Louis Mandel

Université Paris-Sud 11, LRI

Reactivity of ReactiveML programs

A new static analysis for ReactiveML

SYNCHRON 2007 – 28/11/2007

ReactiveML

a language for the programming of interactive systems

- conservative extension of Ocaml
- based on the synchronous reactive model of F. Boussinot
- no real time constraints
  - dynamic creation of processes through recursion

Network simulator demo

ReactiveML

$$
\begin{array}{lcl}
e & ::= & x \mid c \mid (e, e) \mid \lambda x.e \mid e\ e \mid \texttt{rec}\ x = e \mid \texttt{process}\ e \\
 & & \mid e\texttt{;}e \mid \texttt{let}\ x\ \texttt{=}\ e\ \texttt{and}\ x\ \texttt{=}\ e\ \texttt{in}\ e \mid \texttt{pause} \mid \texttt{run}\ e \\
 & & \mid \texttt{signal}\ x\ \texttt{in}\ e \mid \texttt{present}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{emit}\ e \\
 & & \mid \texttt{do}\ e\ \texttt{until}\ e\ \texttt{done} \mid \texttt{do}\ e\ \texttt{when}\ e \\
c & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{()} \mid \texttt{0} \mid \ldots \mid \texttt{+} \mid \texttt{-} \mid \ldots
\end{array}
$$

- Derived operators

$$
\begin{array}{lcl}
e_1 \,||\, e_2 & \stackrel{def}{=} & \texttt{let}\ x_1\ \texttt{=}\ e_1\ \texttt{and}\ x_2\ \texttt{=}\ e_2\ \texttt{in}\ \texttt{()} \\
\texttt{let process}\ f\ x\ \texttt{=}\ e & \stackrel{def}{=} & \texttt{let}\ f\ \texttt{=}\ \lambda x.(\texttt{process}\ e) \\
\texttt{let rec process}\ p\ x\ \texttt{=}\ e & \stackrel{def}{=} & \texttt{rec}\ p\ \texttt{=}\ \lambda x.(\texttt{process}\ e) \\
 & \ldots &
\end{array}
$$

Why a new static analysis?

```
From: Xxxxxx Xxxxx
To: Louis Mandel
Subject: problème rml

Hello,

Je me suis mis un peu au rml et j'essaie d'écrire mon
premier programme [...]
Mais quand je lance le process main, il ne se passe rien.
Il semblerait que print_clock ne reçoive jamais le signal s
émis par clock...

Saurais-tu quel est le problème?

Cordialement,
-- Xxxxxx Xxxxx
```

Why a new static analysis?

```
let process clock timer s =
  let cpt = ref timer in
  loop
    if !cpt = 0 then (cpt := timer; emit s)
    else cpt := !cpt - 1
  end


let process print_clock s =
  loop await s; print_string "top␣\n" end

let process main =
  signal s in run (clock 10 s) || run (print_clock s)
```

Why a new static analysis?

```
let process clock timer s =
  let cpt = ref timer in
  loop
    if !cpt = 0 then (cpt := timer; emit s)
    else cpt := !cpt - 1;
    pause
  end

let process print_clock s =
  loop await s; print_string "top␣\n" end

let process main =
  signal s in run (clock 10 s) || run (print_clock s)
```

Goal

Define a simple static analysis to find *most* of the instantaneous loops and without *to much* wrong warnings.

Remark:

Until ReactiveML version 1.05 there were only dynamic test of instantaneous loops.

```
let process main =
  signal s in run (clock 10 s) || run (print_clock s)
  ||
  loop print_string "**************\n"; pause end
```

Examples

The two simplest examples of non reactive programs:

```
let process p =
  loop () end
```

Line 2, characters 2-13:

Warning: This expression may be an instantaneous loop.

```
let rec process q =
  run q
```

Line 2, characters 2-7:

Warning: This expression may produce an instantaneous recursion.

Outline

1. **Instantaneity analysis**
2. **Instantaneous recursion detection**

Instantaneity analysis

```
(* e1 *) ackerman 3 10
```

e1 is instantaneous.

```
(* e2 *) print_int 1; pause; print_int 2
```

e2 is not instantaneous.

```
(* e3 *) if x then pause else ()
```

e3 may be instantaneous or not.

Instantaneity typing

Type judgement:

$$e : k$$

where k ::= $-$ *instantaneous*
$\mid \pm$ *don't know*
$\mid +$ *non instantaneous*

The order $<$ over k is: $- < \pm < +$

$max(k_1, k_2)$ is define by:

	$-$	$\pm$	$+$
$-$	$-$	$\pm$	$+$
$\pm$	$\pm$	$\pm$	$+$
$+$	$+$	$+$	$+$

$$c : - \qquad \frac{e : -}{\lambda x.e : -} \qquad \frac{e_1 : - \quad e_2 : -}{e_1\ e_2 : -}$$

$$\frac{e : -}{\texttt{rec}\ x \texttt{=} e : -} \qquad \frac{e : k}{\texttt{process}\ e : -} \qquad \frac{e_1 : k_1 \quad e_2 : k_2}{e_1 \texttt{;} e_2 : max(k_1, k_2)}$$

$$\frac{e : - \qquad e_1 : k_1 \qquad e_2 : k_2}{\texttt{present}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : max(\pm, k_1)}$$

$$\texttt{pause} : + \qquad \frac{e : -}{\texttt{run}\ e : \pm}$$

Properties

Property 1
If $e : -$ *and* $e/S \rightarrow^* e'/S'$ *and* e' *is an end of instant expression then* e' *is a value.*

Property 2
If $e : +$ *and* $e/S \rightarrow^* e'/S'$ *and* e' *is an end of instant expression then* e' *is not a value.*

Proof:

Done in the Coq proof assistant with Zaynah Dargaye

Limitations

```
let process dynamic s = loop present s then () else () end
```

Instantaneity depends on dynamic properties.

```
let process static =
  loop
    signal s in present s then () else ()
  end
```

Signal `s` cannot be emitted (⇒ potential analysis of Esterel).

```
let process p = pause
let process q = loop run p end
```

The type of the body of a process is *erased* by the typing.

Typing and instantaneity analysis

Add instantaneity information to the type of processes: $\tau\,\texttt{process}^k$

Typing judgement: $H \vdash e : \tau[k]$

Examples of rules:

$$\frac{H \vdash e : \tau[k]}{H \vdash \texttt{process}\ e : (\tau\,\texttt{process}^k)[-]} \qquad \frac{H \vdash e : (\tau\,\texttt{process}^k)[-]}{H \vdash \texttt{run}\ e : \tau[k]}$$

$$\frac{H \vdash e : (\tau\,\texttt{process}^k)[k']}{H \vdash e : (\tau\,\texttt{process}^{\pm})[k']}$$

2. Instantaneous recursion detection

Instantaneous recursion detection

The `loop` case:

```
loop e end
```

If e does not have type $+$ then a warning must be emitted.

The recursion case:

```
let rec process p = pause; run p
```

Any recursive call can be done during the first instant of a process.

Examples

We must take aliases into account.

```
let rec process p =
  let q = p in
  run q
```

We must take care of function calls.

```
let rec process p =
  let q = (fun x -> x) p in
  run q
```

Examples

A recursive call under an abstraction is not dangerous.

```
let rec process p =
  let process q = run p in
  ()
```

The type system

Type judgement:

$$\Pi \vdash e : \pi$$

where $\Pi, \pi \quad ::= \quad \emptyset$

$\quad\quad\quad\quad\quad\quad | \; x : n, \Pi$ with $n \in \mathbb{N} \cup +\infty$

- Π represents the dangerous variables
- π represents the variables used in e

$\Rightarrow$ structure of types is not kept

Some rules

$$\frac{x \notin Dom(\Pi)}{\Pi \vdash x : \emptyset} \qquad \frac{n = \Pi(x) \quad n > 0}{\Pi \vdash x : \{x : n\}} \qquad \frac{x : 0, \Pi \vdash e : \pi}{\Pi \vdash \texttt{rec}\, x \texttt{=} e : \pi \backslash x}$$

$$\frac{\Pi^{\uparrow} \vdash e : \pi}{\Pi \vdash \texttt{process}\ e : \pi} \qquad \frac{\Pi \vdash e : \pi \quad \pi^{\downarrow} > 0}{\Pi \vdash \texttt{run}\ e : \pi^{\downarrow}}$$

$$\frac{\Pi^{\uparrow} \vdash e : \pi}{\Pi \vdash \lambda x.e : \pi} \qquad \frac{\Pi \vdash e_1 : \pi_1 \quad \Pi \vdash e_2 : \emptyset \quad \pi_1^{\downarrow} > 0}{\Pi \vdash e_1\, e_2 : \pi_1^{\downarrow}}$$

Some rules

$$\frac{\Pi \vdash e_1 : \pi_1 \qquad e_1 : + \qquad \emptyset \vdash e_2 : \pi_2}{\Pi \vdash e_1 ; e_2 : \pi_1}$$

$$\frac{\Pi \vdash e_1 : \pi_1 \qquad not(e_1 : +) \qquad \Pi \vdash e_2 : \pi_2}{\Pi \vdash e_1 ; e_2 : \pi_1 * \pi_2}$$

$$\pi = \pi_1 * \pi_2 \text{ iff } \forall x.\pi(x) = min(\pi_1(x), \pi_2(x))$$

Some rules

$$\frac{\Pi \vdash e_1 : \pi_1 \qquad e_1 : + \qquad \emptyset \vdash e_2 : \pi_2}{\Pi \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \pi_1}$$

$$\frac{\Pi \vdash e_1 : \pi_1 \qquad not(e_1 : +) \qquad x : min(\pi_1), \Pi \vdash e_2 : \pi_2}{\Pi \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : (\pi_1 * \pi_2) \backslash x}$$

Property

Property 3
If $\Pi \vdash e : \pi$ *then* $e/S \rightarrow^* e'/S'$ *terminates.*

Counterexample (Landin):

```
let f =
  let r = ref (process ()) in
  let process g = run !r in
  r := g;
  g
```

Property

with Florence Plateau

Property 4
If $\Pi \vdash e : \pi$ *and* $e/S \rightarrow^* e'/S'$ *terminates then* $e' \neq \texttt{err}$.

We introduce an error if a `rec` is executed two times during an instant.

$$\texttt{rec}\, x = e \,/\, S \;\rightarrow_\varepsilon\; e[x \leftarrow \texttt{rec'}\, x = e] \,/\, S$$

$$\texttt{rec'}\, x = e \,/\, S \;\rightarrow_\varepsilon\; \texttt{err} \,/\, S$$

$$\frac{e \,/\, S \;\rightarrow_\varepsilon\; \texttt{err} \,/\, S}{\Gamma(e)/S \rightarrow \texttt{err}/S}$$

Proof (TODO)

Classical type safety property

Lemma 1
If $\Pi \vdash e : \pi$ *and* $e/S \rightarrow e'/S'$ *then* $\Pi' \vdash e' : \pi'$

Lemma 2
If $\Pi \vdash e : \pi$ *and* $e/S \not\rightarrow$ *then* $e \neq \texttt{err}$

Main difficulty: substitution lemma.

Related works

Frédéric Dabrowski, Frédéric Boussinot and Roberto Amadio :

- complexity of reactive programs

Gérard Boudol :

- reactivity in a language with reference

Conclusion

- Separation between the instantaneity analysis and instantaneous recursion detection.
- The analysis is implemented in ReactiveML:

  `http://reactiveml.org`

Perspectives

- Terminate the proofs
- Extend the instantaneity analysis
- Extend the reactivity analysis