Generation of counterexamples and witnesses for the μ-calculus

August 23, 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 to fair CTL formulae. Model checkers constructed just recently can check arbitrary μ-calculus formulae. How to construct counterexamples for arbitrary μ-calculus formulae has not been investigated yet. This paper shows how counterexamples and witnesses for the whole μ-calculus can be constructed.

*Supported by DFG Vo 287/5-2

Contents

1 Introduction

Complex state-tansitions systems occur frequently in the design of sequential circuits and protocols. Symbolic temporal logic model checking 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. 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]) can check fair CTL (FCTL) ([CGL93]) formulae and construct counterexamples for these formulae. Model checkers constructed just recently ([Rau95], [Cle93], and also at Aalborg and Eindhoven University, etc.) can check μ-calculus formulae [Koz83], [EL86]. At Karlsruhe University, a model checker has been implemented [Bie95a] which is based on OBDDs ([Bry86], [Bry92]) in contrast to [Cle93] and allows - in contrast to [Rau95] where only μ-calculus formulae of alternation depth 2 are allowed - arbitrary μ-calculus formulae to be checked automatically. The model checker described in [Bie95a] can easily be extended to arbitrary interface languages, thus allowing different descriptions of the model and the specification. Its modular design also allows various representations (not just OBDDs, the usual representation for the model) to be used. At the same time this model checker has greater expressive power, since μ-calculus formulae can be checked in contrast to the small subclass FCTL of the μ-calculus.

In [CGMZ94], [CGL93] 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 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. We show that this is possible in general, but computationally expensive for some cases.

The rest of the paper is structured as follows. Section 2 consists of preliminaries where the μ-calculus is repeated and some terminology is introduced. Section 3 reminds the reader of how to construct counterexamples for FCTL formulae. In Section 4, we suggest some algorithms for witness construction for μ-calculus formulae. Sections 5 and 6 show how main paths can be constructed for formulae of type $\mu X.p(X)$ and of type $\nu X.p(X)$, respectively. Section 7 compares our witness construction for the whole μ-calculus to the witness construction in [CGMZ94]. In Section 8, we give a short summary and 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 normal forms for formulae and finally give a modified model checking algorithm which suits our purposes of witness construction.

2.1 Syntax

In summarizing syntax and semantics of the modal μ-calculus we mainly follow [Zuc93a].

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 $\neg\langle A\rangle\neg p$
- $\nu X.p(X)$ abbreviates $\neg\mu X.\neg p(\neg X)$

2.2 Semantics

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: $ProgAt \rightarrow \mathcal{P}(S \times S)$ 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 $\forall i \geq 0 : (s_i, s_{i+1}) \in R$.

Finiteness assumption: 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] and [Bie95b].

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

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

2.3 Some terminology, notation and functions to handle μ-calculus expressions

$\langle\rangle$ shall stand for any $\langle A\rangle$, $[\,]$ for any $[A]$. $p, \ldots$ shall stand for any formulae which do not contain X (which can comprise μ and ν subformulae) and $p(X), \ldots$ which do.

The terms *subformula, closed formula, bound* and *free variables* are used as usual ([EL86]). 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 *$\langle\rangle$-, $[\,]$-subformula*

is a subformula whose main connective is $\langle\rangle$ and $[\,]$, respectively. We say that q is a *top-level* $\langle\rangle$-, $[\,]$-*subformula* of p provided that q is a $\langle\rangle$-, $[\,]$-subformula of p but not a proper $\langle\rangle$-, $[\,]$-subformula of any other $\langle\rangle$-, $[\,]$-subformula of p, respectively. The terms *top-level* μ-, ν-*subformula* are defined analagously.

Alternation depth is defined in [EL86]. L_{μ_i} shall denote the sublanguage of L_μ with alternation depth i.

The formulae $\mu X.p(X)$ and $\nu X.p(X)$ have *iteration depth 1*, denoted by $\mathcal{I}(\mu X.p(X)) = 1, \mathcal{I}(\nu X.p(X)) = 1$, if all proper μ- and ν-subformulae do not contain variable X (supposing that no variables are quantified twice).

The function $occ(X, f)$ shall return $true$ if X occurs in formula f and $false$ otherwise.

$\sigma X.p(X)$ shall stand for either $\mu X.p(X)$ or $\nu X.p(X)$, $\Box$ shall stand for either $[\,]$ or $\langle\rangle$.

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.

Example 2.1 *In Formula* $\mu X.(P \vee [A]Q \vee (\nu Y.Q \wedge [A]Y) \vee \langle A\rangle X)$, X *is not in the scope of* $[\,]$. *However,* X *is in the scope of* $[\,]$ *in the formula* $\mu X.P \vee [A]X$.

We define the depth of an occurrence of variable X in $\sigma X.p(X)$ with $\mathcal{I}(\sigma X.p(X)) = 1$ as the number of $\Box$ within which X occurs. Since there can be an arbitrary number of Xs in $p(X)$ we suppose the different occurrences of X are uniquely labeled, e.g., X^j. $l\colon L_\mu \times \mathbb{N} \times PropVar \to L_\mu$ labels each such X^j with its depth m: $X^{j,m}$.

$l(P, i, X) = P$
$l(Y, i, X) = Y$
$l(\sigma Y.q(Y), i, X) = \sigma Y.q(Y)$
$l(\sigma X.p(X), i, X) = \sigma X.l(p(X), i, X)$
$l(p \wedge q, i, X) = l(p, i, X) \wedge l(q, i, X)$
$l(p \vee q, i, X) = l(p, i, X) \vee l(q, i, X)$
$l(\Box q, i, X) = \Box l(q, i + 1, X)$
$l(\neg q, i, X) = \neg l(q, i, X)$
$l(X^j, i, X) = X^{j,i}$

To label the different occurrences of a variable X in a formula f we start the labeling with $l(f, 0, X)$.

The *smallest depth* of a formula $p(X)$ is defined to be the smallest depth of the depths of all occurrences of X, denoted by $d(p(X), X)$.

Example 2.2 $l(\mu X.P \vee \langle\rangle(X \wedge \nu Y.Q \wedge \langle\rangle[\,]Y) \vee \langle\rangle[\,](X \wedge \langle\rangle X), 0, X) =$
$\mu X.P \vee \langle\rangle(X^{1,1} \wedge \nu Y.Q \wedge \langle\rangle[\,]Y) \vee \langle\rangle[\,](X^{2,1} \wedge \langle\rangle X^{3,2})$
$d(\mu X.P \vee \langle\rangle(X^{1,1} \wedge \nu Y.Q \wedge \langle\rangle[\,]Y) \vee \langle\rangle[\,](X^{2,1} \wedge \langle\rangle X^{3,2}), X) = 2$

2.4 Elimination of X not in scope of $\langle\rangle$ or $[\,]$

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 ([SE89]).

Example 2.3

$$\mu X.(\neg(P \wedge [\,](\mu Y.\neg(X \wedge \langle\rangle\neg Y \vee \langle\rangle(Q \wedge [\,]R)))))$$

is transformed into its PNF

$$\mu X.(\neg P \vee \langle\rangle(\nu Y.(X \wedge \langle\rangle Y \vee \langle\rangle(Q \wedge [\,]R))))$$

Let all formulae $f \in L_\mu$ be in PNF in the rest of the paper.

A variable X in $\mu X.p(X)$ with $\mathcal{I}(\mu X.p(X)) = 1$ always appears in the scope of $\langle\rangle$ or $[\,]$. Otherwise, we transform the formula into a formula without such an occurrence of X. The same for $\nu X.p(X)$.

This becomes clear easily if $p(X)$ in $\mu X.p(X)$ or $\nu X.p(X)$ is in disjunctive normal.

This is intuitively clear when looking at the formula

$$\mu X.P \vee X \vee \langle\rangle X$$

which we claim to be equivalent to

$$\mu X.P \vee \langle\rangle X$$

The first fixpoint iteration yields P for both formulae.
The second fixpoint iteration yields $P \vee P \vee \langle\rangle P$ which is equivalent to $P \vee \langle\rangle P$. Subsequent iterations are also the same.

In the following two lemmas we suppose $p(X)$ in $\sigma X.p(X)$ to be in disjunctive normal form (treating $L_2 = \{P, \neg P, X, \mu Y.q(Y), \nu Y.q(Y), [\,]p, \langle\rangle p\}$ as the set of literals).

In the following lemma, we further suppose that X does not occur in p_l, X can occur in $p_k(X)$ and X occurs in $p_m(X)$.

Lemma 2.1

$$\mu X.(\bigvee_l p_l \vee X \wedge p_k(X) \vee \bigvee_m p_m(X)) = \mu X.(\bigvee_l p_l \vee \bigvee_m p_m(X))$$

Proof: Let l^n and r^n denote the nth fixpoint iteration of the left-hand and right-hand side, respectively. Since the least fixpoint iterations represent an increasing sequence, we have: $r^n \subseteq r^{n+1}$. We use this fact to prove the lemma by induction on the fixpoint iterations of both sides.

- Induction basis
  For both sides we obtain after the first iteration:
  $\bigvee_l p_l$

- Induction step (n+1st fixpoint iteration)
  $l^{n+1} = \bigvee_l p_l \vee r^n \wedge p_k(r^n) \vee \bigvee_m p_m(r^n)$
  using the induction hypothesis
  $= \bigvee_l p_l \vee \bigvee_m p_m(r^n))$
  since $q \wedge r^n \subseteq r^n \subseteq r^{n+1} = \bigvee_l p_l \vee \bigvee_m p_m(r^n))$, q arbitrary
  $= r^{n+1}$

■

In the following lemma, $p_i(X)$ may or may not contain X.

Lemma 2.2

$$\nu X.(X \wedge p_k(X) \vee \bigvee_m p_m(X)) = \nu X.(p_k(X) \vee \bigvee_m p_m(X))$$

Proof: Induction basis: clear
Induction step:
$l^{n+1} = r^n \wedge p_k(r^n) \vee \bigvee_m p_m(r^n)$
$= p_k(r^n) \vee \bigvee_m p_m(r^n)$
since $r^n \supseteq r^{n+1} \supseteq p_k(r^n)$
$= r^{n+1}$

■

If X does not occur in $p(X)$ of $\sigma X.p(X)$ then we can replace $\sigma X.p(X)$ by $p(X)$. $\nu X.X$ can be replaced by $true$ and $\mu X.X$ by $false$.

In the rest of the paper we therefore suppose that all μ-calculus formulae are in PNF and closed and for subformulae $\sigma X.p(X)$, X occurs in $p(X)$ and all occurrences of X in $p(X)$ are in the scope of $\langle\rangle$ or $[\,]$.

2.5 Driving in $\langle\rangle$ and Disjunctive Normal Form

In the following considerations we treat each μ-, ν-subformula of $p(X)$ and all top-level $[\,]$-subformulae as a literal, i.e., we look upon the set

$$L_1 = \{P, \neg P, X, \mu Y.q(Y), \nu Y.q(Y), [\,]p\}$$

as the set of literals.

Example 2.4 *In the formula*

$$\mu X.(P \vee [A]Q \vee (\nu Y.Q \wedge [A]Y) \vee \langle A\rangle X)$$

P, $[A]Q$ and $\nu Y.(Q \wedge [A]Y)$ are treated as literals.

The following are L_μ tautologies.

$$\langle\rangle(p \vee q) \Leftrightarrow \langle\rangle p \vee \langle\rangle q$$

$$\langle\rangle(p \wedge q) \Rightarrow \langle\rangle p \wedge \langle\rangle q$$

$$[\,](p \wedge q) \Leftrightarrow [\,]p \wedge [\,]q$$

$$[\,](p \vee q) \Leftarrow [\,]p \vee [\,]q$$

The first tautology allows us to drive $\langle\rangle$ as far into the formula as possible.

Example 2.5 *$\langle\rangle(X \vee (P \wedge \langle\rangle X))$ can be transformed into $\langle\rangle X \vee \langle\rangle(P \wedge \langle\rangle X)$.*

Example 2.6 *The formula*

$$\langle\rangle\langle\rangle(X \vee [\,](P \vee \langle\rangle X))$$

can be transformed into

$$\langle\rangle\langle\rangle X \vee \langle\rangle\langle\rangle[\,](P \vee \langle\rangle X)$$

Here, $[\,](P \vee \langle\rangle X)$ is treated as a literal.

More formally, E does this transformation of driving in $\langle\rangle$.

$$\forall l \in L_1 \colon E(l) = l$$

$$E(p \wedge q) = N(E(p) \wedge E(q))$$

$$E(p \vee q) = E(p) \vee E(q)$$

$$E(\langle\rangle p) = D(E(p))$$

$$\forall l \in L_2 \colon D(l) = \langle\rangle l$$

$$D(p \wedge q) = \langle\rangle(p \wedge q)$$

$$D(p \vee q) = (D(p) \vee D(q))$$

where $L_2 = \{P, \neg P, X, \mu Y.q(Y), \nu Y.q(Y), [\,]p, \langle\rangle p\}$

Example 2.7

$$
\begin{gathered}
E(\langle\rangle(P \wedge \langle\rangle((\mu Y.q(Y)) \vee \langle\rangle X))) = \\
D(E(P \wedge \langle\rangle((\mu Y.q(Y)) \vee \langle\rangle X))) = \\
D(N(E(P) \wedge E(\langle\rangle((\mu Y.q(Y)) \vee \langle\rangle X)))) = \\
D(N(P \wedge D(E(((\mu Y.q(Y)) \vee \langle\rangle X))))) = \\
D(N(P \wedge D(E(\mu Y.q(Y)) \vee E(\langle\rangle X)))) = \\
D(N(P \wedge D(E(\mu Y.q(Y)) \vee D(E(X))))) = \\
D(N(P \wedge D((\mu Y.q(Y)) \vee D(X)))) = \\
D(N(P \wedge D((\mu Y.q(Y)) \vee \langle\rangle X))) = \\
D(N(P \wedge (D((\mu Y.q(Y))) \vee D(\langle\rangle X)))) = \\
D(N(P \wedge (\langle\rangle(\mu Y.q(Y)) \vee \langle\rangle\langle\rangle X))) = \\
D(N(P \wedge (\langle\rangle(\mu Y.q(Y)) \vee \langle\rangle\langle\rangle X))) = \\
D(P \wedge \langle\rangle(\mu Y.q(Y)) \vee P \wedge \langle\rangle\langle\rangle X) = \\
\langle\rangle(P \wedge \langle\rangle(\mu Y.q(Y))) \vee \langle\rangle(P \wedge \langle\rangle\langle\rangle X)
\end{gathered}
$$

N transforms a formula into disjunctive normal form, where elements of L_2 are treated as literals.

Example 2.8 *In* $\mu X.((\langle\rangle(P \wedge \langle\rangle\mu Y.q(Y)) \vee \langle\rangle(P \wedge \langle\rangle X)) \wedge R)$ *the subformulae* $\langle\rangle(P \wedge \langle\rangle\mu Y.q(Y))$, $\langle\rangle(P \wedge \langle\rangle X)$ *and* R *are literals in* L_2. *The transformation into disjunctive normalform yields:*

$$\mu X.((\langle\rangle(P \wedge \langle\rangle\mu Y.q(Y)) \wedge R) \vee (\langle\rangle(P \wedge \langle\rangle X) \wedge R))$$

By driving in the $\langle\rangle$ operators we are able to dinstinguish between independent (defined in a later section) occurrences of X in $p(X)$ of $\sigma X.p(X)$ and interdependent ones, which we need later.

2.6 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.)

The model checking algorithm follows directly from the semantics of the μ-calculus (and the finiteness assumption of the model). We give here a modified version of the model checking procedure which saves information during model checking which we need for the later constuction of the witnesses.

Function mc computes the set of states which fulfill formula f, i.e. the set $[\![f]\!]\rho$, ρ arbitrary. Note that we identify predicates with sets of states.

Algorithm 2.1 *For a given model M and a given formula f which contains propositional variables $X^1, \ldots, X^m$, $mc(f)$ determines the set of states of the model which fulfill f.*

function *mc(f:Predicate): Predicate*
begin
  case *f of the form*
    X^j : $S' := S^j$;
    P : $S' := \{s | P \in L(s)\}$;
    $p \wedge q$: $S' := mc(p) \cap mc(q)$;
    $p \vee q$: $S' := mc(p) \cup mc(q)$;
    $\neg p$: $S' := S \setminus mc(p)$;
    $\langle\rangle p$: $S' := \{s | \exists t \in mc(p) : (s,t) \in R\}$;
    $[\,]p$: $S' := \{s | \forall t \in mc(p) : (s,t) \in R\}$;
    $\mu X^j.p_j(X)$:
      begin
        $S^j := \emptyset$;
        $i := 0$;
        repeat
          $S' := S^j$;
          $S^j := mc(p_j)$;
          $X_i := S^j$;
          $i := i + 1$;
        until $S' = S^j$;
        $n(X) := i \Leftrightarrow 1$;
      end
    $\nu X^j.p_j(X)$:
      begin
        $S^j := S$;
        repeat
          $S' := S^j$;
          $S^j := mc(p_j)$;
        until $S' = S^j$;
        $X_n := S'$;
      end
  esac
  $f^r := S'$;
  return S'
end

During model checking, some information which we will need later for witness construction is saved. We save

- the result of $mc(p)$ in p^r for each subformula of f

- the sequence of the fixpoint approximation of $\mu X.p(X)$ in $X_0, X_1, \ldots, X_{n(X)}$
- the last fixpoint approximation of $\nu X.p(X)$ in X_n

whenever $mc(p)$, $mc(\mu X.p(X))$ and $mc(\nu X.p(X))$, respectively, is applied.

In this way, when the model checking algorithm terminates, p^r contains the value of p with the variables bound to their last approximations; $X_0, X_1, \ldots, X_{n(X)}$ contain the last approximations of $\mu X.p(X)$; and X_n the final approximation of $\nu X.p(X)$.

3 Counterexamples and witnesses for FCTL

In this section, we remind the reader on how to construct witnesses for the possible types of CTL and FCTL formulae for which witness construction makes sense ($E[fUg], EGf$ and EGf under fairness constraints) except for the witness construction for $E[fUg]$ under fairness constraints which follows from the witness construction for EGf under fairness constraints. By doing this, we also present the witness construction algorithm in [CGMZ94] in a more formal way.

CTL and FCTL are subsets of the μ-calculus, in fact subsets of L_{μ_2}. We do not give the syntax of FCTL and CTL here. Instead, we give their translation into the μ-calculus, in this way giving a meaning to the (F)CTL formulae. For the precise definition of the syntax and semantics of these subclasses of the μ-calculus the reader is referred to [CGL93], [Zuc93b].

A counterexample for a formula can be constructed by constructing a witness for its negation. However, the negation has to belong to the formulae for which a reasonable witness can be constructed. E.g., constructing a witness for AGf would not be sensible. Because of the simple connection between counterexamples and witnesses we will only talk about witnesses in the rest of the paper.

3.1 Witness construction for $E[fUg]$

$E[fUg]$ can be rewritten into the μ-expressions $\mu X.g \vee (f \wedge \langle\rangle X)$. During the fixpoint iteration for this formula, which starts with $X_{-1} = false$, the approximations to X ($X_0 = g, X_1 = g \vee (f \wedge \langle\rangle g), X_2, \ldots$) are saved (cf. Algorithm 2.1).

In model checking we want to show that a formula f is fulfilled for a set of initial states I, i.e., $I \models f$. In the case of formula $E[fUg]$, X_i contains those states from which a state fulfilling g can be reached in less than or equal to i transitions. As a consequence, the shortest path which shows that $I \models E[fUg]$ certainly starts with a state in the first nonempty intersection of the initial states I with an X_i.

Formally, C constructs a shortest path for the μ-expression $\mu X.p(X)$ with $p(X) = g \vee (f \wedge \langle\rangle X)$ with starting state $s \in X_i \setminus X_{i-1}$.

$$C(\mu X.p(X), s, X_i) = \begin{cases} s.C(\mu X.p(X), s', X_{i-1}) & i > 0 \\ s & i = 0 \end{cases}$$

where $s' \in sR \cap X_{i-1}$ and $s \in X_i \setminus X_{i-1}$

Let I be the set of inital states for which the formula $\mu X.p(X)$ is true. f calculates the first X_i which has a nonempty intersection with I, if started with $i = 0$.

$$f(\mu X.p(X), i, I) = \begin{cases} X_i & \text{if } X_i \cap I \neq \emptyset \\ f(\mu X.p(X), i+1, I) & \text{otherwise} \end{cases}$$

V computes the shortest path from an initial state $s \in f(\mu X.p(X), 0, I) \cap I$ which is a witness for $I \models \mu X.p(X)$.

$$V(\mu X.p(X), s) = C(\mu X.p(X), s, f(\mu X.p(X), 0, I))$$

3.2 Witness construction for EGf

Let X_n denote the last approximation of the fixpoint iterations for $EGf = \nu X.f \wedge \langle\rangle X$. Exactly those states s fulfill this formula where there is an infinite path starting at s along which f is always fulfilled. Therefore, $C(\nu X.(f \wedge \langle\rangle X), s)$ calculates the witness which consists of a path which leads into a loop.

$$C(\nu X.(f \wedge \langle\rangle X), s) = \begin{cases} V(\mu Y.s \vee X_n \wedge \langle\rangle Y), s) & s \models \mu Y.s \vee X_n \wedge \langle\rangle Y \\ s.C(\nu X.(f \wedge \langle\rangle X), s') & \text{otherwise} \end{cases}$$

where $s' \in sR \cap X_n$.

V computes a witness for $I \models EGf$ from an initial state $s \in I$. (Note that $I \subseteq X_n$.)

$$V(\nu X.(f \wedge \langle\rangle X), s) = C(\nu X.(f \wedge \langle\rangle X), s)$$

3.3 Witness construction for EGf with fairness constraints

The following formula calculates EGf under fairness constraints h_k.

$$\nu Z.[f \wedge \bigwedge_{k=1}^{n} \langle\rangle[\mu X.Z \wedge h_k \vee (f \wedge \langle\rangle X)]] \tag{1}$$

This formula is fulfilled by those states where there is an infinite path where all states on the path fulfill f and the fairness constraints h_k are fulfilled infinitely often.

How to construct witnesses for FCTL formulae has been shown in [CGMZ94] and [CGL93]. They use the special meaning of FCTL formulae when constructing the witness.

$C(s)$ calculates the witness which consists of a path which leads into a loop where each fairness constraint is fulfilled by at least one state on the loop.

$$C(s) = F(s, s, n)$$

$$Wh_k(s) = V(\mu X.Z \wedge h_{n-k+1} \vee f \wedge \langle\rangle X, s)$$

$$F(t, s, k) = \begin{cases} Wh_k(s').F(t, last(Wh_k(s')), k \Leftrightarrow 1) & k > 0 \\ B(t, s) & k = 0 \end{cases}$$

where $s' \in sR \cap X_n^k$, and X_n^k is the last approximation for X^k of the formula $\mu X^k.Z \wedge h_{n-k+1} \vee (f \wedge \langle\rangle X^k)$. The function *last* returns the last state of a finite path.

$$B(t, s) = \begin{cases} V(\mu Y.t \vee f \wedge \langle\rangle Y, s') & \exists s' \in sR \cap Y_n^t \colon s' \models \mu Y.t \vee f \wedge \langle\rangle Y \\ C(s) & \text{otherwise} \end{cases}$$

where Y_n^t is the last approximation of the formula $\mu Y.t \vee f \wedge \langle\rangle Y$.

In contrast to the witness construction for $E[fUg]$, finding the shortest witness for formula $EG\ true$ under fairness constraints is NP-complete [CGMZ94].

4 Witnesses for the μ-calculus

In this section we develop a more general approach for the construction of witnesses for arbitrary μ-calculus formulae.

4.1 Example

We use an example to give an intuition for the construction of a witness for a formula in the μ-calculus.

What is the witness for

$$\mu X.\nu Y.(P \vee ((\mu Z.(X \vee \langle A\rangle Z)) \wedge \langle B\rangle Y)) \tag{2}$$

What is value returned for this formula by the model checking algorithm, i.e. what is the set of states fulfilling this formula? Let X_n be the value of variable X in the last fixpoint iteration. The value returned by the model checking algorithm (Algorithm 2.1) is then:

$$\nu Y.(P \vee (K \wedge \langle B\rangle Y)) \tag{3}$$

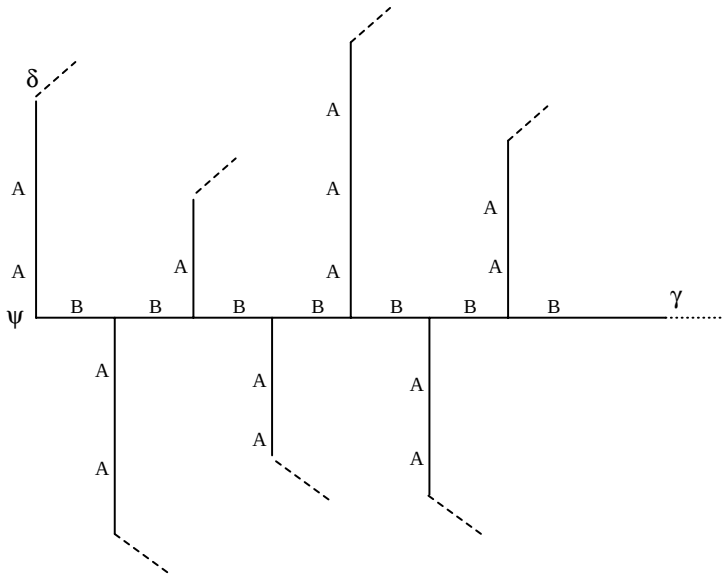

Figure 1: Witnesses: main path and side paths

where $K = \mu Z.(X_n \vee \langle A \rangle Z)$.

Figure 1 shows what kind of states ψ fulfill this formula. A state ψ fulfills this formula if there is a main path with only B-transitions where at each state there is a side path according K. I.e., a sidepath with only A-transitions which ends in a state, e.g., δ where again a structure similar to the one starting at ψ begins. The main path can either end at a state γ where P holds or end up in a loop.

4.2 Witness subclass of the μ-calculus?

In [CGMZ94] witnesses are only constructed for a subclass of FCTL: for formulae of type EGf and $E[fUg]$ with or without fairness constraints. I.e., they would not allow constructing a witness for formulae of type AGf and $A[fUg]$.

For which subclass of the μ-calculus do we want to allow witnesses to be constructed? It does not make sense to compute witnesses which consist of a tree of hundreds of paths. In FCTL the distinction is according to whether [ ] appears in the formula. We have to decide how we shall treat [ ] in the μ-calculus.

We therefore give some definitions of sublanguages of the μ-calculus which restrict the μ-calculus with respect to [ ] in different ways and discuss their appropriateness.

Definition 4.1 *$L_{P\mu}$ is defined as the sublanguage of L_μ where for all X in $\mu X.p(X)$ or $\nu X.p(X)$ it is the case that* within $p(X)$ *X is not in the scope of* [ ].

Example 4.1 $\mu X.(P \vee (\nu Y.X \wedge \langle\rangle Y)) \in L_{P\mu}$
$\mu X.(P \vee (\nu Y.X \wedge [\,]Y)) \notin L_{P\mu}$

Definition 4.2 *Let $L_{W\mu}$ be the sublanguage of L_μ where all variables are not in the scope of $[\,]$.*

Example 4.2 $\mu X.(P \vee [\,](\nu Y.X \wedge \langle\rangle Y)) \notin L_{W\mu}$ *since Y is in the scope of* $[\,]$
However, $\mu X.(P \vee [\,](\nu Y.X \wedge \langle\rangle Y)) \in L_{P\mu}$

An even stronger restriction is

Definition 4.3 *$L_{S\mu}$ is the sublanguage of L_μ where $[\,]$ is not allowed.*

Example 4.3 $\mu X.([\,]P \vee (\nu Y.X \wedge \langle\rangle Y)) \notin L_{S\mu}$
However, this formula belongs to both $L_{P\mu}$ and $L_{W\mu}$.

In the last definition, we can be sure to construct a witness the branching of which depends only on the type and number of connectives in the formula. In the second definition, we would also allow some short paths the lengths of which dependent on the number of nested $[\,]$ and $\langle\rangle$, but the branching dependent on the model. The first definition would allow arbitrary branching where the length of the branches depend on the model. If we ever allowed $[\,]$ arbitrarily within μ-,ν-subformulae we would get really large bushy trees.

These definitions showed the various effects of the position of $[\,]$ in the formula. We could restrict the μ-calculus formulae to one of the above definitions, e.g., to L_{S_μ} and give a witness construction algorithm for this subclass.

In this way, however, we have eliminated just one factor for getting large witnesses. Another factor is the recursive structure of formulae such as in Formula 2. It therefore makes sense to start with a general witness construction algorithm for the whole μ-calculus and then subsequently modify it to construct reduced witnesses. These reduced witnesses will be easier to understand than the original ones and thus enhance the understanding of errors.

4.3 Witness construction

Let $b(X) = \sigma X.p(X)$ if the latter formula appears as a subformula of the original formula f.

We remind the reader that formula p^r shall denote the value of subformula p in the last fixpoint iteration (cf. Algorithm 2.1).

Fact 1 $\sigma X.p(X) = X_n = p(X_n)$
The value for the subformula $\sigma X.p(X)$ is the value returned for it by the model checking algorithm in the last fixpoint iteration, i.e., X_n. The second equality follows from the fact that X_n is a fixpoint of $p(X)$.

Definition 4.4 *For a given model (S, R, L) a witness is a tree-like structure in the set of $\mathcal{T}^\infty$ or $\mathcal{T}$, formally defined as*

$$S^\infty = S^* \cup S^\omega$$

$$\mathcal{T} = S^* \cup \{(S^*,)^* S^*\}$$

$$\mathcal{T}^\infty = S^\infty \cup \{(S^\infty,)^* S^\infty\}$$

Here, S, '{', '}' and ',' belong to the alphabet of the regular expression, whereas the parentheses '(' and ')' denote the scope of the Kleene star.

Algorithm 4.1 *$C(f, s)$ constructs a witness for formula f from a starting state $s \models f$. C is defined by structural induction:*

$$C\colon L_\mu \times S \to \mathcal{T}^\infty$$

1. $C(P, s) = s$
2. $C(\neg P, s) = s$
3. $C(p \wedge q, s) = \{C(p, s), C(q, s)\}$
4. $C(p \vee q, s) = \begin{cases} C(p, s) & p^r \cap \{s\} \neq \emptyset \\ C(q, s) & otherwise \end{cases}$
5. $C(\langle\rangle p, s) = s.C(p, s'),$ *where* $s' \in sR \cap p^r$
6. $C([\,]p, s) = \{s.C(p, s') | s' \in sR\}$
7. $C(\sigma X.p(X), s) = C(p(X), s)$
8. $C(X, s) = C(b(X), s)$

The motivation for the definition of C should be clear. E.g., in order to construct a witness for $p \wedge q$ we construct a witness for p and a witness for q. In order to construct a witness for $p \vee q$ we construct either a witness for p or a witness for q.

Lemma 4.1 *If starting the algorithm $C(f, s)$ with $s \models f$ then the rewrite rules maintain the property $s \models g$ for derived $C(g, s)$.*

Proof:

- (1),(2): For the cases $C(P, s)$ and $C(\neg P, s)$ this is trivially fulfilled.
- (3): The case $C(p \wedge q, s)$ follows from $s \models p \wedge q \Leftrightarrow s \models p$ and $s \models q$.

- (4): Since $s \models p \Leftrightarrow s \models p^r$ we can check easily whether $s \models p$. If $s \models p$, the lemma is trivially fulfilled. Otherwise, $s \models q$ since $s \models p \vee q \Leftrightarrow s \models p$ or $s \models q$.
- (5): Follows directly from $s \models \langle\rangle p \Leftrightarrow \exists s' \colon sRs' \wedge s' \in p$ and $p = p^r$.
- (6): from the semantics of [ ]
- (7): from Fact 1 and the fact that we treat X as $X^r = X_n$ in Equation 8
- (8): $X = X^r = X_n = b(X)$, using Fact 1

■

It should be clear that this algorithm for witness construction does not necessarily terminate.

We can make it terminating by saving for each variable X the states fulfilling it already reached.

Algorithm 4.2

$$C \colon L_\mu \times S \to \mathcal{T}$$

C works in the same way as in Algorithm 4.1 except that

- *for each variable X the states already reached fulfilling it are saved*

  $$h \colon PropVar \to S$$

  where at the beginning of the algorithm

  $$\forall X \in PropVar \colon h(X) = \emptyset$$

- *Equation 7 is replaced by*

  $$C(\sigma X.p(X), s) = \begin{cases} C(p(X), s) & s \notin h(X) \\ s & otherwise \end{cases}$$

  where in the first case h is updated before C is applied to the arguments $p(X)$ and s by

  $$h := h[X/h(X) \cup \{s\}]$$

- *Only one C can operate at a time, so that h is updated correctly.*

4.4 A worked example

Let us use again the formula

$$\mu X.\nu Y.(P \vee ((\mu Z.(X \vee \langle A\rangle Z)) \wedge \langle B\rangle Y))$$

as an example for how Algorithm 4.1 constructs a witness for it from a state s fulfilling this formula.

$$C(\mu X.\nu Y.(P \vee ((\mu Z.(X \vee \langle A\rangle Z)) \wedge \langle B\rangle Y)), s) = \tag{4}$$

$$C(\nu Y.(P \vee ((\mu Z.(X \vee \langle A\rangle Z)) \wedge \langle B\rangle Y)), s) =$$

$$C(P \vee ((\mu Z.(X \vee \langle A\rangle Z) \wedge \langle B\rangle Y)), s) =$$

Let us suppose that s does not fulfill P. Then we have

$$C(\mu Z.(X \vee \langle A\rangle Z) \wedge \langle B\rangle Y, s) =$$

$$\{C(\mu Z.X \vee \langle A\rangle Z, s), C(\langle B\rangle Y, s)\} \tag{5}$$

Let us first continue with the first C-term:

$$C(\mu Z.(X \vee \langle A\rangle Z, s) =$$

$$C(X \vee \langle A\rangle Z, s) =$$

$$C(\langle A\rangle Z, s) =$$

$$s.C(Z, s_1) =$$

$$s.C(\mu Z.(X \vee \langle A\rangle Z), s_1) =$$

$$s.C(X \vee \langle A\rangle Z, s_1) =$$

$$s.C(\langle A\rangle Z, s_1) =$$

$$s.s1.C(Z, s_2) =$$

$$\ldots = s.s_1.\ldots.s_{l-1}.C(X \vee \langle A\rangle Z, s_l) =$$

$$s.s_1.\ldots.s_{l-1}.C(X, s_l) =$$

$$s.s_1.\ldots.s_{l-1}.C(\mu X.\nu Y.(P \vee ((\mu Z.(X \vee \langle A\rangle Z)) \wedge \langle B\rangle Y)), s_l)$$

This expression is in fact similar to 4. We could apply C in a similar fashion. Figure 2 shows which path we have developed so far.
Let us now continue with the second C-term in 5.

$$C(\langle B\rangle Y, s) =$$

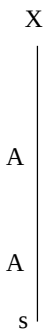

Figure 2: Path developed so far

$$s.C(Y, s') =$$
$$s.C(\nu Y.(P \vee ((\mu Z.(X \vee \langle A \rangle Z)) \wedge \langle B \rangle Y)), s') =$$
$$s.C(P \vee ((\mu Z.(X \vee \langle A \rangle Z)) \wedge \langle B \rangle Y), s') =$$

Let us suppose that s' does not fulfill P. We then have:

$$s.C((\mu Z.(X \vee \langle A \rangle Z)) \wedge \langle B \rangle Y, s') =$$

$$s.\{C(\mu Z.(X \vee \langle A \rangle Z), s'), C(\langle B \rangle Y, s')\} =$$

We are in a similar situation as in 5. We have to develop a path from s' to a state fulfilling X, etc., and also continue along $\langle B \rangle$ to Y.

$$C(\langle B \rangle Y, s') =$$

$$s'.C(Y, s'') =$$
$$s'.C(\nu Y.(P \vee ((\mu Z.(X \vee \langle A \rangle Z)) \wedge \langle B \rangle Y)), s'') =$$
$$s'.C(P \vee ((\mu Z.(X \vee \langle A \rangle Z)) \wedge \langle B \rangle Y), s'') = \ldots$$

Figure 3 shows the paths we have developed so far.

We expand Y as long as we do not reach a state fulfilling P or until we have found a loop back to states already reached.

In this way we subsequently develop a structure as in Figure 1.

4.5 Reduced witnesses

Huge witnesses, as the one constructed in the above example, are difficult to understand. How can we modify function C to construct reduced witnesses. There are two equations in Algorithm 4.1 which cause witnesses to get very bushy: Equation 6 and 3. We suggest two possible modifications to Algorithm 4.1.

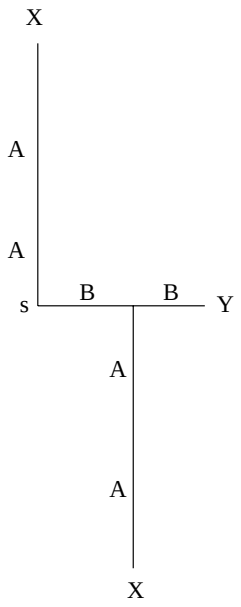

Figure 3: Paths developed so far

Just one successor If we replace equation 6 by

6'. $C([\,]p, s) = s.C(p, s')$, where $s' \in sR$

the witness for just one successor is constructed. Until the end of the paper we use this equation instead of equation 6.

Just one conjunct If we replace equation 3 by

3'. $C(p \wedge q, s) = \mathit{choose}(p, q)$, where $\mathit{choose}(p, q)$ returns either $C(p, s)$ or $C(q, s)$

the witness for just one conjunct is constructed.

In this way we could, e.g., by continuously choosing the second conjunct, construct only the path along the B-arcs in Figure 1.

We also could define *choose* at the beginning of the witness construction, in this way making coherent choices.

4.6 Side paths and main path

When choosing between the different conjuncts (Equation 3) we could always go along the side paths. This does, however, not reflect well the meaning of the formula. We remedy this problem in this subsection.

It does not make sense to return the whole bushy tree in Figure 1 as a witness for Formula 2. Since there are a vast amount of side paths (along $\langle A \rangle$-arcs), we should refrain from constructing these. Indeed, the user of a model checker only wants to see the main path - the path along the B-arcs in Figure 1 - and not all side conditions the states on the path have to fulfill. We therefore suggest returning just the main path as a witness, e.g., the witness for the much simpler Formula 3, treating K as a propositional constant, as a witness for Formula 2.

The main paths of a formula f ($M(f)$) are defined as

$$M\colon L_\mu \to \mathcal{P}(L_\mu)$$

$$M(\mu X.p(X)) = \begin{cases} \{\mu X.p(X)\} & \mathcal{I}(\mu X.p(X)) = 1 \\ M(p(X)) & \text{otherwise} \end{cases}$$

$$M(\nu X.p(X)) = \begin{cases} \{\nu X.p(X)\} & \mathcal{I}(\nu X.p(X)) = 1 \\ M(p(X)) & \text{otherwise} \end{cases}$$

$$M(p \wedge q) = M(p) \cup M(q)$$
$$M(p \vee q) = M(p) \cup M(q)$$
$$M(\langle\rangle p) = M(p)$$
$$M([\,]p) = M(p)$$
$$M(P) = \emptyset$$
$$M(Y) = \emptyset$$
$$M(\neg P) = \emptyset$$

$$VM(f) = \{X | \sigma X.p(X) \in M(f)\}$$

The motivation behind this definition is that the fixpoint iterations of μ- and ν-expressions with iteration depth 1 are the last when model checking a formula f.

Example 4.4

$$M(\mu X.\nu Y.(P \vee ((\mu Z.(X \vee \langle A\rangle Z)) \wedge \langle B\rangle Y))) =$$
$$\{\nu Y.(P \vee ((\mu Z.(X \vee \langle A\rangle Z)) \wedge \langle B\rangle Y))\}$$

Example 4.5

$$M(\mu X.(\langle A\rangle(\nu Y.Q \wedge \langle A\rangle Y \wedge X) \vee \langle A\rangle(\nu Z.R \wedge \langle A\rangle(\mu W.Z \wedge X \vee \langle A\rangle W)))) =$$
$$\{\nu Y.Q \wedge \langle A\rangle Y \wedge X, \mu W.Z \wedge X \vee \langle A\rangle W\}$$

Algorithm 4.3 *We can easily obtain an algorithm which constructs only the main paths of a formula f from Algorithm 4.2 by replacing Equation 7 by*

$$7.\ C(\sigma X.p(X), s) = \begin{cases} s & (\forall X \in VM(f) : X \not\preceq \sigma X.p(X)) \vee s \in h(X) \\ C(p(X), s) & otherwise \end{cases}$$

and Equation 8 by

$$8.\ C(X, s) = \begin{cases} C(b(X), s) & b(X) \in M(f) \\ s & otherwise \end{cases}$$

Example 4.6 *In the formula*

$$\mu X.((\nu Y.(X \wedge h \wedge \langle\rangle Y)) \wedge \langle\rangle(P \wedge X))$$

X does not only appear within a subformula with iteration depth 1. In this case, Algorithm 4.3 constructs the witness for $\nu Y.(X \wedge h \wedge \langle\rangle Y)$ plus another short path of length 1 to a state which fulfills both P and X as a witness for this formula. In other words, we refrain from constructing another witness for the last X. This makes sense, since otherwise we would find ourselves in a cycle where we construct a path for X again and again.

4.7 Abstract and concrete witnesses

As we have seen in the case of CTL, $\mu X.p(X)$ represents states from which there is a finite path to certain conditions. In the case of CTL, $\nu X.p(X)$ represents states from which there is an infinite path along states fulfilling certain conditions.

In the general case of the μ-calculus, what is the meaning of a formula $\mu X.p(X)$? The states fulfilling this formula are those from which states fulfilling $p(false)$ can be reached. Therefore, a natural witness for $s \models \mu X.p(X)$ would be a path from s to $p(false)$.

Let us consider Figure 4. The witness construction algorithms presented so far could construct a witness for this formula consisting of the path from s to s' and the path along the circle back to s'. I.e., an abstract witness would be constructed, in the sense that for each state on the witness for $\mu X.p(X)$ which is supposed to fulfill $\mu X.p(X)$, it is shown that it fulfills $p(X)$.

In the case of $\nu X.p(X)$ we do not have this problem since, of course, a looping path can be returned as a witness for $\nu X.p(X)$. This is a concrete witness.

We have to modify Algorithm 4.3 so that it is ensured that for both $\mu X.p(X)$ and $\nu X.p(X)$ the natural (concrete) witnesses are constructed, i.e, for $\mu X.p(X)$ a path to $p(false)$.

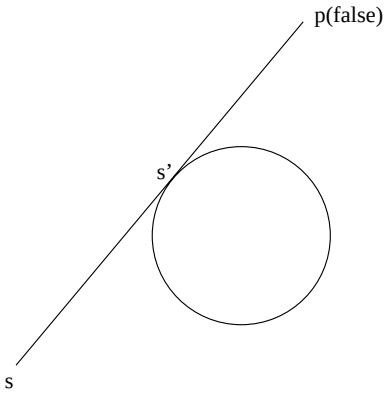

Figure 4: An abstract witness for $\mu X.p(X)$

Algorithm 4.4 *We can obtain an algorithm which constructs only concrete witnesses from Algorithm 4.3 by replacing Equation 7 by*

$$7''.\ C(\sigma X.p(X), s) = \begin{cases} V(\sigma X.p(X), s) & \sigma X.p(X) \in M(f) \\ C(p(X), s) & \textit{otherwise} \end{cases}$$

and Equation 8 by

$$8''.\ C(X, s) = s$$

where V constructs concrete witnesses for the main paths of f.

Termination of this algorithm follows from the termination of V and the fact that C is non-recursive (Equation 8").

In the following two sections we investigate how to construct concrete witnesses for the main paths.

5 Main paths of type $\mu X.p(X)$

5.1 Shortest paths for $\mu X.p(X)$

We did not care about constructing shortest witnesses in the previous section. We will talk about this now.

In the case of CTL, a shortest witness for $\mu X.p(X)$ can be constructed efficiently. In the following, we will see how shortest witnesses can be constructed for general formulae of type μ.

5.1.1 Occurrences of X independent

Definition 5.1 *The different occurrences of X in $p(X)$ in $\mu X.p(X)$ are called*

independent of each other if $p(X)$ *can be rewritten into* $\bigvee p_j(X)$ *where each* $p_j(X)$ *does not contain* X *or* $p_j(X)$ *is of the form* $q_{j_0} \wedge \Box(q_{j_1} \wedge \ldots \Box(q_{j_n} \wedge X)\ldots)$ *where each* q_{j_m} *does not contain* X*.*

For this class of $\mu X.p(X)$ we can give an algorithm which calculates the shortest witness.

When calculating the fixpoint approximations we save the values of the disjuncts $p_j(X)$ containing X in $p(X)$ separately. In this way, we achieve a kind of tree (Figure 5) when the approximations are computed. X_i^j denotes the value obtained in the ith fixpoint iteration by substituting $father(X_i^j)$, i.e., one of the splitted values in the previous fixpoint iteration, for X in the jth disjunct containing X. During the fixpoint iterations we also save for each X_i^j the total distance t in transitions from X_0: $X_i^{j,t}$.

Example 5.1 *The formula* $\mu X.(P \vee \langle\rangle X \vee \langle\rangle\langle\rangle\langle\rangle X)$ *has the approximations* $X_0, X_1, \ldots$ *to its least fixpoint. These approximations are split according to* $\langle\rangle X$ *and* $\langle\rangle\langle\rangle\langle\rangle X$*. Let* $p_1(X) = \langle\rangle X$ *and* $p_2(X) = \langle\rangle\langle\rangle\langle\rangle X$*.*

at the beginning:
$X_0 = P$
after the first fixpoint iteration:
$X_1^1 = \langle\rangle P$, $X_1^2 = \langle\rangle\langle\rangle\langle\rangle P$
after the second fixpoint iteration:
substituting X_1^1 *and* X_1^2 *in* $p_1(X)$ *we obtain:*
$X_2^1 = \langle\rangle\langle\rangle P$ *and* $X_2^2 = \langle\rangle\langle\rangle\langle\rangle\langle\rangle P$
substituting X_1^1 *and* X_1^2 *in* $p_2(X)$ *we obtain:*
$X_2^1 = \langle\rangle\langle\rangle\langle\rangle\langle\rangle P$ *and* $X_2^2 = \langle\rangle\langle\rangle\langle\rangle\langle\rangle\langle\rangle\langle\rangle P$

At this time we have constructed a tree as in Figure 5.
The process continues similarly.
The total distance for, e.g., X_2^2 *as a son of* X_1^2 *would be 6.*

We construct the shortest path in a similar fashion as in the case of CTL.
f returns the X_i^j with smallest distance from X_0.

$$f\colon L_\mu \times \mathcal{P}(S) \to PropVar$$

$f(\mu X.p(X), I) =$
loop
h = 0
if $\exists X_i^{j,h} : X_i^{j,h} \cap I \neq \emptyset$ return $X_i^{j,h}$
h = h + 1
pool

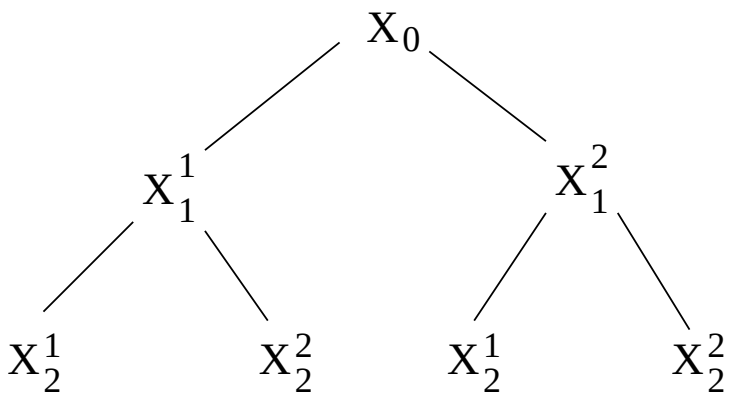

Figure 5: Approximations where there are different depths in a μ expression

$$C\colon L_\mu \times S \times PropVar \to \mathcal{T}$$

$$C(\mu X.p(X), s, X_i^j) = \begin{cases} D(\mu X.p(X), s, X_i^j, n) & i > 0 \\ s & i = 0 \end{cases}$$

where $p_j(X) = q_{j_0} \wedge \Box(q_{j_1} \wedge \ldots \Box(q_{j_n} \wedge X)\ldots)$

$$D\colon L_\mu \times S \times PropVar \times \mathbb{N} \to \mathcal{T}$$

$$D(\mu X.p(X), s, X_i^j, k) = \begin{cases} s.D(\mu X.p(X), s', X_i^j, k \Leftrightarrow 1) & k > 1 \\ s.C(\mu X.p(X), s', father(X_i^j)) & k = 1 \end{cases}$$

where $s' \in sR \cap q_{l_{n-k+1}} \cap R^{k-1} father(X_i^j)$ and $s \in X_i^j \setminus father(X_i^j)$.

$V(\mu X.p(X), s)$ constructs the shortest witness for formula $\mu X.p(X)$ for a starting state s where all transitions are shown.

$$V\colon L_\mu \times S \to \mathcal{T}$$

$$V(\mu X.p(X), s) = C(\mu X.p(X), s, f(\mu X.p(X), I))$$

where $s \in f(\mu X.p(X), I) \cap I$.

We can define a rougher witness construction as follows:

$$C(\mu X.p(X), s, X_i^j) = \begin{cases} s.C(\mu X.p(X), s', father(X_i^j)) & i > 0 \\ s & i = 0 \end{cases}$$

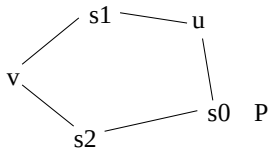

Figure 6: Interdependent X

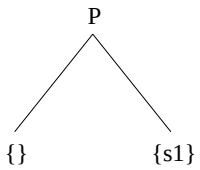

Figure 7: Wrong tree for interdependent X

where $s' \in sR^n \cap father(X_i^j)$ and $s \in X_i^j \setminus father(X_i^j)$.

In this way we achieve in a faster way the rough structure of the witness which might also be easier to understand.

The tree in Figure 5 can also degenerate to a linear list. This is trivially the case when there is just one X in $p(X)$ of $\mu X.p(X)$.

When all occurrences of X have the same depth, we can also make the tree degenerate to a linear list. This can be achieved by not differentiating between the different disjuncts at the time of the calculation of the approximations. The price we have to pay is that we always have to find the first j with $s \models p_j(X_{i-1})$ at the time of witness construction where s is the current state.

5.1.2 Interdependent occurrences of X

An example for interdependent X is the formula

$$\mu X.P \vee \langle\rangle X \wedge \langle\rangle\langle\rangle X \vee \langle\rangle\langle\rangle X$$

Figure 6 illustrates why the witness construction given in the previous subsubsection does not work. In this model, only $s0 \models P$ and only $s2 \in I$. The first fixpoint iterations for the above formula yields the tree in Figure 7. In the second fixpoint iteration we would like to replace the empty set of the left son by $\{s_2\}$. We see that with higher fixpoint iteration the shortest path to the goal state (P) does not necessarily increase, but can in fact decrease with each iteration. This is the case in this model where the shortest witness $s2 \rightarrow s0$ is added via a different path and where the paths to $s0$ from s_2 in previous fixpoint iterations were longer.

6 Main paths of type $\nu X.p(X)$

6.1 Passive generation of witnesses for $\nu X.p(X)$

Similarly to Algortihm 4.2 the states already reached during the witness construction are saved and the construction is stopped as soon as a state is reached again.

Algorithm 6.1 *$W_\nu : L_\mu \times PropVar \times S \to \mathcal{T}$ operates in the same way as C in Algortihm 4.2, except for the last two equations:*

1. $W_\nu(\sigma Y.p(Y), X, s) = \begin{cases} W_\nu(p(Y), X, s) & Y \equiv X \wedge s \notin h(X) \\ s & otherwise \end{cases}$

2. $W_\nu(Y, X, s) = \begin{cases} W_\nu(b(Y), X, s) & Y \equiv X \\ s & otherwise \end{cases}$

The algorithm starts with $W_\nu(\nu X.p(X), X, s)$ where $\nu X.p(X)$ is a main path. In this algorithm h only needs to save the states marked with X.

The above algorithm may still construct large branching trees as a witness. The reason for this are conjoined subformulae within $\sigma X.p(X)$ which contain X.

We can remedy this problem by replacing Equation 3 by

$$W(p \wedge q, X, s) = \begin{cases} W_\nu(p, X, s) & occ(X, p) \\ W_\nu(q, X, s) & \text{otherwise} \end{cases}$$

In this way, just one path is constructed.

The following definition of V combines this algorithm with Algorithm 4.4

$$V(\nu X.p(X), s) = W_\nu(\nu X.p(X), X, s)$$

6.2 Active generation of witnesses for $\nu X.p(X)$

In the case of $EGf = \nu X.f \wedge \langle\rangle X$ the witness construction procedure for CTL and FCTL searches for a loop in a more goal-directed way. We extend this approach to a bit broader class of formulae.

6.2.1 Fixed depth

We restrict the class of formulae to the type

$$\nu X.(p \wedge \langle\rangle^m X)$$

where $m \geq 1$, $p \neq true$ and p is a conjunction of disjuncts not containing X (L_2 are the subformulae treated as literals.).

Let X_n denote the value of the last fixpoint iteration of $\nu X.p \wedge \langle\rangle^m X$. Let $s \in X_n \cap I$. X_n contains all states where there is an infinite path taking m steps between such states, each of which fulfills p.

$$C(\nu X.(p \wedge \langle\rangle^m X), s) = \begin{cases} V(\mu Y.s \vee X_n \wedge \langle\rangle^m Y, s) & s \models \mu Y.s \vee X_n \wedge \langle\rangle^m Y \\ D(\nu X.(p \wedge \langle\rangle^m X), s, m) & \text{otherwise} \end{cases}$$

where

$$E\colon L_\mu \times S \times \mathbb{N} \to \mathcal{T}$$

$$E(\nu X.(p \wedge \langle\rangle^m X), s, d) = \begin{cases} s.E(\nu X.(p \wedge \langle\rangle^m X), s', d \Leftrightarrow 1) & d > 0 \\ C(\nu X.(p \wedge \langle\rangle^m X), s) & \text{otherwise} \end{cases}$$

where $s' \in sR \cap R^{d-1} X_n$.

$$V(\nu X.p(X), s) = C(\nu X.p(X), s)$$

where $s \in X_n \cap I$.

Obviously, the calculation of $C(\nu X.p(X), s)$ is expensive since for each newly reached state $s \in X_n$, $\mu Y.s \vee X_n \wedge \langle\rangle^m Y$ has to be calculated. In [CGMZ94] they do not have a similar problem since they do such a test only after having calculated long paths. We could do it similarly: one could come away with fewer of such fixpoint iterations by computing longer paths in X_n before such a test is performed.

6.2.2 Different depths and arbitrary occurrence of X

Actively looking for loops could also be extended to general $\nu X.p(X)$. It remains to be seen in practice what kind of method is in general computationally the less expensive.

6.3 Normalization of ν-expressions

For ν-expressions of arbitrary form it is not the case as in CTL that ν-expressions represent only infinite paths. Instead, they can represent both finite and infinite trees. However, we are able to extract the finite witnesses from the ν-expressions.

Lemma 6.1

$$\mu X.p(X) \subseteq \nu X.p(X)$$

Proof: By induction on the number of fixpoint iterations and the fact that $p(X)$ is monotone.

■

Corollary 6.1

$$\nu X.p(X) = \mu X.p(X) \vee \nu X.p(X)$$

If we impose certain restrictions on the ν-expressions we are even able to separate finite from infinite witnesses.

When treating L_2 as the set of literals we bring $p(X)$ in $\nu X.p(X)$ into disjunctive normal form. We then have:

Lemma 6.2

$$\nu X.(\bigvee_l p_l \vee \bigvee_m p_m(X))$$

where each p_l and p_m is constructed by the operators $\langle\rangle$ and $\wedge$ and the literals in L_2 (p_m does and p_l does not contain X) and $\forall q \in L_2 \setminus \{X\} : (q \preceq \nu X.(\bigvee_l p_l \vee \bigvee_m p_m(X))) \rightarrow \neg occ(X, q)$.

$$\nu X.(\bigvee_l p_l \vee \bigvee_m p_m(X)) =$$
$$\mu X.(\bigvee_l p_l \vee \bigvee_m p_m(X)) \vee \nu X.(\bigvee_m p_m(X))$$

Proof: We show that the nth fixpoint iteration on the left-hand side is equivalent to the union of the two nth fixpoint iterations on the right-hand side. I.e., $l^n = r_1^n \vee r_2^n$ where

$$l = \nu X.(\bigvee_l p_l \vee \bigvee_m p_m(X))$$

$$r_1 = \mu X.(\bigvee_l p_l \vee \bigvee_m p_m(X))$$

and

$$r_2 = \nu X.(\bigvee_m p_m(X))$$

The rest follows from the fact that all three fixpoints exist.

Induction basis:
at iteration -1: $true = true$
at iteration 0: $\bigvee_l p_l \vee \bigvee_m p_m(true)) = \bigvee_l p_l \vee \bigvee_m p_m(false)) \vee \bigvee_m p_m(true))$
Induction step:
$l^{n+1} = \bigvee p_l \vee \bigvee p_m(\bigvee p_n \vee \bigvee p_o(l^{n-1})) =$

$\bigvee p_l \vee \bigvee p_m(\bigvee p_n \vee \bigvee p_o(r_1^{n-1} \vee r_2^{n-1})) =$
induction hypothesis
$\bigvee p_l \vee \bigvee p_m(\bigvee p_n) \vee \bigvee\bigvee p_m(p_o(r_1^{n-1})) \vee \bigvee\bigvee p_m(p_o(r_2^{n-1}))$
using the fact that both operators $\wedge$ and $\langle\rangle$ (and these are the only operators in p_m, p_o (besides the literals in L_2)) distribute over $\vee$

$r_1^{n+1} = \bigvee p_l \vee \bigvee p_m(r_1^n) =$
$\bigvee p_l \vee \bigvee p_m(\bigvee p_n \vee \bigvee p_o(r_1^n)) =$
$\bigvee p_l \vee \bigvee p_m(\bigvee p_n) \vee \bigvee\bigvee p_m(p_o(r_1^n))$

$r_2^{n+1} = \bigvee p_m(p_o(r_2^{n-1}))$

It follows directly that $l^{n+1} = r_1^{n+1} \vee r_2^{n+1}$.

■

This lemmas allow us to simplify the witness construction for $\nu X.p(X)$.

Let $\nu X.p(X) = \mu Y.q(Y) \vee \nu Z.r(Z)$ where the right hand-side is obtained by application of the previous lemma or previous corollary. $\mu Y.q(Y)$ then contains all finite witnesses contained in $\nu X.p(X)$. Let $\mu Y.q(Y) = false$ if such a transformation is not possible. Let

$$norm_1(\nu X.p(X)) = \mu Y.q(Y)$$

$$norm_2(\nu X.p(X)) = \nu Z.r(Z)$$

The following definition of V incorporates the normalization:

$$V(\nu X.p(X), s) = \begin{cases} V(norm_1(\nu X.p(X)), s) & s \models norm_1(\nu X.p(X)) \\ V(norm_2(\nu X.p(X)), s) & \text{otherwise} \end{cases}$$

I.e., in order to calculate a witness for $\nu X.p(X)$ we try first to find a finite witness for the μ-expression obtained by normalization. If there is no such finite witness, we calculate the ν-expressions and construct the infinite witness from it. In this way, we could get away without the expensive witness construction for $\nu X.p(X)$.

7 Comparing μ-calculus to FCTL witness generation

7.1 Witnesses for μ-expressions of type FCTL

FCTL is a subclass of L_{μ_2}. The counterexamples for FCTL in [CGMZ94] are for a special type of L_{μ_2} formulae. In the construction of the witnesses the special

meaning of the FCTL formula is exploited. Therefore, their counterexample construction does not extend to the whole μ-calculus .

Our algorithm above would construct a path for each separate $\wedge$ in the formula

$$\nu Z.[f \wedge \bigwedge_k \langle\rangle[\mu X.Z \wedge h_k \vee (f \wedge \langle\rangle X)]] \tag{6}$$

The algorithm in [CGMZ94], however, would construct a single path with a cycle in which all fairness constraints h_k are contained.

If we wanted a similar witness we would have to modify Algorithm 4.4 so that the first subformula with iteration depth 2 of type

$$\nu Z.[L \wedge \bigwedge_k \langle\rangle[\mu X.Z \wedge h_k \vee (P \wedge \langle\rangle^m X)]]$$

is dealt with as in [CGMZ94]. Note that L and P can refer to higher level variables and that we can have arbitrary nesting everywhere else in the formula which contains this subformula.

7.2 Interactive generation of witnesses

The witness constructed by Algorithm 4.4 could be expanded interactively by the user.

For example, Algorithm 4.4 would construct a path for each $\wedge$ in Formula 1. Each such path ends with a state (e.g. s) fulfilling $h_k \wedge Z$. The user could demand to extend such a path by interactively typing $V(\mu X.Z \wedge h_2 \vee (f \wedge \langle\rangle X), s)$, which would construct a path to fairness constraint h_2. In this way, the user could himself make construct the counterexample as it would be constructed by the algorithm in [CGMZ94].

We could also modify Algorithm 4.4 in such a way that only the path for just one conjunct is constructed. The path could then be extended by, e.g., the witness for a different conjunct in the case of Formula 1.

Interactive generation of witnesses allows different paths to be pursued. This allows a much more flexible witness generation.

8 Conclusions and future work

We have presented an algorithm for witness generation for formulae of the whole μ-calculus. Calculating reduced witnesses enhances the understanding of witnesses. It turned out that constructing short witnesses is not computationally expensive for many μ-calculus formulae.

The μ-calculus allows greater flexibility than formulae in CTL. The interactive generation of counterexamples/witnesses can also increase the user's flexibility.

In future work, the suggested algorithms have to be implemented into our μ-calculus model checker [Bie95a]. Testing and analysis will show which of the algorithms are most appropriate for what kind of model checking problems/formulae. Especially the classes of L_μ formulae for which the different main path witness construction algorithms are best have to be determined in practice.

References

[Bie95a] A. Biere. μcke – an evaluator of μ-calculus formulae. Technical report, Fakultät für Informatik,University of Karlsruhe,Germany, 1995.

[Bie95b] A. Biere. Semantik eines μ-Kalkül-Auswerters. Technical report, Fakultät für Informatik,University of Karlsruhe,Germany, 1995.

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

[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.

[Cle93] Rance Cleaveland. The concurrency workbench: A semantics-based verification tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[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.

[Rau95] Antoine Rauzy. Toupie: A constraint language for model checking. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910. Springer-Verlag, 1995. (Châtillon-sur-Seine Spring School, France, May 1994).

[SE89] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, June 1989.

[Zuc93a] J. Zucker. The propositional mu-calculus and its use in model checking. In P. E. Lauer, editor, *Functional Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *LNCS*, pages 117–128. Springer-Verlag, Berlin, DE, 1993.

[Zuc93b] J. Zucker. Propositional temporal logics and their use in model checking. In P. E. Lauer, editor, *Functional Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *LNCS*, pages 108–116. Springer-Verlag, Berlin, DE, 1993.