Tableaux and witnesses for the μ-calculus

October 7, 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 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.

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

*Supported by DFG Vo 287/5-2

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. The notion of tableau motivates the definition of witness in Section 4 where we also present an algorithm to construct witnesses. 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. In Section 5 we compare our witness construction for the whole μ-calculus to the special witness construction for FCTL formulae in [CGMZ94]. In Section 6 we draw some conclusions.

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 $\rho[X/T]$ is defined by

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

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

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

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 $b(X) = \sigma X.p(X)$ if the latter formula 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: $(x) < (y) \Leftrightarrow x < y$, $(x_1, \ldots, x_m) < (y_1, \ldots, y_m) \Leftrightarrow x_1 < y_1 \vee x_1 = y_1 \wedge (x_2, \ldots, x_m) < (y_2, \ldots, y_m)$. This vector shall denote the values of the iterations of the μ-variables 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, $mc(f, \vec{x})$ determines the set of states of the model which fulfill f.

function *mc(f:Predicate, $\vec{x}$:$\mathbb{N}_0^m$): Predicate*
begin
  case *f of the form*
    X^j : $S' := S^j$;
    P : $S' := \{s | P \in L(s)\}$;
    $p \wedge q$: $S' := mc(p, \vec{x}) \cap mc(q, \vec{x})$;
    $p \vee q$: $S' := mc(p, \vec{x}) \cup mc(q, \vec{x})$;
    $\neg p$: $S' := S \setminus mc(p, \vec{x})$;

```
        ⟨⟩p          : S' := {s ∈ S | ∃t ∈ mc(p, x⃗) : (s,t) ∈ R};
        [ ]p         : S* = mc(p, x⃗); S' := {s ∈ S | ∀t ∈ S : (s,t) ∈ R → t ∈ S*};
        μX^j.p_j(X) :
            begin
                S^j := ∅;
                i := 0;
                repeat
                    S' := S^j;
                    S^j := mc(p_j, (x_1, ..., x_{j-1}, i, ..., x_m));
                    i := i + 1;
                until S' = S^j;
            end
        νX^j.p_j(X) :
            begin
                S^j := S;
                repeat
                    for all g ≺ νX^j.p_j(X) for all x⃗ : g_x⃗ := ∅
                    S' := S^j;
                    S^j := mc(p_j, x⃗);
                until S' = S^j;
            end
    esac
    f_x⃗ := S';
    return S'
end

for all p ⪯ f for all x⃗ : p_x⃗ := ∅;
mc(f, (0, ..., 0));
```

Definition 1
In the following, let $p \preceq f$, $p_{\vec{x}}$ *obtained by* $mc(f,(0,\dots,0))$ *where the model is* $M=(S,R,L)$, $s \in S$ *and* $\vec{x} \in \mathbb{N}_0^m$.

- $\forall p \preceq f$: $p \neq \mu X.q \rightarrow \forall \vec{x}$:
  $(p^{(x_1,\dots,x_j+1,\dots)} = p_{(x_1,\dots,x_j+1,\dots)} \backslash p_{(x_1,\dots,x_j,\dots)}) \wedge (p^{(x_1,\dots,x_{j-1},0,\dots)} = p_{(x_1,\dots,x_{j-1},0,\dots)})$
  $\forall \mu X^j.p^j \preceq f \forall \vec{x}$: $(\mu X^j.p^j)^{\vec{x}} = (p^j)^{\vec{x}}$

- $min\colon S \times L_\mu \rightarrow L_\mu \times \mathbb{N}_0^m \cup \{\bot\}$
  $$min(s,p) = \begin{cases} p^{min\{\vec{y} \mid s \in p^{\vec{y}}\}} & s \models p \\ \bot & otherwise \end{cases}$$
  $v\colon L_\mu \times \mathbb{N}_0^m \cup \{\bot\} \rightarrow \mathbb{N}_0^m \cup \{\infty\}$
  $$v(g) = \begin{cases} \vec{x} & g = p^{\vec{x}} \\ \infty & g = \bot \end{cases}$$

In the following, let $\forall \vec{x} \in \mathbb{N}_0^m : \vec{x} < \infty$.

- $\forall p \preceq f : l(s,p) = (s \in \bigvee_{\vec{x}} p^{\vec{x}})$

Lemma 1 *Let* $p \wedge q, p \vee q, \langle\rangle p, [\,]p$ *be subformulae of formula* f *model checked by the above algorithm and* $s \in S$ *arbitrary with* $s \models p \wedge q, p \vee q, \ldots$, *respectively, in the items below. Then*

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

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* $\mu X^j.p(X)$*:*

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

and in particular

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

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.

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 $s \models p$ 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 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.

3.1 Tableau construction

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

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 $U_1, \ldots, U_{i-1}$. $\Delta.(U = A)$ means appending $U = A$ to the definition list Δ. Definition lists are used to keep track of the "dynamically changing" subformulae as fixpoints are unrolled.

Definition 2 (Tableau system MC)

$$\frac{s \vdash_\Delta \neg\neg p}{s \vdash_\Delta p}$$

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

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

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

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

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

$$\frac{s \vdash_\Delta U}{s \vdash_\Delta p[Z := U]} \quad \mathcal{C} \text{ and } \Delta(U) = \sigma Z.p$$

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. 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 *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 $t \vdash_{\Delta'} U$.

Theorem 2 (Stirling, Walker) *$s \vdash f$ has a successful tableau if and only if $s \models f$.*

The tableau rules work according to the semantic definition of the operators. The only interesting cases are $\sigma X.p(X)$. 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.

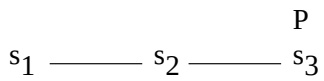

Figure 1: A model

3.2 An example

For the model in Figure 1 a tableau for $s_1 \vdash \mu X.P \vee \mu Y.P \vee \langle\rangle(X \wedge Y)$ is developed in Figure 2. In this tableau the follwing abbreviations are used:
$\Delta_1 = (U = \mu X.P \vee \mu Y.P \vee \langle\rangle(X \wedge Y))$
$\Delta_2 = \Delta_1.(V = \mu Y.P \vee \langle\rangle(U \wedge Y))$
$\Delta_3 = \Delta_2.(W = \mu Y.P \vee \langle\rangle(U \wedge Y))$

The above example shows that tableaux constructed by the method described in [SW91] can be exponentially large. In the example, the subtree below $s_2 \vdash_{\Delta_2} V$ is identical to the subtree below $s_2 \vdash_{\Delta_3} W$ if we replace V by W. We can save the user who wants to use a witness to find an error a great deal of work if identical subtrees are not demonstrated in the witness. In this case, we also do not need any declarations and definition lists. This reasoning leads to the definition of witness in the following section.

4 Witness generation

Definition 3 (Abstract witness)
An abstract witness $W_{s,f}$ for $s \models_M f$, where $M = (S, R, L)$, $s \in S, f \in L_\mu$, is a triple (V, E, m) where $V \subseteq S$, $E \subseteq R$ and $m\colon V \to \mathcal{P}(L_\mu)$. For given s, f and M the components V, E and m of the abstract witness are inductively defined as follows:

1. $s \in V$, $f \in m(s)$
2. (a) $p \wedge q \in m(s)$ *implies* $p \in m(s)$ *and* $q \in m(s)$
   (b) $p \vee q \in m(s)$ *implies:*
   if $s \models p$ *and* $s \models q$ *then* $p \in m(s)$ *or* $q \in m(s)$
   if $s \models p$ *and* $s \not\models q$ *then* $p \in m(s)$
   if $s \not\models p$ *and* $s \models q$ *then* $q \in m(s)$
   (c) $\langle\rangle p$ *implies: for an arbitrary* $s' \in \{s''|(s, s'') \in R \wedge s'' \models p\}$: $s' \in V$, $p \in m(s')$, $(s, s') \in E$
   (d) $[\,]p$ *implies: for all* $s' \in \{s''|(s, s'') \in R\}$: $s' \in V$, $p \in m(s')$, $(s, s') \in E$
   (e) $\sigma X.p(X) \in m(s)$ *implies* $p(X) \in m(s)$
   (f) $X \in m(s)$ *implies* $b(X) \in m(s)$

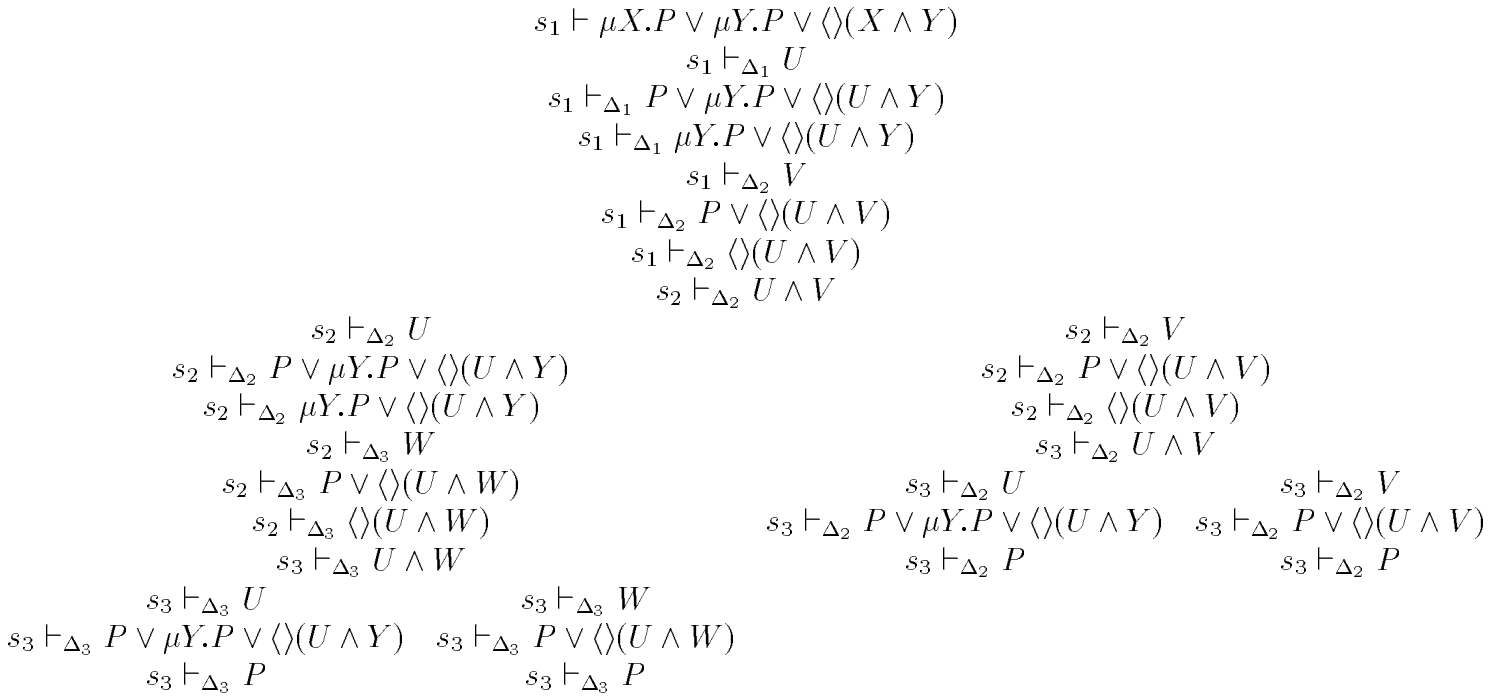

Figure 2: A tableau for $s_1 \vdash \mu X.P \vee \mu Y.P \vee \langle\rangle(X \wedge Y)$

3. *No other states, edges and formulae belong to V, E and $m(s)$, s arbitrary, respectively.*

Definition 3 is motivated by the premise that we need to demonstrate $s \models p$ for a formula p and a fixed state s just once.

Definition 3 does not ensure that the μ-paths - paths produced by subsequent unwinding of $\mu X.p$ (Definition 5) - are dealt with properly. For this reason, we define 'concrete witnesses' in Definition 6.

In the following definition, we define the intermediate paths between subsequent states which model $\mu X.p$ on a μ-path.

Definition 4 (Xpath)
For a witness $W = (V, E, m)$ and model M, π is an Xpath for a subformula g of a formula f and a propositional variable X if $Xp(\pi, g, X)$ where

$$Xp\colon S^* \times L_\mu \times PropVar \to \{true, false\}$$

$$Xp(s\pi, p \wedge q, X) \Leftrightarrow (p \wedge q) \in m(s) \wedge (Xp(s\pi, p, X) \vee Xp(s\pi, q, X))$$

$$Xp(s\pi, p \vee q, X) \Leftrightarrow (p \vee q) \in m(s) \wedge (p \in m(s) \wedge Xp(s\pi, p, X) \vee q \in m(s) \wedge Xp(s\pi, q, X))$$

$$Xp(ss'\pi, \Box p, X) \Leftrightarrow \Box p \in m(s) \wedge (s, s') \in R \wedge Xp(s'\pi, p, X)$$

$$Xp(s\pi, \sigma Y.p, X) \Leftrightarrow \sigma Y.p \in m(s) \wedge Xp(s\pi, p, X)$$

$$(Y \not\equiv X) \wedge (b(Y) \prec \mu X.p) \to [Xp(s\pi, Y, X) \Leftrightarrow Y \in m(s) \wedge Xp(s\pi, b(Y), X)]$$

$$(Y \not\equiv X) \wedge (b(Y) \not\prec \mu X.p) \to [Xp(s\pi, Y, X) = false]$$

$$Xp(s, X, X) \Leftrightarrow X \in m(s)$$

Definition 5 (μ-path in a witness)
A μ-path π in a witness $W = (V, E, m)$ for a formula $\mu X.p(X)$ is a finite sequence of states $s_0 s_1 \ldots s_m$ with $(\forall 0 \leq i \leq m\colon s_i \in V) \wedge \mu X.p(X) \in m(s_0) \wedge (\forall 0 \leq i < m\colon \exists \rho = \rho_0 \ldots \rho_n\colon Xp(\rho, \mu X.p(X), X) \wedge \rho_0 = s_i \wedge \rho_n = s_{i+1})$. π_i shall denote the ith state in the μ-path. The set of all μ-paths in a witness W for a formula $\mu X.p(X)$ is denoted by $Mp(\mu X.p(X))$.

Definition 6 (Concrete witness)
A concrete witness for $s \models_M f$, $M = (S, R, L)$, $s \in S$, $f \in L_\mu$ is an abstract witness $W = (V, E, m)$ for $s \models_M f$ with the additional constraint

$$\forall \mu X.p(X) \preceq f \forall s \in V\colon \mu X.p(X) \in m(s) \to \exists \pi = \pi_0 \ldots \pi_n \in Mp(\mu X.p(X))\colon \pi_0 = s \wedge p(false) \in m(\pi_n) \quad (1)$$

Algorithm 2 (Concrete witness generation)
procedure $c(s : S, f^{\vec{x}} : L_\mu)$
begin
  if $f^{\vec{x}} \in m(s)$ **then return**
  else
    begin
      $m(s) := m(s) \cup \{f^{\vec{x}}\}$
      case $f^{\vec{x}}$ *of the form*
        $P, \neg P$ *:* **return**;
        $p \wedge q$ *:* $c(s, min(s, p)); c(s, min(s, q))$;
        $p \vee q$ *:* **if** $p \in m(s)$ *or* $q \in m(s)$ **then return**;
              if $s \models p$ **then** $c(s, min(s, p))$ **else** $c(s, min(s, q))$;
        $\langle\rangle p$ *: let* $\vec{y} = min\{\vec{z} | \exists s'\colon (s, s') \in R \wedge s' \in p^{\vec{z}}\}$;
            let s' *be such that* $s' \in p^{\vec{y}} \wedge (s, s') \in R$;
            $V := V \cup \{s'\}; E := E \cup \{(s, s')\}$; $c(s', p^{\vec{y}})$;
        $[\,]p$ *:* **for all** s' *with* $(s, s') \in R$ **do**
              begin
                $V := V \cup \{s'\}; E := E \cup \{(s, s')\}; c(s', min(s, p))$;
              end
        $\sigma X.p(X)$ *:* $c(s,min(s,p(X)))$;
        X *:* $c(s,min(s,b(X)))$;
      esac
    end
end

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

Theorem 3 *Algorithm 2 constructs a concrete witness for* $s \models_M f$ *and terminates.*

Proof: Algorithm 2 terminates:
At each call of c(s,p) the procedure c either stops if $p \in m(s)$ or p is added to $m(s)$. Since the number of calls of c in the body of c is limited the algorithm terminates latest when $\forall s \in S \forall p \preceq f\colon p \in m(s)$ and all outstanding calls of c have been performed.

Algorithm 2 fulfills Condition 1 of Definition 3 since after the call of c(s,f) f will have been added to $m(s)$. For arbitrary s, p, a call of c(s,p) also ensures Conditions 2 and 3 of Definition 3. In the case of $p \in m(s)$, p could only have been added to $m(s)$ by a call c(s,p), the latter already ensuring Conditions 2 and 3. In the case of $p \notin m(s)$, p will be added to $m(s)$ and the cases in the body of Algorithm 2 ensure Conditions 2 and 3.

If p is of the form $q \wedge r$ then the calls of c(s,q) and c(s,r) will add p, q to $m(s)$

if they do not already belong to $m(s)$ and ensure that for s, p and s, q Conditions 2 and 3 are fulfilled. The argumentation for the other cases is similar.

Condition 3 holds since no other states, edges and formulae are added by the calls c(s,p) to V, E, and $m(s)$, s arbitrary, than according to Definition 3.

In Algorithm 2 we use the knowledge from prior model checking of a formula to construct a concrete witness for it. When the algorithm is at a state s with $\mu X.p$ newly added to $m(s)$ the algorithm walks along an Xpath until a variable X is reached at a state t. Since the algorithm always proceeds along $min(s, p)$ going down the subformulae we can conclude from Lemma 1 that for X at state t, $v(X)$ is smaller than or equal to $v(\mu X.p) = v(p)$ at state s. At state t, X is unrolled to $\mu X.p$ and then to p. From Fact 1 it then follows that the measure $v(p)$ decreases at such subsequent states s and t. It follows directly from the definition of min that not only such subsequent states s and t are therefore different but the whole chain of such states at the beginning and end of such Xpaths. I.e., this process will not lead to a loop.

The process of unrolling can therefore only stop when $p(false)$ is finally reached or when already $q^{\vec{x}} \in m(s)$ with $q \preceq \mu X.p$ at a state s. In the latter case, we can use the induction hypothesis which ensures that further unrolling leads to a state fulfilling $p(false)$. Therefore, Condition (1) of Definition 6 is also fulfilled.

■

Theorem 4 *Algorithm 2 has time complexity* $O(|f| \cdot |M|)$.

Proof: For each subformula of f and each state $s \in S$, the body of Algorithm 2 apart from the test $f \in m(s)$ is executed at most once because of the test $f \in m(s)$. As a consequence, there are at most $|S| \cdot |f|$ executions of the body. Consequently, for each state the outgoing edges are also considered at most once for each subformula in the cases $\langle\rangle p$ and $[\,]p$. Therefore, the test $p \in m(s)$ can be done only $|E| \cdot |f|$ times. This leads to the total complexity of $O(|M| \cdot |f|)$.

Note that the test $s \in p^{\vec{x}}$ can be done in constant time if there is a hash table for $s, p \preceq f$ computed during model checking of $s \models f$, similarly for the test $f \in m(s)$.

■

This theorem shows that it makes sense to use the *faster* symbolic model checking (compared to local model checking if the locality condition does not hold) to guide the construction of a *small* witness.

5 Comparison to witness construction for FCTL

FCTL is a subclass of L_{μ_2}. The witness construction for FCTL in [CGMZ94] are for the special type of L_{μ_2} formulae of the form $\nu Z.[f \wedge \bigwedge_{k=1}^{n} \langle\rangle[\mu X.Z \wedge h_k \vee (f \wedge \langle\rangle X)]]$. We repeat their witness construction in a more formal way in Algortihm 3. The algorithm in [CGMZ94] would construct a single path with a cycle in which all fairness constraints h_k are contained in contrast to Algorithm 2 which would construct a path for each separate conjunct $\langle\rangle[\mu X.Z \wedge h_k \vee (f \wedge \langle\rangle X)]]$ whenever a state s for which $s \models \nu Z. \ldots$ has to be demonstrated is reached.

Algorithm 3 (Witness generation for FCTL formulae [CGMZ94])
We use the abbreviations $G_k = \mu X.Z \wedge h_k \vee (f \wedge \langle\rangle X)$ *and* $\forall i \in \mathbb{N}_0 : G_k^i = G_k^{(p_1,\ldots,p_k,\ldots,p_n)}$ *where* $p_k = i$ *and* $\forall 0 \leq j \leq n : j \neq k \rightarrow p_j = 0$
input: start state s *and FCTL formula* $F = \nu Z.[f \wedge \bigwedge_{k=1}^{n} \langle\rangle[\mu X.Z \wedge h_k \vee (f \wedge \langle\rangle X)]]$
output: witness for $s \models F$

$V := \{s\}; E := \emptyset; t := s;$
repeat
  $C := \{1, \ldots, n\};$
  while $C \neq \emptyset$ **do**
    begin
      let $s' \in G_k^{min\{i | s'' \in G_k^i \wedge k \in C \wedge (t,s'') \in R\}} \wedge k \in C \wedge (t, s') \in R;$
      $V := V \cup \{s'\}; E := E \cup \{(t, s')\};$
      $C := C \setminus \{k\}; t := s';$
      while $\neg l(t, Z \wedge h_k)$ **do**
        begin
          let $s' \in G_k^{min\{i | s'' \in G_k^i \wedge (t,s'') \in R\}} \wedge (t, s') \in R;$
          $V := V \cup \{s'\}; E := E \cup \{(t, s')\};$
          $t := s';$
        end
    end
  $mc(\mu X.s \vee f \wedge \langle\rangle X, (0));$
  $ct := l(t, \mu X.s \vee f \wedge \langle\rangle X);$
  if $ct = true$ **then**
    while $t \neq s$ **do**
      begin
        let $s' \in (\mu X.s \vee f \wedge \langle\rangle X)^{min\{(i) | (t,s'') \in R \wedge s'' \in (\mu X.s \vee f \wedge \langle\rangle X)^{(i)}\}} \wedge (t, s') \in R;$
        $V := V \cup \{s'\}; E := E \cup \{(t, s')\};$
        $t := s'$
      end
  else $s := t$
until ct

In [CGMZ94], the last approximations for the G_k are saved in the same way as in the modified model checking algorithm presented in this paper. In fact, the witness construction works in the same way as our general algorithm for witness construction for the μ-calculus, however with 2 significant differences. First, when the witness construction is at a state $s \models \bigwedge_{k=1}^{n} \langle\rangle[\mu X.Z \wedge h_k \vee (f \wedge \langle\rangle X)]$ the witness for exactly one conjunct is constructed in contrast to the general μ-calculus witness construction where each conjunct would be developed. Second, when a concatenated sequence of witnesses for each conjunct was developed a path back to the starting point is built if possible. If not possible the procedure restarts. To check whether there is a path back to the beginning requires additional model checking. This is not necessary in the general witness construction for the μ-calculus.

In the witness construction in [CGMZ94] for this special type of FCTL formulae the particular intended meaning of the FCTL formula is exploited: the fairness constraints h_k occur infinitely often on a path. Therefore, their counterexample construction does not extend to the whole μ-calculus. However, their special purpose witness construction will in general be preferable for FCTL since, probably in most cases, their construction will be smaller than the general witness construction for the μ-calculus.

6 Conclusions

We have shown how to construct counterexamples and witnesses for the whole μ-calculus. This eliminates the most important disadvantage of μ-calculus model checkers and allows a much more general approach to model checking than usual CTL model checkers.

In summary, a tableau for $s \vdash f$ can be regarded as a witness for $s \models f$. However, tableau construction is expensive and a tableau can be exponential in the model. We motivated our definition of witness as a kind of tableau where redundant information in the tableau is hidden from the user.

In order to construct a concrete witness information obtained during symbolic model checking can be used. The construction of a concrete witness can be performed in time polynomial in the model and the formula. The construction of concrete witnesses for μ-calculus expressions is polynomial in $|f|$ and $|M|$ in contrast to model checking f which is exponential in the alternation depth of f. As a consequence, the construction of witnesses is only a minor factor in the verification of reactive systems. If the locality condition does not hold then compared to tableau model checking, the witness can be constructed faster because of the faster symbolic model checking and only additional polynomial

expense. At the same time, the witness will be much smaller and more easily to understand.

As is clear from the special witness construction for FCTL formulae in [CGMZ94] special witness generation algorithms for subclasses of the μ-calculus which exploit the intended meaning of the formulae in the particular subclass can be advantageous compared to the general witness construction algorithm (Algorithm 2) presented in this paper.

We advocate an even more general and more flexible approach to the generation of witnesses for the μ-calculus than the one presented in this paper. The user of a model checker should decide on which paths of the witness to be constructed. Interactive generation of witnesses would allow the user of the verification tool to pursue the paths of the witness in which he/she is mainly interested.

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.

[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.