Temporal Verification of Higher-Order Functional Programs

Akihiro Murase[1], **Tachio Terauchi[2]**
Naoki Kobayashi[3], Ryosuke Sato[3], Hiroshi Unno[4]

[1] Nagoya U. / Toshiba
[2] JAIST
[3] U. Tokyo
[4] U. Tsukuba

Appeared in POPL 2016

Functional program

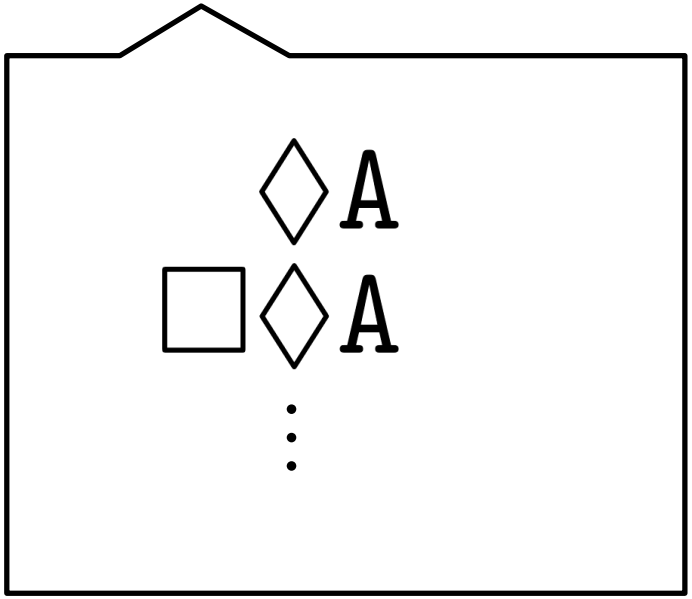
Temporal property

$$P \subseteq \Psi$$

Higher-order

```
letrec repeat g =
   g (nondet ());repeat g
and f x =
   if x > 0 then f (x-1)
   else event A
in
 repeat f
```

Events



Automata-theoretic approach[Vardi'91]

- Input:
  - Program P
  - ω-regular temporal property Ψ

1. Make ω-automaton $\mathcal{A}_{\neg\Psi}$ recognizing $\neg\Psi$
2. Make product program $P \times \mathcal{A}_{\neg\Psi}$
3. Verify that $P \times \mathcal{A}_{\neg\Psi}$ terminates

Theorem: $P \subseteq \Psi$ iff $P \times \mathcal{A}_{\neg\Psi}$ terminates

Verifying $P \times A_{\neg\Psi}$ terminates

Instance of **"fair termination"** problem

Problem Definition:

- Fairness Constraint: $C = \{(A_0, B_0), (A_1, B_1), ..., (A_n, B_n)\}$
- Infinite sequence π is **fair** wrt C if for each $(A_i, B_i) \in C$
  - A_i occurs only finitely often in π; or
  - B_i occurs infinitely often in π

- P is **fair terminating** wrt C if P has no infinite execution trace that is fair wrt C

This Paper's Contribution:

Sound & complete verification method for fair termination of FP

Fair termination for imperative programs

Def: Finite sequence ϖ is **fair** wrt C if for each $(A, B) \in C$

- A does not occur in ϖ; or
- B occurs in ϖ

- For binary relation R, let

$$R^{+\restriction C} = \left\{ (\varpi_0, \varpi_n) \;\middle|\; \begin{array}{c} \forall 0 \leq i < j \leq n.(\varpi_i, \varpi_j) \in R \wedge \\ \varpi_0 \varpi_1 \ldots \varpi_n \text{ is fair wrt } C \end{array} \right\}$$

Intuition: $R^{+\restriction C}$ is the subset of R^+ that is fair wrt C

Theorem[Cook+'07]: P is fair terminating wrt C iff ${Trans_P}^{+\restriction C}$ is disjunctively well-founded

- $Trans_P$ = transition relation of P
- Disjunctively well-founded = finite union of well-founded relations

Why not just apply this to FP?

Just use [Cook+'07] and check $Trans_P{}^{+\restriction C}$ is dwf to verify fair termination of FPs?

- After all, any program (functional, imperative, concurrent or whatever) can be considered a transition system…

Unfortunately, this turns out to be an awful idea

$$
\begin{array}{l}
\texttt{let rec ack}\ m\ n\ = \\
\quad \texttt{if}\ m = 0\ \texttt{then}\ n+1 \\
\quad \texttt{else if}\ n = 0\ \texttt{then ack}\ (m-1)\ 1 \\
\quad \texttt{else ack}\ (m-1)\ (\texttt{ack}\ m\ (n-1)) \\
\texttt{let main}\ ()\ = \texttt{ack}\ *_{\texttt{pos}}\ *_{\texttt{pos}}
\end{array}
$$

Terminates, but transition relation is quite complex:

$$
\begin{array}{rl}
\mathsf{ack}\ m\ n & \to \mathsf{ack}\ (m-1)\ (\mathsf{ack}\ m\ (n-1)) \\
& \to \mathsf{ack}\ (m-1)\ (\mathsf{ack}\ (m-1)\ (\mathsf{ack}\ m\ (n-2))) \\
& \vdots \\
& \to \mathsf{ack}\ (m-1)\ (\mathsf{ack}\ (m-1)\ (\ldots, (\mathsf{ack}\ m\ 0)\ldots)) \\
& \to \mathsf{ack}\ (m-1)\ (\mathsf{ack}\ (m-1)\ (\ldots, (\mathsf{ack}\ (m-1)\ 1)\ldots)) \\
& \vdots
\end{array}
$$

[Cook+'07] needs to reason about change in calling context / call stack

Theorem[Berardi+'14,Yokoyama'14]: [Cook+'07] can only prove termination of primitive recursive functions (when usable wf relations have height at most ω)

NOTE1: This is just 1st-order function! Things get worse with higher-order
NOTE2: This is just plain termination! Things get worse with fair termination

Our Approach

Check dwf of (transitive closure of fair part of) "calling relation"

- Formally,

  $Call_P = \{(f\ \vec{v}, g\ \vec{w}) \mid g\ \vec{w} \text{ is called from } f\ \vec{v} \text{ in an execution of } P\}$

  Note: $Call_P{}^+ = \{(f\ \vec{v}, g\ \vec{w}) \mid E[f\ \vec{v}] \in \mathcal{R}_P \wedge f\ \vec{v} \to_P{}^+ E'[g\ \vec{w}]\}$

  where $\mathcal{R}_P$ is set of reachable states and $\to_P$ is 1-step reduction
- **Theorem[Kuwahara+'14]:** P (plain) terminates iff $Call_P{}^+$ is dwf

  Q: P fair-terminates wrt C iff $Call_P{}^{+\restriction C}$ is dwf?

  A: Unfortunately, No (cf. paper for counterexample)
- **ALTERNATIVE** $\rhd_P^C = \{(f\ \vec{v}, g\ \vec{w}) \mid E[f\ \vec{v}] \in \mathcal{R}_P \wedge f\ \vec{v} \to_P{}^{+\restriction C} E'[g\ \vec{w}]\}$

  Note: $\rhd_P^C$ is $Call_P{}^+$ but with $\to_P{}^+$ replaced by $\to_P{}^{+\restriction C}$
- **Theorem[this paper]:** P fair-terminates wrt C iff $\rhd_P^C$ is dwf

Checking $\triangleright_P^C$ is dwf

Algorithm:

1. Initialize **candidate** disjunctively well-founded rel. D
2. Build program $\lceil P \rceil_{D,C}$ that is assertion safe iff $\triangleright_P^C \subseteq D$
3. Check assertion safe of $\lceil P \rceil_{D,C}$ **KEY STEP**
   - Use reachability checker for FP [Terauchi'10, Kobayashi+'11, etc.]

   a. **Safe** -> done. Output "P fair terminates wrt C"

   b. **Unsafe** ->
   - Get **c.ex. trace** ϖ of $\lceil P \rceil_{D,C}$ s.t. $[\![\varpi]\!] \subseteq \triangleright_P^C \not\subseteq D$
   - Infer dwf $D' \supseteq [\![\varpi]\!]$ via rank function inference [Podelski+'04, etc.]
   - Repeat from 2. with $D \leftarrow D \cup D'$

This verification style is called **Binary Reachability Analysis**
- The style itself is not new. E.g., [Cook+'07] also uses it

Rest of the Talk

1. Transforming P to $\lceil P \rceil_{D,C}$
   a. Transformation for plain termination
   b. Extending to fair termination
2. Experiments

Transformation for plain termination

- Recall

Theorem[Kuwahara+'14]: P terminates iff $Call_P{}^+$ is dwf

$$Call_P{}^+ = \{(f\ \vec{v}, g\ \vec{w}) \mid E[f\ \vec{v}] \in \mathcal{R}_P \wedge f\ \vec{v} \rightarrow_P{}^+ E'[g\ \vec{w}]\}$$

GOAL: Build $\lceil P \rceil_D$ that is assertion safe iff $Call_P{}^+ \subseteq D$

Observation:

$Call_P{}^+$ is dwf iff $\{(\vec{v}, \vec{w}) \mid (f\ \vec{v}, f\ \vec{w}) \in Call_P{}^+\}$ is dwf for each f

So, we will actually build $\lceil P \rceil_{D,f}$ that is assertion safe iff $\{(\vec{v}, \vec{w}) \mid (f\ \vec{v}, f\ \vec{w}) \in Call_P{}^+\} \subseteq D$ for each f

Informal Overview

Key Observation:

$(f\ \vec{v}, f\ \vec{w}) \in Call_P{}^+$ **iff** $f\ \vec{w}$ **is called after** $f\ \vec{v}$ **is called but before** $f\ \vec{v}$ **returns**

- so, when f is called, record the arguments, and pass recorded arguments to calls that occur in $f's$ body
  - also make all other functions take and pass the recorded arguments down
- when f is called again, assert $(\text{recorded}, \text{current}) \in D$
- non-deterministically decide when to record so as to compare with all possible descendants' arguments

Transformation Example

- Let $D = \{(x', x) \mid x \geq 0 \wedge x' > x\}$
- Check $\{(x', x) \mid (\mathtt{f}\ x', \mathtt{f}\ x) \in {Call_P}^+\} \subseteq D$

```
f x =
  if x ≤ 0 then 0
       else g (x-1)
g x = f x
main () = g ∗
```

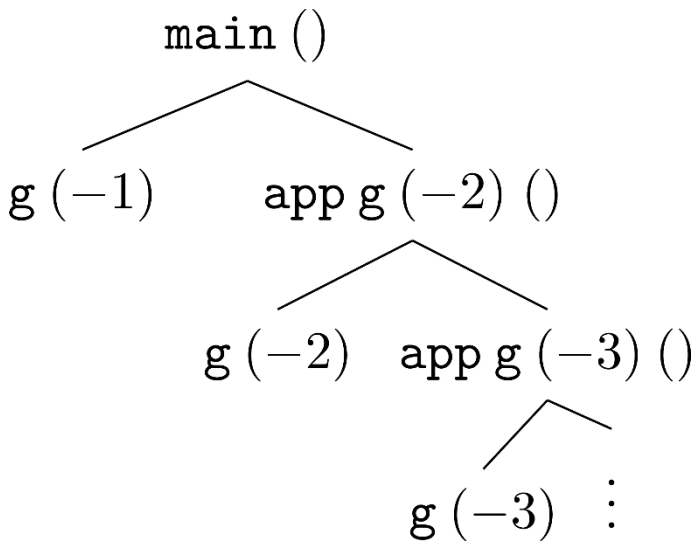
```
f x' x =
   assert D⊥(x',x);
   let x' = ∗?x':x in
    if x ≤ 0 then 0
        else g x' (x-1)
g x' x = f x' x
main () = g ⊥ ∗
```

- $\ast$ = non-deterministic choice
- $D_\perp = D \cup \{(\perp, x) \mid x \in \mathbb{Z}\}$

Higher-order is trickier

```
app f x u = f x u
id u = u
g x = if x = 0 then id
      else app g (x-1)
main () = g * ()
```

Q: Does this terminate?



Transformation Example (higher-order)

- Let $D = \{((f', x', u'), (f, x, u)) \mid x \geq 0 \wedge x' > x\}$
- Check

$$\{((f', x', u'), (f, x, u)) \mid (\texttt{app}\ f'\ x'\ u', \texttt{app}\ f\ x\ u) \in Call_P{}^{+}\} \subseteq D$$

```
app f x u = f x u
id u = u
g x = if x ≤ 0 then id
      else app g (x-1)
main () = g ∗ ()
```

```
app _ f _ x (f',x',u') u =
   assert  D⊥((f',x',u'),(f,x,u));
   let f',x',u' = ∗?(f',x',u'):(f,x,u) in
    f (f',x',u') x (f',x',u') u
id _ u = u
g (f',x',u') x = if x ≤0 then id
      else app (f',x',u') g (f',x',u') (x-1)
main () = g ⊥ ∗⊥ ()
```

Remark

We pass recorded arguments everywhere

- i.e., at every (possibly partial) function applications

- Because it is impossible to statically decide
  - which calls are to the target function
  - which calls are total

  Delegate these tasks to backend reachability checker!

- Previous work make conservative approximations

Transformation, formally (1/2)

$$
\begin{aligned}
P &::= \emptyset \mid P \cup \{\mathtt{f}\ \vec{x} = e\} \\
e &::= \mathtt{f} \mid x \mid c \mid e_1\ op\ e_2 \mid e_1\ e_2 \\
&\mid\ \ \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \mid \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \\
\\
c &::= * \mid \mathtt{true} \mid \mathtt{false} \mid 0 \mid 1 \mid ... \\
op &::= + \mid - \mid \times \mid \leq \mid < \mid \wedge \mid ...
\end{aligned}
$$

λ-lifted functional language (with non-determinism)

Transformation, formally (2/2)

$$\lceil P \rceil_{D,\mathtt{f}} = \{ \lceil \mathtt{g}\ \vec{x} = e \rceil_{D,\mathtt{f}} \mid \mathtt{g}\ \vec{x} = e \in P \}$$

$$\lceil \mathtt{g}\ \vec{x} = e \rceil_{D,\mathtt{f}} = \begin{cases} \mathtt{g}\ s_1\ x_1\ s_2\ x_2\ \ldots\ s_n\ x_n = & \text{if } \mathtt{g} \neq \mathtt{f} \\ \quad \lceil e \rceil_{s_n} & \\ \mathtt{g}\ s_1\ x_1\ s_2\ x_2\ \ldots\ s_n\ x_n = & \text{if } \mathtt{g} = \mathtt{f} \\ \quad \mathtt{assert}\ D_\perp(s_n, (\vec{x})); & \\ \quad \mathtt{let}\ s = *?s_n : (\vec{x})\ \mathtt{in}\ \lceil e \rceil_s & \end{cases}$$

where $\vec{x} = x_1, x_2, \ldots, x_n$

$$\lceil c \rceil_s = c \qquad \lceil \mathtt{f} \rceil_s = \mathtt{f} \qquad \lceil x \rceil_s = x$$

$$\lceil \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \rceil_s = \mathtt{let}\ x = \lceil e_1 \rceil_s\ \mathtt{in}\ \lceil e_2 \rceil_s$$

$$\lceil e_1\ op\ e_2 \rceil_s = \lceil e_1 \rceil_s\ op\ \lceil e_2 \rceil_s$$

$$\lceil \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 \rceil_s = \mathtt{if}\ \lceil e_1 \rceil_s \mathtt{then}\ \lceil e_2 \rceil_s \mathtt{else}\ \lceil e_3 \rceil_s$$

$$\lceil e_1\ e_2 \rceil_s = \lceil e_1 \rceil_s\ s\ \lceil e_2 \rceil_s$$

Rest of the Talk

1. Transforming P to $\lceil P \rceil_{D,C}$
   - ✓ Transformation for plain termination
   - b. Extending to fair termination
2. Experiments

Review

GOAL: Check $\{(\vec{v}, \vec{w}) \mid (f\ \vec{v}, f\ \vec{w}) \in \triangleright_P^C\} \subseteq D$

- $\triangleright_P^C = \{(f\ \vec{v}, g\ \vec{w}) \mid E[f\ \vec{v}] \in \mathcal{R}_P \wedge f\ \vec{v} \rightarrow_P{}^{+\restriction C}\ E'[g\ \vec{w}]\}$
- $Call_P{}^+ = \{(f\ \vec{v}, g\ \vec{w}) \mid E[f\ \vec{v}] \in \mathcal{R}_P \wedge f\ \vec{v} \rightarrow_P{}^+\ E'[g\ \vec{w}]\}$

Just showed how to check $\{(\vec{v}, \vec{w}) \mid (f\ \vec{v}, f\ \vec{w}) \in Call_P{}^+\} \subseteq D$

- Transformation Informal Overview:
  - As before, we record arguments of "ancestor" calls and compare with current arguments

But, only do this for fair traces (i.e., ones that follow $\rightarrow_P{}^{+\restriction C}$)

Transformation Example

- Let $C = \{(A, \texttt{false})\}$
  - P fair terminates wrt C iff A occurs infinitely often in any non-terminating execution

$$
\begin{array}{l}
\texttt{repeat}\ g = \texttt{let}\ x = *\ \texttt{in}\ g\ x; \texttt{repeat}\ g \\
f\ x = \texttt{if}\ x > 0\ \texttt{then}\ f(x-1)\ \texttt{else event}\ A \\
\texttt{main}\ () = \texttt{repeat}\ f
\end{array}
$$

$\lceil P \rceil_{D,C,f}$

$$
\begin{array}{l}
\texttt{repeat}\ s\ x'\ g = \texttt{let}\ x = *\ \texttt{in} \\
\qquad\qquad\qquad\qquad \texttt{let}\ (s', \_) = g\ s\ x'\ x\ \texttt{in}\ \texttt{repeat}\ s'\ x'\ g \\
f\ s\ x'\ x = \texttt{assert}(s = \texttt{false} \Rightarrow D_\perp(x', x)); \\
\qquad\qquad \texttt{let}\ (s, x') = \\
\qquad\qquad\qquad \texttt{if}\ *\ \texttt{then}\ (\texttt{false}, x)\ \texttt{else}\ (s, x')\ \texttt{in} \\
\qquad\qquad \texttt{if}\ x > 0\ \texttt{then}\ f\ s\ x'\ (x-1)\ \texttt{else}\ (\texttt{true}, ()) \\
\texttt{main}\ () = \texttt{repeat false}\ \perp\ f
\end{array}
$$

Rest of the Talk

- ✓ Transforming P to $\lceil P \rceil_{D,C}$
  - ✓ Transformation for plain termination
  - ✓ Extending to fair termination
- 2. Experiments

Prototype Implementation

- Fair termination verifier for OCaml programs
  - MOCHI [1] as backend reachability checker
  - Z3 [2] for constraint solving in rank func. inference

Prototype implementation web interface [3]

[1] http://www.kb.is.s.u-tokyo.ac.jp/~ryosuke/mochi/

[2] http://z3.codeplex.com/

[3] http://www-kb.is.s.u-tokyo.ac.jp/~ryosuke/fair_termination/

Experiment Result

program	cycle1	cycle2	time
intro	3	14	11.492
repeat	4	12	2.276
closure	6	18	9.76
hofmann-1 [1]	2	4	0.232
hofmann-2 [1]	3	8	1.032
koskinen-1 [2]	7	27	43.344
koskinen-2 [2]	5	16	3.412
koskinen-3-1 [2]	6	17	2.752
koskinen-3-2 [2]	4	14	2.216
koskinen-3-3 [2]	6	23	4.964
koskinen-4 [2]	10	35	132.552
lester [3]	8	36	38.356

[1] Hoffmann, Chen LICS'14
[2] Koskinen, Terauchi LICS'14
[3] Lester et al. '11

None of these can be verified automatically by previous methods

Conclusion

Automatic method for temporal property verification of functional programs

- Based on Binary Reachability Analysis approach
- Supports
  - Higher-order functions
  - Arbitrary omega-regular properties
- Sound & complete
  - relative to soundness & completeness of backend reachability checker and rank func. inference