Proving Properties of Infinite Behaviours by Transformation of ω–programs

Alberto Pettorossi (Univ. Tor Vergata, Rome, Italy)
Maurizio Proietti (IASI-CNR, Rome, Italy)
Valerio Senni (Univ. Tor Vergata, Rome, Italy)

IFIP WG 2.1, Atlantic City, NJ, USA *September 19-24, 2010*

Motivations

- Transformations are useful for theorem proving and program verification: from FOL to clause form, from Temporal Logics (or monadic second order logic) to Büchi automata, quantifier elimination, etc.

- Unfold/fold transformations have been used for proving program properties [Kott 1982, P.P. 1993, Roychoudhury et al. 1999, Seki 2009].

- Goal of this work: a general methodology for proving properties of reactive systems via unfold/fold transformations.

Reactive Systems

- Reactive systems: communication protocols, security protocols, hardware controllers, etc.

- Various models of reactive systems with infinite behaviour:

  ω-languages,
  Büchi automata,
  temporal and modal logics,
  ω-programs.

Properties as Language Inclusions

A: **while** true **do** $think_A$; $wait_A$; use_A **od**

B: **while** true **do** $think_B$; $wait_B$; use_B **od**

---

no starvation (liveness) for A :

$$System_{AB} \subseteq \left(\overline{wait_A}^{\,*} \,;\; wait_A \,;\; \overline{use_A}^{\,*} \,;\; use_A\right)^{\omega}$$

mutual exclusion (safety) :

$$\left|\begin{matrix} any_A \\ any_B \end{matrix}\right|^{*} \left|\begin{matrix} use_A \\ use_B \end{matrix}\right| \left|\begin{matrix} any_A \\ any_B \end{matrix}\right|^{\omega} \cap \left|\begin{matrix} System_A \\ System_B \end{matrix}\right|^{\omega} = \varnothing$$

Bakery Protocol for Processes A and B

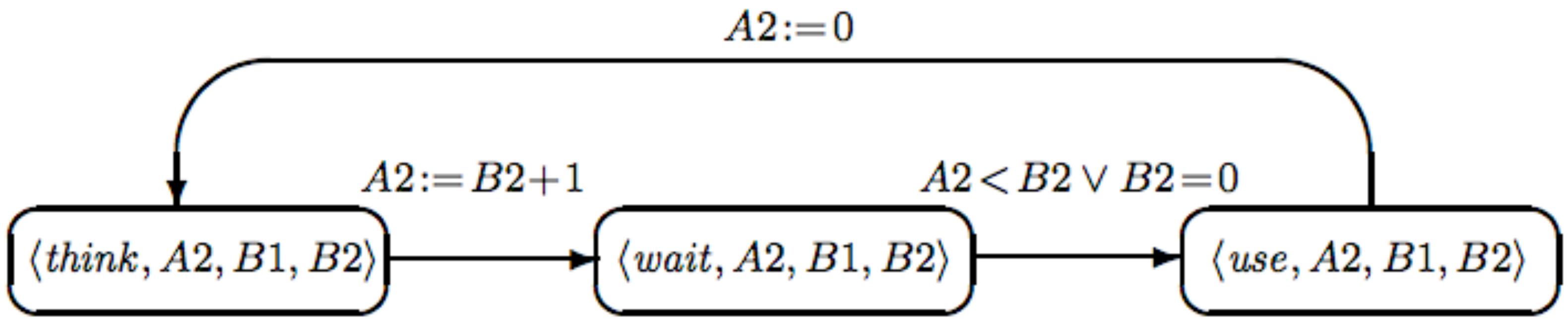


Fig. 2. The Bakery protocol: a graphical representation of the transition relation t_A for the agent A. The assignment $X := e$ on the arc from a state s_1 to a state s_2 tells us that the value of the variable X in s_2 is the value of the expression e in s_1. The boolean expression b on the arc from a state s_1 to a state s_2 tells us that the transition from s_1 to s_2 takes place iff b holds.

Roland's ω-Equivalence

$$a\,(ba)^{*} = (ab)^{*}\,a$$

$$a\,(ba)^{\omega} = (ab)^{\omega}$$

Plan of the Talk

- Specifying reactive systems by logic programs on infinite lists: ω-programs.

- Transformation rules for ω-programs.

- A transformation-based method for proving properties of ω-programs.

Proofs via Transformations

- Given any ω-program P and unary predicate p, in order to prove

  $M(P) \models \exists X\, p(X)$,

1. try to transform P into a monadic ω-program T, such that

   $M(P) \models \exists X\, p(X) \quad \text{iff} \quad M(T) \models \exists X\, p(X)$

2. apply the decision algorithm MDec to check whether or not

   $M(T) \models \exists X\, p(X)$

ω-programs

- ω-programs are typed, logic programs with three types:

  fterm (finite term), elem (element of an infinite list), ilist (infinite list).

- [_ | _]: elem $\times$ ilist $\rightarrow$ ilist is interpreted as the constructor of infinite lists.

- Each predicate has at most one argument of type ilist (to avoid unification between infinite lists).

Semantics of ω-programs

For a locally stratified ω-program P, the perfect model M(P) is constructed over the Herbrand universe extended with infinite lists.

Examples

Let {a,b} be the set of constants of type elem.

Let L be a variable of type ilist, i.e., L ranges over $(a+b)^\omega$.

P: p([a|L]) ← q(L)
   q(L) ←

$p(L) \in M(P)$ iff $L \in a(a+b)^\omega$

P: p(L) ← ¬q(L)
   q([a|L]) ← q(L)
   q([b|L]) ←

$p(L) \in M(P)$ iff $L \in a^\omega$

$q(L) \in M(P)$ iff $L \in a^*b(a+b)^\omega$

P: p([a|L]) ← p(L)

$M(P) = \emptyset$.

Monadic ω-programs

- A monadic ω-program is a set of clauses of the form:

$$p_0([s|X_0]) \leftarrow \boxed{p_1(X_1) \wedge \ldots \wedge p_k(X_k)} \wedge \boxed{\neg\, p_{k+1}(X_{k+1}) \wedge \ldots \wedge \neg\, p_m(X_m)}$$

where:

- s is a constant of type elem,
- $X_0, X_1, \ldots, X_k, X_{k+1}, \ldots, X_m$ are variables of type ilist, and
- there exists a level mapping h: Pred $\rightarrow$ N such that:

for $i=1,\ldots,k$, if $X_i=X_0$ then $h(p_i) \leq h(p_0)$ else $h(p_i) < h(p_0)$

for $i=k+1,\ldots,m$, $h(p_i) < h(p_0)$

- Some of the predicates p_i's may be nullary.

- A monadic ω-program is stratified (hence locally stratified).

Plan of the Talk

- Specifying reactive systems by logic programs on infinite lists: ω-programs.

- Transformation rules for ω-programs.

- A transformation-based method for proving properties of ω-programs.

Transformation Sequences

- A transformation sequence is a sequence of locally stratified ω-programs

$$P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_n$$

where $P_i \rightarrow P_{i+1}$ is obtained by applying one of the following rules: Definition Introduction, Instantiation, Positive Unfolding, Negative Unfolding, Positive Folding, Negative Folding, Subsumption.

- The rules are similar to [Seki 91, Maher 93, Roychoudhury et al. 02, FPP 04, Seki 09], but with different applicability conditions, needed for the correctness of the proof technique.

A Transformation Example

Property: There exists an infinite list $L=[s_0, s_1, s_2, \ldots]$ in $\{a,b\}^{\omega}$ whose elements at even positions are all a's (that is, $s_0=s_2= \ldots = a$).

$$\exists L\ \boxed{\forall X\ (\mathit{position}(X) \wedge \mathit{even}(X) \rightarrow \mathit{member}(X,L,a))}\ )$$

prop(L)

prop(L) ← ¬ negprop(L)
negprop(L) ← position(X) ∧ even(X) ∧ ¬member(X,L,a)
position(0) ←
position(s(X)) ← position(X)
even(0) ←
even(s(X)) ← ¬ even(X)
member(0,[H|T],H) ←
member(s(X),[H|T],S) ← member(X,T,S)

Locally stratified w.r.t. a stratification function σ

This ω-program is not monadic.

A Transformation Example

Property: There exists an infinite list $L=[s_0, s_1, s_2, \ldots]$ in $\{a,b\}^{\omega}$ whose elements at even positions are all a's (that is, $s_0=s_2= \ldots = a$).

$$\exists L\ \forall X\ (position(X) \wedge even(X) \rightarrow member(X,L,a)\ )$$

prop(L)

prop(L) ← ¬ negprop(L)
negprop(L) ← position(X) ∧ even(X) ∧ ¬member(X,L,a)
position(0) ←
position(s(X)) ← position(X)
even(0) ←
even(s(X)) ← ¬ even(X)
member(0,[H|T],H) ←
member(s(X),[H|T],S) ← member(X,T,S)

Locally stratified w.r.t. a stratification function σ

This ω-program is not monadic.

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)

non monadic

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

Instantiation   L / [a|T] ; L / [b|T]

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a) non monadic

Instantiation

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

position(0) ←
position(s(X)) ← position(X)

Positive Unfolding

negprop([a|T]) ← even(0) ∧ ¬ member(0,[a|T],a)
negprop([a|T]) ← position(X) ∧ even(s(X)) ∧ ¬ member(s(X),[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding***$^{+}$

negprop([a|T]) ← ¬ member(0,[a|T],a)
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(s(X),[a|T],a)
negprop([b|T]) ← ¬ member(0,[b|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding^{+}; Negative Unfolding***

negprop([a|T]) ← ¬ member(0,[a|T],a)
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(s(X),[a|T],a)
negprop([b|T]) ← ¬ member(0,[b|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

Instantiation

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

Positive Unfolding⁺; Negative Unfolding

negprop([a|T]) ← ¬ member(0,[a|T],a)   H/a   member(0,[H|T],H) ←
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(s(X),[a|T],a)
negprop([b|T]) ← ¬ member(0,[b|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding⁺; Negative Unfolding***

~~negprop([a|T]) ← ¬ member(0,[a|T],a)~~
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(s(X),[a|T],a)
negprop([b|T]) ← ¬ member(0,[b|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding$^+$; Negative Unfolding$^+$***

~~negprop([a|T]) ← ¬ member(0,[a|T],a)~~
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(s(X),[a|T],a)
negprop([b|T]) ← ¬ member(0,[b|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

H/a, S/a

member(s(X),[H|T],S) ← member(X,T,S)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding*** $^{+}$ ***; Negative Unfolding*** $^{+}$

~~negprop([a|T]) ← ¬ member(0,[a|T],a)~~
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)
negprop([b|T]) ← ¬ member(0,[b|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

Instantiation

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

Positive Unfolding⁺ ; Negative Unfolding⁺

~~negprop([a|T]) ← ¬ member(0,[a|T],a)~~
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)
negprop([b|T]) ← ¬ member(0,[b|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

member(0,[H|T],H) ←

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding^{+} ; Negative Unfolding^{+}***

~~negprop([a|T]) ← ¬ member(0,[a|T],a)~~
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)
negprop([b|T]) ← ~~¬ member(0,[b|T],a)~~
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding⁺ ; Negative Unfolding⁺***

~~negprop([a|T]) ← ¬ member(0,[a|T],a)~~
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)
negprop([b|T]) ← ~~¬ member(0,[b|T],a)~~
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

↓ ***Subsumption***

negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)
negprop([b|T]) ←
~~negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)~~

Transformation into Monadic ω-Program

negprop(L) ← position(X) ∧ even(X) ∧ ¬ member(X,L,a)   non monadic

↓ ***Instantiation***

negprop([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[a|T],a)
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,[b|T],a)

↓ ***Positive Unfolding⁺ ; Negative Unfolding⁺***

~~negprop([a|T]) ← ¬ member(0,[a|T],a)~~
negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)
negprop([b|T]) ← ~~¬ member(0,[b|T],a)~~
negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)

↓ ***Subsumption***

negprop([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)
negprop([b|T]) ←
~~negprop([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(s(X),[b|T],a)~~

↓ ***Positive Folding***   newp(T) ← position(X) ∧ ¬even(X) ∧ ¬member(X,T,a)

negprop([a|T]) ← newp(T)
negprop([b|T]) ←   monadic

Transformation into Monadic ω-Program

newp(T) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,T,a)   non monadic

Instantiation

newp([a|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,[a|T],a)
newp([b|T]) ← position(X) ∧ ¬ even(X) ∧ ¬ member(X,[b|T],a)

***Positive Unfolding*+**

~~newp([a|T]) ← ¬ even(0) ∧ ¬ member(0,[a|T],a)~~
newp([a|T]) ← position(X) ∧ ¬ even(s(X)) ∧ ¬ member(s(X),[a|T],a)
~~newp([b|T]) ← ¬ even(0) ∧ ¬ member(0,[b|T],a)~~
newp([b|T]) ← position(X) ∧ ¬ even(s(X)) ∧ ¬ member(s(X),[b|T],a)

***Negative Unfolding*+**

newp([a|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,T,a)
newp([b|T]) ← position(X) ∧ even(X) ∧ ¬ member(X,T,a)

***Positive Folding*+**   negprop (T) ← position(X) ∧¬even(X) ∧¬member(X,T,a)

newp([a|T]) ← negprop(T)
newp([b|T]) ← negprop(T)   monadic

Transformation into Monadic ω-Program

prop(L) ← ¬ negprop (L)   non monadic

Instantiation

prop([a|T]) ← ¬ negprop ([a|T])
prop([b|T]) ← ¬ negprop ([b|T])

Negative Unfolding^{+}

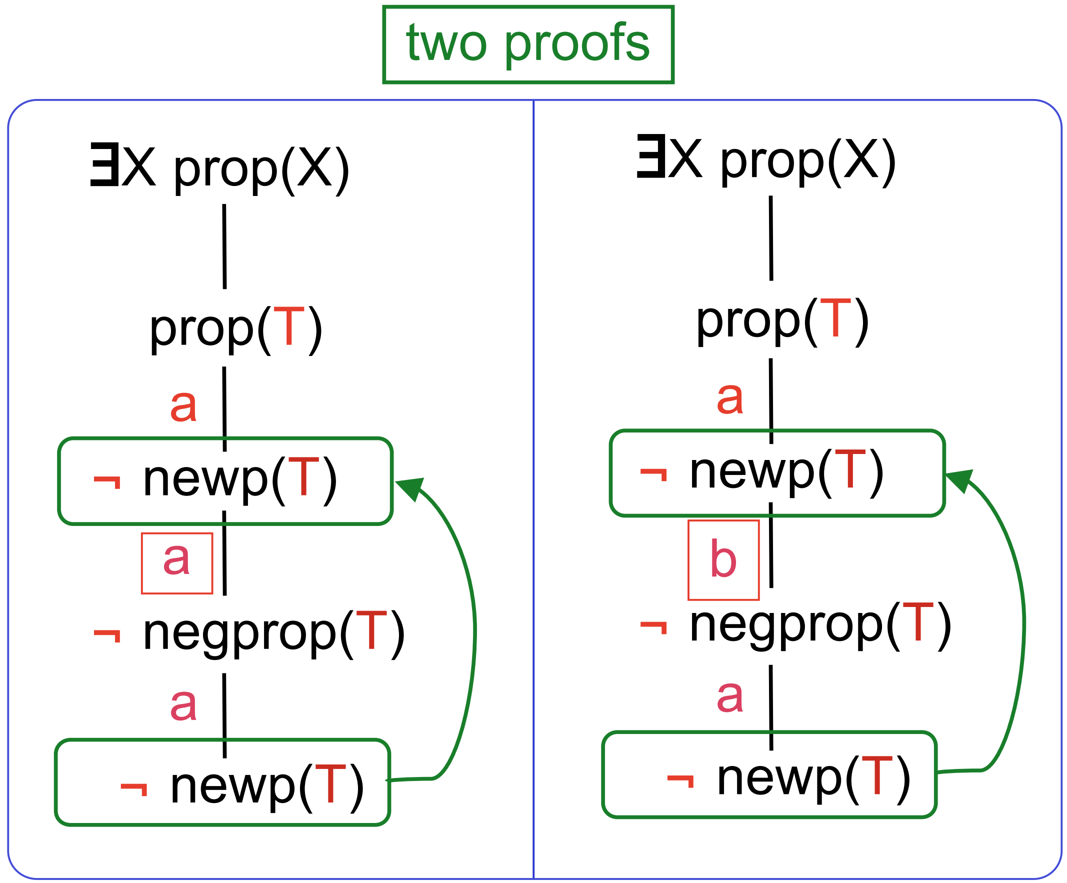
prop([a|T]) ← ¬ newp(T)   monadic

Monadic ω-Program T

T : prop([a|T]) ← ¬ newp(T)

newp([a|T]) ← negprop(T)
newp([b|T]) ← negprop(T)

negprop([a|T]) ← newp(T)
negprop([b|T]) ←



"two negative loops"

Thus, $a((a+b)a)^{\omega}$.

Plan of the Talk

- Specifying reactive systems by logic programs on infinite lists: ω-programs.

- Transformation rules for ω-programs.

- A transformation-based method for proving properties of ω-programs:
  - Strategy to get monadic ω-programs
  - Proof method for monadic ω-programs

The Transformation Strategy TransfM

Input: an ω-program P

Output: a monadic ω-program T such that

$$M(P) \models \exists X\ prop(X) \quad \text{iff} \quad M(T) \models \exists X\ prop(X)$$

---

1. *Specialize*(P,SpecP);
2. *Eliminate-Finite-Terms*(SpecP,T)

ω-regular Languages

$e ::= a \mid e_1\, e_2 \mid e_1+e_2 \mid e^*$ (regular expressions over Σ. symbol(a) iff $a \in \Sigma$)

$f ::= e^\omega \mid e_1\, e_2^{\,\omega} \mid f_1+f_2$ (ω-regular expressions over Σ. ε^ω not in f)

P:

$acc(E,[E]) \leftarrow symbol(E)$

$acc(E_1E_2,X) \leftarrow append(X_1,X_2,X) \land acc(E_1,X_1) \land acc(E_2,X_2)$

$acc(E_1+E_2,X) \leftarrow acc(E_1,X)$

$acc(E_1+E_2,X) \leftarrow acc(E_2,X)$

$acc(E^*,[\ ]) \leftarrow$

$acc(E^*,X) \leftarrow append(X_1,X_2,X) \land acc(E,X_1) \land acc(E^*,X_2)$

$\omega\text{-}acc(E^\omega,X) \leftarrow \neg\, new_1(E,X)$

$\omega\text{-}acc(E_1E_2^\omega, X) \leftarrow prefix(X,N,X_1) \land acc(E_1,X_1) \land \omega\text{-}acc1(E_2^\omega,X_1,X)$

$new_1(E,X) \leftarrow nat(M) \land \neg\, new_2(E,M,X)$

$new_2(E,M,X) \leftarrow geq(N,M) \land prefix(X,N,V) \land acc(E^*,V)$

$\omega\text{-}acc1(E,[\ ],X) \leftarrow \omega\text{-}acc(E,X)$

$\omega\text{-}acc1(E,[H|T],[H|X]) \leftarrow \omega\text{-}acc1(E,T,X)$

Containment of ω-regular Languages (1)

$expr_1(X) \leftarrow \omega\text{-}acc(a^\omega, X)$
$expr_2(X) \leftarrow \omega\text{-}acc((b^*a)^\omega, X)$
$prop(X) \leftarrow expr_1(X) \wedge \neg\, expr_2(X)$

$a^\omega \not\subseteq (b^*a)^\omega$

After *Specialize*:

SpecP:

$prop(X) \leftarrow expr_1(X) \wedge \neg\, expr_2(X)$
$expr_1(X) \leftarrow \neg\, new_1(X)$
$new_1(X) \leftarrow nat(M) \wedge \neg\, new_2(X,Y)$
$new_2(X,Y) \leftarrow geq(Z,Y) \wedge prefix(X,Z,W) \wedge new_3(W)$
$new_3([\,]) \leftarrow$
$new_3([a|X]) \leftarrow new_3(X)$
...

Containment of ω-regular Languages (2)

After *Eliminate-Finite-Terms*:

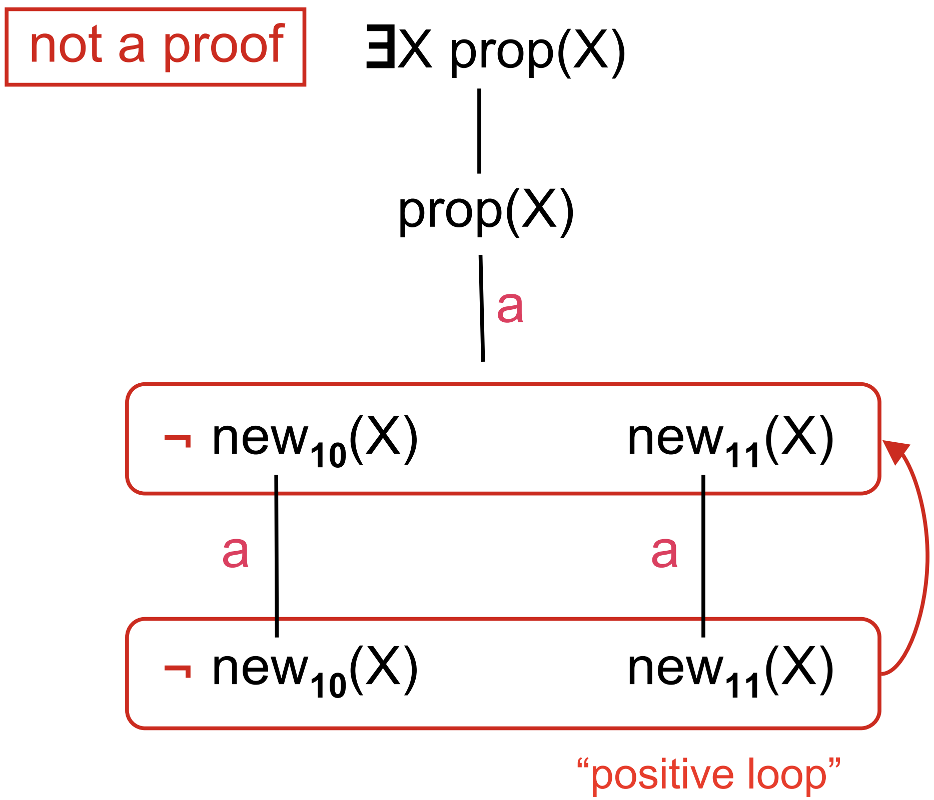
T:

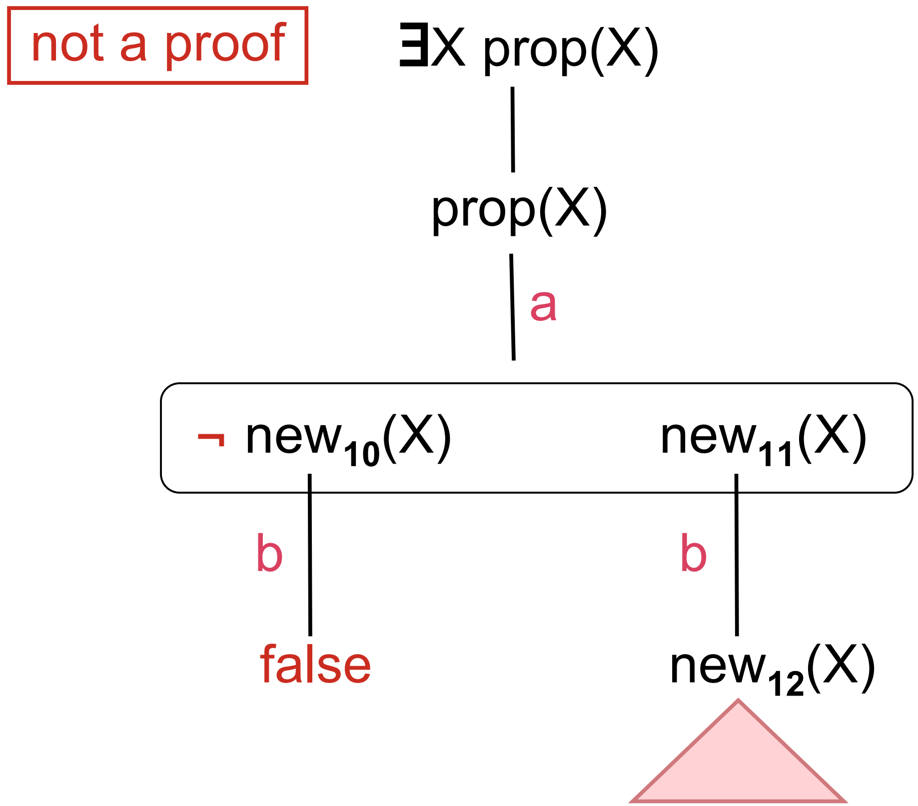
$prop([a|X]) \leftarrow \neg\ new_{10}(X) \wedge new_{11}(X)$
$new_{10}([a|X]) \leftarrow new_{10}(X)$
$new_{10}([b|X]) \leftarrow$
$new_{11}([a|X]) \leftarrow new_{11}(X)$
$new_{11}([b|X]) \leftarrow new_{12}(X)$
$new_{12}([a|X]) \leftarrow new_{11}(X)$
$new_{12}([b|X]) \leftarrow new_{12}(X)$
$new_{12}([b|X]) \leftarrow \neg\ new_{13}(X)$
$new_{13}([a|X]) \leftarrow$
$new_{13}([b|X]) \leftarrow new_{13}(X)$

Containment of ω-regular Languages (3)

T:

$prop([a|X]) \leftarrow \neg\, new_{10}(X) \wedge new_{11}(X)$
$new_{10}([a|X]) \leftarrow new_{10}(X)$
$new_{10}([b|X]) \leftarrow$
$new_{11}([a|X]) \leftarrow new_{11}(X)$
$new_{11}([b|X]) \leftarrow new_{12}(X)$
…





Containment of ω-regular Languages (4)

T:

$\text{prop}([a|X]) \leftarrow \neg\, \text{new}_{10}(X) \wedge \text{new}_{11}(X)$
$\text{new}_{10}([a|X]) \leftarrow \text{new}_{10}(X)$
$\text{new}_{10}([b|X]) \leftarrow$
$\text{new}_{11}([a|X]) \leftarrow \text{new}_{11}(X)$
$\text{new}_{11}([b|X]) \leftarrow \text{new}_{12}(X)$
...

not a proof

$\exists X\ \text{prop}(X)$

prop(X)

b

false

Thus, $a^{\omega} \subseteq (b^{*}a)^{\omega}$.

Future Work

- Consider infinite trees and other infinite structures.
- Synthesis of protocols and reactive systems.