formulae in a given model $M = (S, R, L)$ from the definition of tableaux. This paper also shows that the more ad hoc definition given in [Kic95] is equivalent to the formally derived definition of witness in this paper.

References

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293 – 318, September 1992.

[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.

[CGMZ94] E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. Technical Report CMU-CS-94-204, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, October 1994.

[Cle90] Rance Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Inf.*, 27:725–747, 1990.

[EL86] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *IEEE Symposium on Logic in Computer Science*, pages 267–278, 1986.

[Kic95] A. Kick. Tableaux and witnesses for the μ-calculus. Technical Report 44/95, Faculty of Computer Science, University of Karlsruhe, D-76128 Karlsruhe, Germany, October 1995.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[McM93] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Boston,USA, 1993.

[SW91] Stirling and Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89, 1991.