Can we automate proof of contextual equivalence using Logical Relations?

Guilhem Jaber

PPS, IRIF, Université Paris Diderot

Workshop on Program Equivalence
April 12th 2016

Goals

- Present techniques to prove *contextual equivalence* of programs:
  - $\rightsquigarrow$ Program as a black-box
  - $\rightsquigarrow$ the context can call it many time and whenever he wants,

Goals

- Present techniques to prove *contextual equivalence* of programs:
  - $\rightsquigarrow$ Program as a black-box
  - $\rightsquigarrow$ the context can call it many time and whenever he wants,
- for a functional language:
  - $\rightsquigarrow$ Open programs and callbacks,
  - $\rightsquigarrow$ Reentrant calls,

Goals

- Present techniques to prove *contextual equivalence* of programs:
  - ⇝ Program as a black-box
  - ⇝ the context can call it many time and whenever he wants,
- for a functional language:
  - ⇝ Open programs and callbacks,
  - ⇝ Reentrant calls,
- with references:
  - ⇝ Local states,
  - ⇝ Disclosure between the program and the context.

Goals

- Present techniques to prove *contextual equivalence* of programs:
  - ⇝ Program as a black-box
  - ⇝ the context can call it many time and whenever he wants,
- for a functional language:
  - ⇝ Open programs and callbacks,
  - ⇝ Reentrant calls,
- with references:
  - ⇝ Local states,
  - ⇝ Disclosure between the program and the context.

Kripke Logical Relations

Goals

- Present techniques to prove *contextual equivalence* of programs:
  - ⇝ Program as a black-box
  - ⇝ the context can call it many time and whenever he wants,
- for a functional language:
  - ⇝ Open programs and callbacks,
  - ⇝ Reentrant calls,
- with references:
  - ⇝ Local states,
  - ⇝ Disclosure between the program and the context.

Kripke Logical Relations

Can we automate them ?

For what kind of Language: RefML

A typed functional programs : $\texttt{fun test}(\texttt{n}, \texttt{g}) = \texttt{n} + \texttt{g}(1)$

For what kind of Language: RefML

A typed functional programs : $\texttt{fun test}(\texttt{n}, \texttt{g}) = \texttt{n} + \texttt{g}(1)$

with Integers and Booleans: $\texttt{if b then 0 else n} + 1$

with pairs: $\langle u, v \rangle$

For what kind of Language: RefML

A typed functional programs : $\texttt{fun test}(\texttt{n}, \texttt{g}) = \texttt{n} + \texttt{g}(1)$

with Integers and Booleans: $\texttt{if b then 0 else n} + 1$

with pairs: $\langle u, v \rangle$

with higher-order references: $\mathrm{ref}\ 2, \mathrm{ref}\ (\lambda x.M)$

stored in heap via locations: $(\mathrm{ref}\ v, h) \to (\ell, h \cdot [\ell \mapsto v])$
(ℓ fresh in h)

mutable: $x := !x + 1$

For what kind of Language: RefML

A typed functional programs :	$\texttt{fun test}(\texttt{n}, \texttt{g}) = \texttt{n} + \texttt{g}(1)$
with Integers and Booleans:	$\texttt{if b then 0 else n} + 1$
with pairs:	$\langle u, v \rangle$
with higher-order references:	$\mathrm{ref}\, 2, \mathrm{ref}\, (\lambda x.M)$
stored in heap via locations:	$(\mathrm{ref}\, v, h) \to (\ell, h \cdot [\ell \mapsto v])$
	(ℓ fresh in h)
mutable:	$x :=\!!x + 1$
No pointer arithmetic:	$(\ell + 1)$ is ill-typed
But equality test:	$\ell_1 == \ell_2$ is well-typed

For what kind of Language: RefML

A typed functional programs :	$\texttt{fun test}(\texttt{n}, \texttt{g}) = \texttt{n} + \texttt{g}(1)$
with Integers and Booleans:	$\texttt{if b then 0 else n} + 1$
with pairs:	$\langle u, v \rangle$
with higher-order references:	$\mathrm{ref}\, 2, \mathrm{ref}\, (\lambda x.M)$
stored in heap via locations:	$(\mathrm{ref}\, v, h) \to (\ell, h \cdot [\ell \mapsto v])$ (ℓ fresh in h)
mutable:	$x :=\!!x + 1$
No pointer arithmetic:	$(\ell + 1)$ is ill-typed
But equality test:	$\ell_1 == \ell_2$ is well-typed

Full recursion (via Landin "knot").

Contextual Equivalence

Contextual equivalence of M_1, M_2:

$$\forall C.\forall h.(C[M_1] \Downarrow, h) \Longleftrightarrow (C[M_2] \Downarrow, h)$$

Observation $(M \Downarrow, h)$: Termination.

Contextual Equivalence

Contextual equivalence of M_1, M_2:

$$\forall C.\forall h.(C[M_1] \Downarrow, h) \Longleftrightarrow (C[M_2] \Downarrow, h)$$

Observation $(M \Downarrow, h)$: Termination.

- Robust w.r.t the choice of observation.
- Depend on the language contexts are written in.

Contextual Equivalence

Contextual equivalence of M_1, M_2:

$$\forall C.\forall h.(C[M_1] \Downarrow, h) \Longleftrightarrow (C[M_2] \Downarrow, h)$$

Observation $(M \Downarrow, h)$: Termination.

- Robust w.r.t the choice of observation.
- Depend on the language contexts are written in.
- Undecidable in general
  - $\rightsquigarrow$ Even in a finitary setting (finite datatypes, no recursion): Murawski & Tzevelekos

Contextual Equivalence

Contextual equivalence of M_1, M_2:

$$\forall C.\forall h.(C[M_1] \Downarrow, h) \Longleftrightarrow (C[M_2] \Downarrow, h)$$

Observation $(M \Downarrow, h)$: Termination.

- Robust w.r.t the choice of observation.
- Depend on the language contexts are written in.
- Undecidable in general
  - $\rightsquigarrow$ Even in a finitary setting (finite datatypes, no recursion): Murawski & Tzevelekos
- Many techniques to reason on contextual equivalence:
  - $\rightsquigarrow$ Algorithmic Game Semantics,
  - $\rightsquigarrow$ Environmental Bisimulations, Open Bisimulations,
  - $\rightsquigarrow$ Kripke Logical Relations.

Synchronization of Callbacks (I/II)

$\texttt{fun test}_1(\texttt{f}_1) = \texttt{f}_1()$ is **not** equivalent to
$\texttt{fun test}_2(\texttt{f}_2) = \texttt{f}_2(); \texttt{f}_2()$

Synchronization of Callbacks (I/II)

$$\texttt{fun test}_1(\texttt{f}_1) = \texttt{f}_1() \qquad \text{is } \textbf{not} \text{ equivalent to}$$
$$\texttt{fun test}_2(\texttt{f}_2) = \texttt{f}_2(); \texttt{f}_2()$$

- $\rightsquigarrow$ Contexts can check how many time $\texttt{f}_1, \texttt{f}_2$ are called.
- $\rightsquigarrow$ Callbacks are fully observable!

$C[\bullet] \stackrel{def}{=} \texttt{let x} = \text{ref } 0 \texttt{ in } \bullet\,(\lambda\_.\texttt{x} := !\texttt{x} + 1); \texttt{if } !\texttt{x} > 1 \texttt{ then } \Omega \texttt{ else}()$
can discriminate them.

Synchronization of Callbacks (II/II)

$$\texttt{fun add}_1(\texttt{f}_1) = (\texttt{f}_1\ 1) + (\texttt{f}_1\ 2) \qquad \text{is } \textbf{not} \text{ equivalent to}$$
$$\texttt{fun add}_2(\texttt{f}_2) = (\texttt{f}_2\ 2) + (\texttt{f}_2\ 1)$$

Synchronization of Callbacks (II/II)

$$\texttt{fun add}_1(\texttt{f}_1) = (\texttt{f}_1\ 1) + (\texttt{f}_1\ 2) \qquad \text{is } \textbf{not} \text{ equivalent to}$$
$$\texttt{fun add}_2(\texttt{f}_2) = (\texttt{f}_2\ 2) + (\texttt{f}_2\ 1)$$

$\rightsquigarrow$ Arguments given to callbacks must be related.

$C[\bullet] \stackrel{def}{=} \texttt{let x} = \mathrm{ref}\ 0\ \texttt{in}\ \bullet\,(\lambda \texttt{y.x} := \texttt{y}); \texttt{if !x} == 1\ \texttt{then}\ \Omega\ \texttt{else}()$
can discriminate them.

Disclosure of Locations (I/II)

$$\texttt{fun const}_1() = \texttt{let x = ref0 in 1} \quad \text{is equivalent to} \quad \texttt{fun const}_2() = \texttt{1}$$

- $\rightsquigarrow$ The creation of the reference bounded to x is not observable by the context.
- $\rightsquigarrow$ It is private to the term!

Disclosure of Locations (II/II)

$$\texttt{fun discl}_1\texttt{(f)} = \texttt{let x = ref0 in fx; x} := 1$$

is **not** equivalent to

$$\texttt{fun discl}_2\texttt{(f)} = \texttt{let x = ref0 in fx; x} := 2$$

Disclosure of Locations (II/II)

$$\texttt{fun discl}_1\texttt{(f)} = \texttt{let x = ref0 in fx; x} := \texttt{1}$$

is **not** equivalent to

$$\texttt{fun discl}_2\texttt{(f)} = \texttt{let x = ref0 in fx; x} := \texttt{2}$$

- $\rightsquigarrow$ The reference bound to x is disclosed to the context.
- $\rightsquigarrow$ It can look inside afterwards to see what is stored.

$C[\bullet] \stackrel{def}{=} \texttt{let z} = \mathrm{ref}\,(\mathrm{ref}\,0)\ \texttt{in}\ \bullet\,(\lambda\texttt{y.z} := \texttt{y}); \texttt{if !!z == 1 then } \Omega\ \texttt{else()}$
can discriminate them.

Representation Independence

The two following programs are equivalent:

$$
\begin{array}{l}
\texttt{let } \mathtt{c}_1 \;=\; \texttt{ref0} \\
\qquad\qquad\quad \texttt{fun } \mathtt{inc}_1().\mathtt{c}_1 := !\mathtt{c}_1 + 1 \\
\qquad\qquad\quad \texttt{fun } \mathtt{get}_1().!\mathtt{c}_1 \\
\texttt{in } \langle \mathtt{inc}_1, \mathtt{get}_1 \rangle
\end{array}
$$

$$
\begin{array}{l}
\texttt{let } \mathtt{c}_2 \;=\; \texttt{ref0} \\
\qquad\qquad\quad \texttt{fun } \mathtt{inc}_2().\mathtt{c}_2 := !\mathtt{c}_2 - 1 \\
\qquad\qquad\quad \texttt{fun } \mathtt{get}_2().-!\mathtt{c}_2 \\
\texttt{in } \langle \mathtt{inc}_2, \mathtt{get}_2 \rangle
\end{array}
$$

Need a relational invariant between c_1 and c_2.

Quiz

Are the two following programs equivalent?

$$
\begin{array}{l}
\texttt{let } \mathtt{c}_1 \texttt{ = ref0} \\
\qquad\qquad \texttt{fun } \mathtt{inc}_1(\mathtt{f}) = \mathtt{f}(); \mathtt{c}_1 := !\mathtt{c}_1 + 1 \\
\qquad\qquad \texttt{fun } \mathtt{get}_1() = !\mathtt{c}_1 \\
\texttt{in } \langle \mathtt{inc}_1, \mathtt{get}_1 \rangle
\end{array}
$$

$$
\begin{array}{l}
\texttt{let } \mathtt{c}_2 \texttt{ = ref0} \\
\qquad\qquad \texttt{fun } \mathtt{inc}_2(\mathtt{f}) = \texttt{ let n } = !\mathtt{c}_2 \texttt{ in } \mathtt{f}(); \mathtt{c}_2 := \mathtt{n} + 1 \\
\qquad\qquad \texttt{fun } \mathtt{get}_2() = !\mathtt{c}_2 \\
\texttt{in } \langle \mathtt{inc}_2, \mathtt{get}_2 \rangle
\end{array}
$$

Quiz

Are the two following programs equivalent?

$$
\begin{array}{l}
\texttt{let } \mathtt{c}_1 \;=\; \texttt{ref0} \\
\qquad\qquad\quad \texttt{fun } \mathtt{inc}_1(\mathtt{f}) = \mathtt{f}(); \mathtt{c}_1 := !\mathtt{c}_1 + 1 \\
\qquad\qquad\quad \texttt{fun } \mathtt{get}_1() = !\mathtt{c}_1 \\
\texttt{in } \langle \mathtt{inc}_1, \mathtt{get}_1 \rangle \\
\hline
\texttt{let } \mathtt{c}_2 \;=\; \texttt{ref0} \\
\qquad\qquad\quad \texttt{fun } \mathtt{inc}_2(\mathtt{f}) = \texttt{ let n } = !\mathtt{c}_2 \texttt{ in } \mathtt{f}(); \mathtt{c}_2 := \mathtt{n} + 1 \\
\qquad\qquad\quad \texttt{fun } \mathtt{get}_2() = !\mathtt{c}_2 \\
\texttt{in } \langle \mathtt{inc}_2, \mathtt{get}_2 \rangle
\end{array}
$$

No, because of reentrant calls !

$$
C[\bullet] \stackrel{def}{=} \texttt{let } \langle \mathtt{inc}, \mathtt{get} \rangle \;=\; \bullet \texttt{ in let d } =\; \mathtt{get}() \texttt{ in} \\
\qquad\qquad \mathtt{inc}(\lambda\_.\mathtt{inc}\ (\lambda \mathtt{x}.\mathtt{x})); \texttt{ if } \mathtt{get}() \neq \mathtt{d} + 2 \texttt{ then } \Omega \texttt{ else } ()
$$

can discriminate them.

Logical Relations

Binary relations $\mathcal{E}[\![\tau]\!], \mathcal{V}[\![\tau]\!]$ on closed terms and values

$\rightsquigarrow$ inductively defined on types.

$$\mathcal{V}[\![\mathrm{Int}]\!] \stackrel{def}{=} \{(n, n) \mid n \in \mathbb{Z}\}$$

$$\mathcal{V}[\![\tau \to \sigma]\!] \stackrel{def}{=} \{(\lambda x_1.M_1, \lambda x_2.M_2) \mid \forall (v_1, v_2) \in \mathcal{V}[\![\tau]\!]. \\ ((\lambda x_1.M_1)v_1, (\lambda x_2.M_2)v_2) \in \mathcal{E}[\![\sigma]\!]\}$$

$$\mathcal{E}[\![\tau]\!] \stackrel{def}{=} \{(M_1, M_2) \mid (M_1 \Uparrow \wedge M_2 \Uparrow) \\ \vee((M_1 \mapsto^* v_1) \wedge (M_2 \mapsto^* v_2) \wedge (v_1, v_2) \in \mathcal{V}[\![\tau]\!]\}$$

Operational reasoning for functions with local state

Andrew Pitts and Ian Stark

Abstract

Languages such as ML or Lisp permit the use of recursively defined function expressions with locally declared storage locations. Although this can be very convenient from a programming point of view it severely complicates the properties of program equivalence even for relatively simple fragments of such languages—such as the simply typed fragment of Standard ML with integer-valued references considered here. This paper presents a method for reasoning about *contextual equivalence* of programs involving this combination of functional and procedural features. The method is based upon the use of a certain kind of *logical relation* parameterised by relations between program states. The form of this logical relation is novel, in as much as it involves relations not only between program expressions, but also between program continuations (also known as *evaluation contexts*). The authors found this approach necessary in order to establish the 'Fundamental Property of logical relations' in the presence of both dynamically allocated local state and recursion. The logical relation characterises contextual equivalence and yields a proof of the best known context lemma for this kind of language—the Mason-Talcott 'ciu' theorem. Moreover, it is shown that the method can prove examples where such a context lemma is not much help and which involve representation independence, higher order memoising functions, and profiling functions.

1 Introduction

Kripke Logical Relations

Extension to languages with references.

- $\rightsquigarrow$ Need *worlds* w, i.e. invariants on heaps,
- $\rightsquigarrow$ Extensible on disjoint parts : $w' \sqsupseteq w$
- $\rightsquigarrow$ Parametrize the definition of logical relations with such worlds.

Kripke Logical Relations

Extension to languages with references.

- $\rightsquigarrow$ Need *worlds* w, i.e. invariants on heaps,
- $\rightsquigarrow$ Extensible on disjoint parts : $w' \sqsupseteq w$
- $\rightsquigarrow$ Parametrize the definition of logical relations with such worlds.

$$\mathcal{E}\,[\![\tau]\!]\,w \stackrel{def}{=} \Big\{(M_1, M_2) \mid \forall (h_1, h_2) : w.\big((M_1, h_1) \Uparrow \wedge (M_2, h_2) \Uparrow\big) \vee \big(((M_1, h_1) \mapsto^* (v_1, h_1')) \wedge ((M_2, h_2) \mapsto^* (v_2, h_2'))\big) \exists w' \sqsupseteq w.(h_1', h_2') \in w' \wedge (v_1, v_2) \in \mathcal{V}\,[\![\tau]\!]\,w'\big)\Big\}$$

Kripke Logical Relations

Extension to languages with references.

- $\rightsquigarrow$ Need *worlds* w, i.e. invariants on heaps,
- $\rightsquigarrow$ Extensible on disjoint parts : $w' \sqsupseteq w$
- $\rightsquigarrow$ Parametrize the definition of logical relations with such worlds.

$$\mathcal{E}\,[\![\tau]\!]\,w \stackrel{def}{=} \Big\{(M_1, M_2) \mid \forall (h_1, h_2) : w.\big((M_1, h_1) \Uparrow \wedge (M_2, h_2) \Uparrow\big) \vee \big(((M_1, h_1) \mapsto^* (v_1, h_1')) \wedge ((M_2, h_2) \mapsto^* (v_2, h_2'))\big) \exists w' \sqsupseteq w.(h_1', h_2') \in w' \wedge (v_1, v_2) \in \mathcal{V}\,[\![\tau]\!]\,w'\big)\Big\}$$

$$\mathcal{V}\,[\![\tau \to \sigma]\!]\,w \stackrel{def}{=} \{(\lambda x_1.M_1, \lambda x_2.M_2) \mid \forall w' \sqsupseteq w.\forall (v_1, v_2) \in \mathcal{V}\,[\![\tau]\!]\,w'. ((\lambda x_1.M_1)v_1, (\lambda x_2.M_2)v_2) \in \mathcal{E}\,[\![\sigma]\!]\,w'\}$$

Equivalence of the Representation Independence example (I/II)

$$
\begin{array}{l}
\texttt{let } \mathtt{c}_1 \;=\; \texttt{ref0} \\
\qquad\qquad\quad \texttt{fun } \mathtt{inc}_1() = \mathtt{c}_1 := !\mathtt{c}_1 + 1 \\
\qquad\qquad\quad \texttt{fun } \mathtt{get}_1() = !\mathtt{c}_1 \\
\texttt{in } \langle \mathtt{inc}_1, \mathtt{get}_1 \rangle
\end{array}
\qquad
\begin{array}{l}
\texttt{let } \mathtt{c}_2 \;=\; \texttt{ref0} \\
\qquad\qquad\quad \texttt{fun } \mathtt{inc}_2() = \mathtt{c}_2 := !\mathtt{c}_2 - 1 \\
\qquad\qquad\quad \texttt{fun } \mathtt{get}_2() = -!\mathtt{c}_2 \\
\texttt{in } \langle \mathtt{inc}_2, \mathtt{get}_2 \rangle
\end{array}
$$

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$
- Prove $(\texttt{inc}_1, \texttt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$
- Prove $(\texttt{inc}_1, \texttt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$
- Prove $(\mathtt{inc}_1, \mathtt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\mathtt{inc_i}(), h_i) \mapsto^* ((), h_i')$,

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$

- Prove $(\texttt{inc}_1, \texttt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\texttt{inc}_\texttt{i}(), h_i) \mapsto^* ((), h_i')$,
  - $\rightsquigarrow$ $h_1' = h_1[c \mapsto (h_1(c) + 1)]$ and $h_2' = h_2[c \mapsto (h_2(c) - 1)]$,

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$

- Prove $(\texttt{inc}_1, \texttt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\texttt{inc}_\texttt{i}(), h_i) \mapsto^* ((), h'_i)$,
  - $\rightsquigarrow$ $h'_1 = h_1[c \mapsto (h_1(c) + 1)]$ and $h'_2 = h_2[c \mapsto (h_2(c) - 1)]$,
  - $\rightsquigarrow$ So that $(h'_1, h'_2) \in w$!

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$
- Prove $(\mathtt{inc}_1, \mathtt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\mathtt{inc_i}(), h_i) \mapsto^* ((), h_i')$,
  - $\rightsquigarrow$ $h_1' = h_1[c \mapsto (h_1(c) + 1)]$ and $h_2' = h_2[c \mapsto (h_2(c) - 1)]$,
  - $\rightsquigarrow$ So that $(h_1', h_2') \in w$!
- Prove $(\mathtt{get}_1, \mathtt{get}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Int}]\!]\,w$

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$
- Prove $(\texttt{inc}_1, \texttt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\texttt{inc}_i(), h_i) \mapsto^* ((), h_i')$,
  - $\rightsquigarrow$ $h_1' = h_1[c \mapsto (h_1(c) + 1)]$ and $h_2' = h_2[c \mapsto (h_2(c) - 1)]$,
  - $\rightsquigarrow$ So that $(h_1', h_2') \in w$!
- Prove $(\texttt{get}_1, \texttt{get}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Int}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$
- Prove $(\texttt{inc}_1, \texttt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\texttt{inc}_i(), h_i) \mapsto^* ((), h_i')$,
  - $\rightsquigarrow$ $h_1' = h_1[c \mapsto (h_1(c) + 1)]$ and $h_2' = h_2[c \mapsto (h_2(c) - 1)]$,
  - $\rightsquigarrow$ So that $(h_1', h_2') \in w$!
- Prove $(\texttt{get}_1, \texttt{get}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Int}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\texttt{get}_1(), h_1) \mapsto^* (h_1(c_1), h_1)$ and $(\texttt{get}_2, h_2) \mapsto^* (-h_2(c_2), h_2)$,

Equivalence of the Representation Independence example (I/II)

- Take $w = \{(h_1, h_2) \mid h_1(c_1) = -h_2(c_2)\}$
- Prove $(\mathtt{inc}_1, \mathtt{inc}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Unit}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\mathtt{inc_i}(), h_i) \mapsto^* ((), h_i')$,
  - $\rightsquigarrow$ $h_1' = h_1[c \mapsto (h_1(c) + 1)]$ and $h_2' = h_2[c \mapsto (h_2(c) - 1)]$,
  - $\rightsquigarrow$ So that $(h_1', h_2') \in w$!
- Prove $(\mathtt{get}_1, \mathtt{get}_2) \in \mathcal{V}\,[\![\mathrm{Unit} \to \mathrm{Int}]\!]\,w$
  - $\rightsquigarrow$ Take $(h_1, h_2) \in w$,
  - $\rightsquigarrow$ $(\mathtt{get}_1(), h_1) \mapsto^* (h_1(c_1), h_1)$ and $(\mathtt{get}_2, h_2) \mapsto^* (-h_2(c_2), h_2)$,
  - $\rightsquigarrow$ Need to prove $h_1(c_1) = -h_2(c_2)$: Straightforward from $(h_1, h_2) \in w$!

Invariants are not Enough

$$\texttt{let x} = \texttt{ref0 in fun awk}_1\texttt{(f)} = \texttt{x} := \texttt{1}; \texttt{f}(); !\texttt{x} \;\simeq\; \texttt{fun awk}_2\texttt{(f)} = \texttt{f}(); \texttt{1}$$

Invariants are not Enough

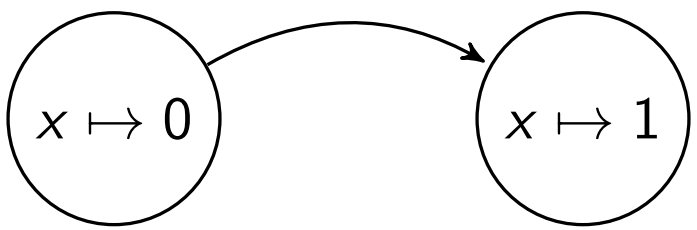
$$\texttt{let x} = \texttt{ref0 in fun awk}_1(\texttt{f}) = \texttt{x} := 1; \texttt{f}(); !\texttt{x} \;\simeq\; \texttt{fun awk}_2(\texttt{f}) = \texttt{f}(); 1$$

Need transition system of Invariants !

Invariants are not Enough

$$\texttt{let x} = \texttt{ref0 in fun awk}_1(\texttt{f}) = \texttt{x} := 1; \texttt{f}(); !\texttt{x} \;\simeq\; \texttt{fun awk}_2(\texttt{f}) = \texttt{f}(); 1$$

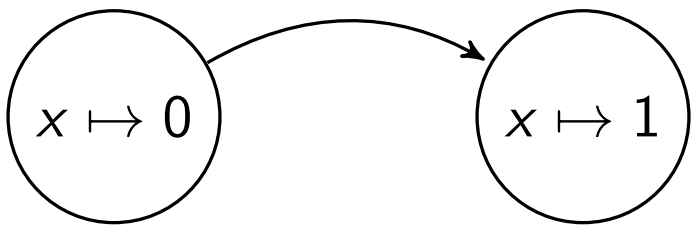
Need transition system of Invariants !



Invariants are not Enough

$$\texttt{let x} = \texttt{ref0 in fun awk}_1(\texttt{f}) = \texttt{x} := 1; \texttt{f}(); !\texttt{x} \simeq \texttt{fun awk}_2(\texttt{f}) = \texttt{f}(); 1$$

Need transition system of Invariants !



- Future world $w' \sqsupseteq w$ can either:
  - Add a new transition system of invariants on a disjoint part of the heaps,
  - Take a transition to get the following invariant.

A Landmark Paper

The Impact of Higher-Order State and Control Effects on Local Relational Reasoning

Derek Dreyer
MPI-SWS
dreyer@mpi-sws.org

Georg Neis
MPI-SWS
neis@mpi-sws.org

Lars Birkedal
IT University of Copenhagen
birkedal@itu.dk

Abstract

Reasoning about program equivalence is one of the oldest problems in semantics. In recent years, useful techniques have been developed, based on bisimulations and logical relations, for reasoning about equivalence in the setting of increasingly realistic languages—languages nearly as complex as ML or Haskell. Much of the recent work in this direction has considered the interesting representation independence principles *enabled* by the use of local state, but it is also important to understand the principles that powerful features like higher-order state and control effects *disable*. This latter topic has been broached extensively within the framework of game semantics, resulting in what Abramsky dubbed the "semantic cube": fully abstract game-semantic characterizations of various axes in the design space of ML-like languages. But when it comes to reasoning about many actual examples, game semantics does not yet supply a useful technique for proving equivalences.

In this paper, we marry the aspirations of the semantic cube to the powerful proof method of *step-indexed Kripke logical relations*. Building on recent work of Ahmed, Dreyer, and Rossberg, we define the first fully abstract logical relation for an ML-like language with recursive types, abstract types, general references and call/cc. We then show how, under orthogonal restrictions to the expressive power of our language—namely, the restriction to first-order state and/or the removal of call/cc—we can enhance the proving power of our possible-worlds model in correspondingly orthogonal ways, and we demonstrate this proving power on a range of interesting examples. Central to our story is the use of *state transition systems* to model the way in which properties of local state evolve over time.

1. Introduction

Reasoning about program equivalence is one of the oldest problems in semantics, with applications to program verification ("Is an optimized program equivalent to some reference implementation?"), compiler correctness ("Does a program transformation preserve the semantics of the source program?"), representation independence ("Can we modify the internal representation of an abstract data type without affecting the behavior of clients?"), and more besides.

The canonical notion of program equivalence for many applications is *observational* (or *contextual*) equivalence. Two programs are observationally equivalent if no program context can distinguish them by getting them to exhibit observably different input/output behavior. Reasoning about observational equivalence directly is difficult, due to the universal quantification over program contexts. Consequently, there has been a huge amount of work on developing useful models and logics for observational equivalence, and in recent years this line of work has scaled to handle increasingly realistic languages—languages nearly as complex as ML or Haskell, with features like general recursive types, general (higher-order) mutable references, and first-class continuations.

The focus of much of this recent work—*e.g.*, environmental bisimulations [36, 17, 32, 35], normal form bisimulations [34, 16], step-indexed Kripke logical relations [4, 2, 3]—has been on establishing some effective techniques for reasoning about programs that actually *use* the interesting, semantically complex features (state, continuations, etc.) of the languages being modeled. For instance, most of the work on languages with state concerns the various kinds of representation independence principles that arise due to the use of *local state* as an abstraction mechanism.

How to automate reasoning on Kripke Logical Relations

Obstacles to automate Kripke Logical Relations:

- Definition of $\mathcal{V}[\![\tau \to \sigma]\!]$ quantify over pairs of values in $\mathcal{V}[\![\tau]\!]$,
- Future worlds $w' \sqsupseteq w$ can add arbitrary new systems of invariants.

Solutions:

How to automate reasoning on Kripke Logical Relations

Obstacles to automate Kripke Logical Relations:

- Definition of $\mathcal{V}[\![\tau \to \sigma]\!]$ quantify over pairs of values in $\mathcal{V}[\![\tau]\!]$,
- Future worlds $w' \sqsupseteq w$ can add arbitrary new systems of invariants.

Solutions:

- Reason on **open** terms with **functional names** (i.e. uninterpreted functions)
  - $\rightsquigarrow$ Make the full control flow apparent in the operational reduction,
  - $\rightsquigarrow$ Keep track of related functional names in an environment e,

How to automate reasoning on Kripke Logical Relations

Obstacles to automate Kripke Logical Relations:

- Definition of $\mathcal{V}[\![\tau \to \sigma]\!]$ quantify over pairs of values in $\mathcal{V}[\![\tau]\!]$,
- Future worlds $w' \sqsupseteq w$ can add arbitrary new systems of invariants.

Solutions:

- Reason on **open** terms with **functional names** (i.e. uninterpreted functions)
  - $\rightsquigarrow$ Make the full control flow apparent in the operational reduction,
  - $\rightsquigarrow$ Keep track of related functional names in an environment e,
- Fix a "World Transition System" $\mathcal{A}$:
  - $\rightsquigarrow$ Transitions specify only the evolution of worlds
  - $\rightsquigarrow$ Transition system represents the control flow between the term and its environment.

How to automate reasoning on Kripke Logical Relations

Obstacles to automate Kripke Logical Relations:

- Definition of $\mathcal{V}[\![\tau \to \sigma]\!]$ quantify over pairs of values in $\mathcal{V}[\![\tau]\!]$,
- Future worlds $w' \sqsupseteq w$ can add arbitrary new systems of invariants.

Solutions:

- Reason on **open** terms with **functional names** (i.e. uninterpreted functions)
  - $\rightsquigarrow$ Make the full control flow apparent in the operational reduction,
  - $\rightsquigarrow$ Keep track of related functional names in an environment e,
- Fix a "World Transition System" $\mathcal{A}$:
  - $\rightsquigarrow$ Transitions specify only the evolution of worlds
  - $\rightsquigarrow$ Transition system represents the control flow between the term and its environment.

How to automate reasoning on Kripke Logical Relations

Obstacles to automate Kripke Logical Relations:

- Definition of $\mathcal{V}[\![\tau \to \sigma]\!]$ quantify over pairs of values in $\mathcal{V}[\![\tau]\!]$,
- Future worlds $w' \sqsupseteq w$ can add arbitrary new systems of invariants.

Solutions:

- Reason on **open** terms with **functional names** (i.e. uninterpreted functions)
  - $\rightsquigarrow$ Make the full control flow apparent in the operational reduction,
  - $\rightsquigarrow$ Keep track of related functional names in an environment e,
- Fix a "World Transition System" $\mathcal{A}$:
  - $\rightsquigarrow$ Transitions specify only the evolution of worlds
  - $\rightsquigarrow$ Transition system represents the control flow between the term and its environment.

How to automate reasoning on Kripke Logical Relations

Obstacles to automate Kripke Logical Relations:

- Definition of $\mathcal{V}[\![\tau \to \sigma]\!]$ quantify over pairs of values in $\mathcal{V}[\![\tau]\!]$,
- Future worlds $w' \sqsupseteq w$ can add arbitrary new systems of invariants.

Solutions:

- Reason on **open** terms with **functional names** (i.e. uninterpreted functions)
  - $\rightsquigarrow$ Make the full control flow apparent in the operational reduction,
  - $\rightsquigarrow$ Keep track of related functional names in an environment e,
- Fix a "World Transition System" $\mathcal{A}$:
  - $\rightsquigarrow$ Transitions specify only the evolution of worlds
  - $\rightsquigarrow$ Transition system represents the control flow between the term and its environment.

Give rise to our definition of "**Kripke Open Bisimulations**"
(Joint Work with **N. Tabareau**)
(Published at APLAS'15)

Relations on Terms

$(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e \, w$ when, for all $(h_1, h_2) \in w$,

Relations on Terms

$(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e \, w$ when, for all $(h_1, h_2) \in w$,

- Either both $(M_1, h_1) \Uparrow$ and $(M_2, h_2) \Uparrow$,

Relations on Terms

$(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e \, w$ when, for all $(h_1, h_2) \in w$,

- Either both $(M_1, h_1) \Uparrow$ and $(M_2, h_2) \Uparrow$,
- Or both $(M_1, h_1) \mapsto^* (v_1, h_1')$ and $(M_2, h_2) \mapsto^* (v_2, h_2')$ and there exists $w' \sqsupseteq_{\mathcal{A}} w$ with $(h_1', h_2') \in w'$ and $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\tau]\!]_e \, w'$,

Relations on Terms

$(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e \, w$ when, for all $(h_1, h_2) \in w$,

- Either both $(M_1, h_1) \Uparrow$ and $(M_2, h_2) \Uparrow$,
- Or both $(M_1, h_1) \mapsto^* (v_1, h_1')$ and $(M_2, h_2) \mapsto^* (v_2, h_2')$ and there exists $w' \sqsupseteq_{\mathcal{A}} w$ with $(h_1', h_2') \in w'$ and $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\tau]\!]_e \, w'$,
- Or both $(M_1, h_1) \mapsto^* (K_1[f_1 \ v_1], h_1')$ and $(M_2, h_2) \mapsto^* (K_2[f_2 \ v_2], h_2')$ with $(f_1, f_2, \sigma \to \sigma') \in e$ and there exists $w' \sqsupseteq_{\mathcal{A}} w$ with $(h_1', h_2') \in w'$ and $(K_1, K_2) \in \mathcal{K}_{\mathcal{A}} [\![\sigma', \tau]\!]_e \, w'$, $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_e \, w'$,

Relations on Terms

$(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e \, w$ when, for all $(h_1, h_2) \in w$,

- Either both $(M_1, h_1) \Uparrow$ and $(M_2, h_2) \Uparrow$,
- Or both $(M_1, h_1) \mapsto^* (v_1, h_1')$ and $(M_2, h_2) \mapsto^* (v_2, h_2')$ and there exists $w' \sqsupseteq_{\mathcal{A}} w$ with $(h_1', h_2') \in w'$ and $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\tau]\!]_e \, w'$,
- Or both $(M_1, h_1) \mapsto^* (K_1[f_1 \ v_1], h_1')$ and $(M_2, h_2) \mapsto^* (K_2[f_2 \ v_2], h_2')$ with $(f_1, f_2, \sigma \to \sigma') \in e$ and there exists $w' \sqsupseteq_{\mathcal{A}} w$ with $(h_1', h_2') \in w'$ and $(K_1, K_2) \in \mathcal{K}_{\mathcal{A}} [\![\sigma', \tau]\!]_e \, w'$, $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_e \, w'$,
- (+ deferred divergence).

Soundness and Completeness

Theorem

Two terms M_1, M_2 of type τ are contextually equivalent iff there exists a WTS $\mathcal{A}$ s.t. $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!] w_0$.

- Proof: Correspondence with "simple" bisimulations on the LTS generating traces.
- Soundness:
  - $\rightsquigarrow$ WTS $\mathcal{A}$ as an abstraction of the LTS generating traces.
- Completeness:
  - $\rightsquigarrow$ Not for free: no more biorthogonality,
  - $\rightsquigarrow$ Need *exhaustive* WTS,
  - $\rightsquigarrow$ First complete method which does not rely on closure in its definition.

Further in Abstraction : Symbolic and Temporal Reasoning

- $\mathcal{E}_{\mathcal{A}}\,[\![\tau]\!]_e\, w$ uses operational reduction
  - $\rightsquigarrow$ Replace it with symbolic execution,
  - $\rightsquigarrow$ Generate arithmetic constraints,
  - $\rightsquigarrow$ Block on callbacks and recursive calls.

- Quantification over future worlds
  - $\rightsquigarrow$ Abstract over it using temporal logic,
  - $\rightsquigarrow$ Temporal modalities $\Box\phi, X(\phi)$.

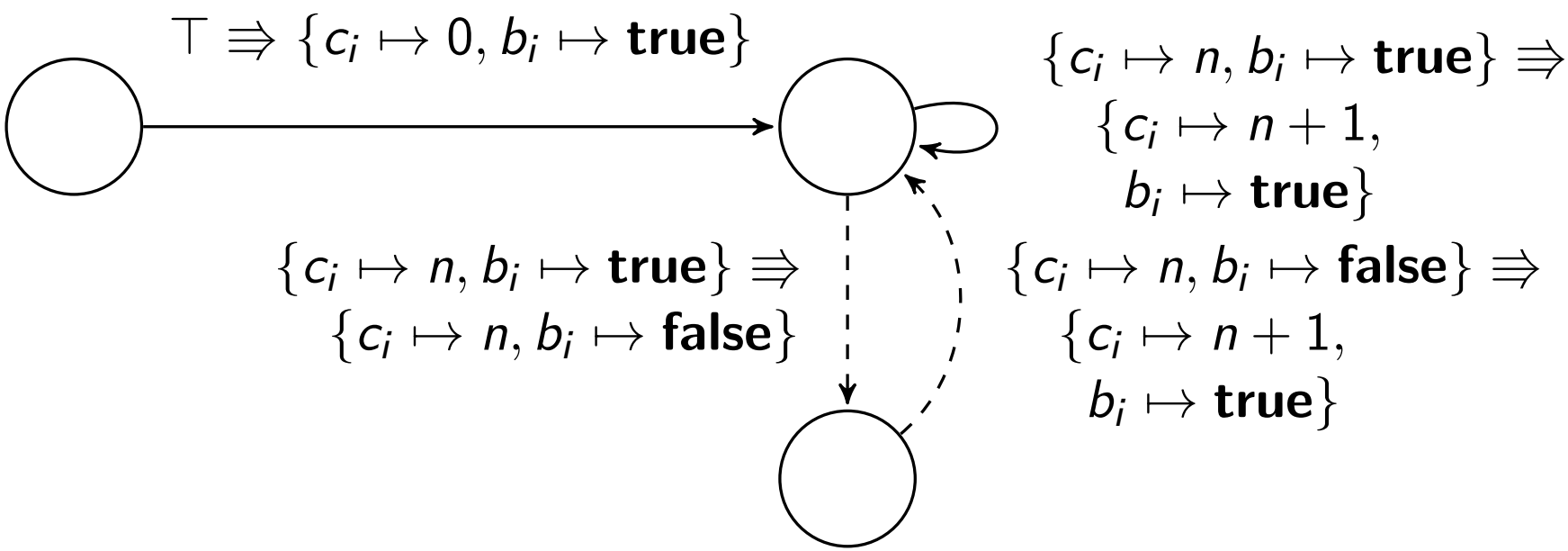
- Give rise to Temporal Symbolic Kripke Open Bisimulations $\mathbb{E}[\![\tau]\!]$.
  - $\rightsquigarrow$ Model check $\mathcal{A} \models \mathbb{E}[\![\tau]\!]$.

An Example: Callback with lock

$$
\begin{array}{l|l}
\texttt{let}\ \mathtt{b}_1 = \mathrm{ref}\ \textbf{true},\ \mathtt{c}_1 = \mathrm{ref}\ 0, & \texttt{let}\ \mathtt{b}_2 = \mathrm{ref}\ \textbf{true},\ \mathtt{c}_2 = \mathrm{ref}\ 0, \\
\quad \texttt{fun}\ \mathtt{inc}_1(\mathtt{f}) = \texttt{if}\ !\mathtt{b}_1\ \texttt{then}\{ & \quad \texttt{fun}\ \mathtt{inc}_2(\mathtt{f}) = \texttt{if}\ !\mathtt{b}_2\ \texttt{then}\ \{ \\
\qquad \mathtt{b}_1 := \textbf{false}; & \qquad \mathtt{b}_2 := \textbf{false}; \\
\qquad f(); c_1 := !c_1 + 1; & \qquad \mathit{let}\ n = !c_2\ \mathit{in}\ f(); c_2 := n + 1; \\
\qquad \mathtt{b}_1 := \textbf{true}\}\ \texttt{else}\ (), & \qquad \mathtt{b}_2 := \textbf{true}\}\ \texttt{else}\ (), \\
\quad \texttt{fun}\ \mathtt{get}_1() = !\mathtt{c}_1 & \quad \texttt{fun}\ \mathtt{get}_2() = !\mathtt{c}_2 \\
\texttt{in}\ \langle \mathtt{inc}_1, \mathtt{get}_1 \rangle & \texttt{in}\ \langle \mathtt{inc}_2, \mathtt{get}_2 \rangle
\end{array}
$$

An Example: Callback with lock

$$
\begin{array}{l|l}
\texttt{let}\ \mathtt{b}_1 = \mathrm{ref}\ \mathbf{true},\ \mathtt{c}_1 = \mathrm{ref}\ 0, & \texttt{let}\ \mathtt{b}_2 = \mathrm{ref}\ \mathbf{true},\ \mathtt{c}_2 = \mathrm{ref}\ 0, \\
\quad \texttt{fun}\ \mathtt{inc}_1(\mathtt{f}) = \texttt{if}\ !\mathtt{b}_1\ \texttt{then}\{ & \quad \texttt{fun}\ \mathtt{inc}_2(\mathtt{f}) = \texttt{if}\ !\mathtt{b}_2\ \texttt{then}\ \{ \\
\qquad \mathtt{b}_1 := \mathbf{false}; & \qquad \mathtt{b}_2 := \mathbf{false}; \\
\qquad f(); c_1 := !c_1 + 1; & \qquad \mathit{let}\ n = !c_2\ \mathit{in}\ f(); c_2 := n + 1; \\
\qquad \mathtt{b}_1 := \mathbf{true}\} \ \texttt{else}\ (), & \qquad \mathtt{b}_2 := \mathbf{true}\} \ \texttt{else}\ (), \\
\quad \texttt{fun}\ \mathtt{get}_1() = !\mathtt{c}_1 & \quad \texttt{fun}\ \mathtt{get}_2() = !\mathtt{c}_2 \\
\texttt{in}\ \langle \mathtt{inc}_1, \mathtt{get}_1 \rangle & \texttt{in}\ \langle \mathtt{inc}_2, \mathtt{get}_2 \rangle
\end{array}
$$



Automatically generated temporal formula

$\mathbb{E}[\![\tau]\!](M_1^{clb}, M_2^{cbl})$ is equal to

$$\mathcal{H}\mathtt{N}_0.(\mathcal{N}_1\ell_2.(\mathcal{N}_1\ell_1.(\mathcal{N}_2\ell_4.(\mathcal{N}_2\ell_3.(\mathtt{X}((\ell_2 \mapsto_1 0) \wedge (\ell_1 \mapsto_1 \mathbf{true}) \wedge (\ell_4 \mapsto_2 0) \wedge (\ell_3 \mapsto_2 \mathbf{true}) \wedge$$
$$((\Box(\mathcal{H}\mathtt{N}_5.(\forall \mathtt{x}_6, \mathtt{x}_7, \mathtt{x}_8, \mathtt{x}_9.(((\ell_2 \mapsto_1 \mathtt{x}_6) \wedge (\ell_1 \mapsto_1 \mathtt{x}_7) \wedge (\ell_4 \mapsto_2 \mathtt{x}_8) \wedge (\ell_3 \mapsto_2 \mathtt{x}_9)) \Rightarrow$$
$$((\mathtt{X}(((\mathtt{x}_7 = \mathbf{true}) \wedge (\mathtt{x}_9 = \mathbf{true})) \Rightarrow ((\ell_2 \mapsto_1 \mathtt{x}_6) \wedge (\ell_1 \mapsto_1 \mathbf{false}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_8) \wedge (\ell_3 \mapsto_2 \mathbf{false}) \wedge$$
$$(\Box_{\mathrm{pub}}(\forall \mathtt{x}_{10}, \mathtt{x}_{11}, \mathtt{x}_{13}, \mathtt{x}_{14}.(((\ell_2 \mapsto_1 \mathtt{x}_{10}) \wedge (\ell_1 \mapsto_1 \mathtt{x}_{11}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_{13}) \wedge (\ell_3 \mapsto_2 \mathtt{x}_{14})) \Rightarrow$$
$$(\mathtt{X}(\forall \mathtt{x}_{12}, \mathtt{x}_{15}.(((\mathtt{x}_{12} = \mathtt{x}_{10} + 1) \wedge (\mathtt{x}_{15} = \mathtt{x}_8 + 1)) \Rightarrow ((\ell_2 \mapsto_1 \mathtt{x}_{12}) \wedge (\ell_1 \mapsto_1 \mathbf{true}) \wedge$$
$$(\ell_4 \mapsto_2 \mathtt{x}_{15}) \wedge (\ell_3 \mapsto_2 \mathbf{true}) \wedge (\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_5)))))))))))) \wedge (\mathtt{not}((\mathtt{x}_7 = \mathbf{true}) \wedge (\mathtt{x}_9 = \mathbf{false}))) \wedge$$
$$(\mathtt{not}((\mathtt{x}_7 = \mathbf{false}) \wedge (\mathtt{x}_9 = \mathbf{true}))) \wedge (\mathtt{X}(((\mathtt{x}_7 = \mathbf{false}) \wedge (\mathtt{x}_9 = \mathbf{false})) \Rightarrow$$
$$((\ell_2 \mapsto_1 \mathtt{x}_6) \wedge (\ell_1 \mapsto_1 \mathtt{x}_7) \wedge (\ell_4 \mapsto_2 \mathtt{x}_8) \wedge (\ell_3 \mapsto_2 \mathtt{x}_9) \wedge (\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_5)))))))))))$$
$$\wedge(\Box(\mathcal{H}\mathtt{N}_1 6.(\forall [\mathtt{x}_{17}, \mathtt{x}_{18}, \mathtt{x}_{19}, \mathtt{x}_{20}].(((\ell_2 \mapsto_1 \mathtt{x}_{17}) \wedge (\ell_1 \mapsto_1 \mathtt{x}_{18}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_{19}) \wedge (\ell_3 \mapsto_2 \mathtt{x}_{20})) \Rightarrow$$
$$(\mathtt{X}((\ell_2 \mapsto_1 \mathtt{x}_{17}) \wedge (\ell_1 \mapsto_1 \mathtt{x}_{18}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_{19}) \wedge (\ell_3 \mapsto_2 \mathtt{x}_{20}) \wedge (\mathtt{x}_{17} = \mathtt{x}_{19}) \wedge (\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_1 6)))))))))$$
$$\wedge(\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_0)))))))))$$

Translation to SMT-LIB

```
 (assert (exists ((s21 Int)(h22 Heap)(h23 Heap)(l2 Int)(l1 Int)) (and (not (= l1 l2))
(exists ((l4 Int)(l3 Int)) (and (not (= l3 l4)) (exists ((s25 Int)(h26 Heap)(h27 Heap)(S28 LocSpan))
(and (TransPriv s21 s25 h22 h23 h26 h27 S28) (and (= (select h26 l2) 0) (= (select h26 l1) 0)
(= (select h27 l4) 0) (= (select h27 l3) 0) (and (forall ((s29 Int)(h30 Heap)(h31 Heap)(S32 LocSpan))
(=> (TransPrivT s25 s29 h26 h27 h30 h31 S32) (forall ((x6 Int)(x7 Int)(x8 Int)(x9 Int))
(=> (and (= (select h30 l2) x6) (= (select h30 l1) x7) (= (select h31 l4) x8) (= (select h31 l3) x9) )
 (and (exists ((s33 Int)(h34 Heap)(h35 Heap)(S36 LocSpan)) (and (TransPriv s29 s33 h30 h31 h34 h35 S36)
 (=> (and (= x7 0) (= x9 0) ) (and (= (select h34 l2) x6) (= (select h34 l1) 1) (= (select h35 l4) x8)
(= (select h35 l3) 1) (forall ((s37 Int)(h38 Heap)(h39 Heap)(S40 LocSpan))
 (=> (TransPubT s33 s37 h34 h35 h38 h39 S40) (forall ((x10 Int)(x11 Int)(x13 Int)(x14 Int))
 (=> (and (= (select h38 l2) x10) (= (select h38 l1) x11) (= (select h39 l4) x13) (= (select h39 l3) x14))
 (exists ((s41 Int)(h42 Heap)(h43 Heap)(S44 LocSpan)) (and (TransPriv s37 s41 h38 h39 h42 h43 S44)
 (forall ((x12 Int)(x15 Int)) (=> (and (= x12 (+ x10 1)) (= x15 (+ x8 1)) ) (and (= (select h42 l2) x12)
 (= (select h42 l1) 0) (= (select h43 l4) x15) (= (select h43 l3) 0)
 (TransPub s29 s41 h30 h31 h42 h43 S44))))))))))))))
 (not (and (= x7 0) (= x9 1) )) (not (and (= x7 1) (= x9 0) ))
 (exists ((s45 Int)(h46 Heap)(h47 Heap)(S48 LocSpan)) (and (TransPriv s29 s45 h30 h31 h46 h47 S48)
 (=> (and (= x7 1) (= x9 1) ) (and (= (select h46 l2) x6) (= (select h46 l1) x7) (= (select h47 l4) x8)
 (= (select h47 l3) x9) (TransPub s29 s45 h30 h31 h46 h47 S48)))))))))
 (forall ((s49 Int)(h50 Heap)(h51 Heap)(S52 LocSpan))(=> (TransPrivT s25 s49 h26 h27 h50 h51 S52)
 (forall ((x17 Int)(x18 Int)(x19 Int)(x20 Int)) (=> (and (= (select h50 l2) x17) (= (select h50 l1) x18)
 (= (select h51 l4) x19) (= (select h51 l3) x20) ) (exists ((s53 Int)(h54 Heap)(h55 Heap)(S56 LocSpan))
 (and (TransPriv s49 s53 h50 h51 h54 h55 S56) (and (= (select h54 l2) x17) (= (select h54 l1) x18)
 (= (select h55 l4) x19) (= (select h55 l3) x20) (= x17 x19) (TransPub s49 s53 h50 h51 h54 h55 S56)))))))))
 (TransPub s21 s25 h22 h23 h26 h27 S28)))))))))

 (check-sat)
```

Going further: SyTeCi

- Combine Symbolic, Temporal with Circular reasoning
  - $\rightsquigarrow$ Deal with recursive functions,
  - $\rightsquigarrow$ Synchronization of recursive calls,
  - $\rightsquigarrow$ Simple heuristics to unfold recursive functions,
  - $\rightsquigarrow$ Circular proof system.
- Computation of the transitive closure of the WTS $\mathcal{A}$
  - $\rightsquigarrow$ Undecidable in general,
  - $\rightsquigarrow$ Over-approximations via abstractions.
- Automatic Computation of $\mathcal{A}$.