ACI Alidecs

Reactive ML

Louis Mandel Marc Pouzet

Laboratoire d'Informatique de Paris 6

22 octobre 2004

Plan

- Presentation du langage
- Exemples
- Sémantique comportementale et analyses statiques
- Conclusion

Reactive ML

Le noyau

$$
\begin{array}{rcl}
e & ::= & x \mid c \mid (e, e) \mid \lambda x.e \mid e\ e \mid \texttt{rec}\ x\ \texttt{=}\ e \mid \texttt{proc}\ e \\
& & \mid e\texttt{;}e \mid e \texttt{||} e \mid \texttt{loop}\ e \mid \texttt{present}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
& & \mid \texttt{signal}\ x\ \texttt{default}\ e\ \texttt{gather}\ e\ \texttt{in}\ e \mid \texttt{emit}\ e\ e \\
& & \mid \texttt{let}\ x\ \texttt{=}\ e\ \texttt{in}\ e \mid \texttt{let}\ e(x)\ \texttt{in}\ e \mid \texttt{run}\ e \\
& & \mid \texttt{do}\ e\ \texttt{until}\ e \mid \texttt{do}\ e\ \texttt{when}\ e
\end{array}
$$

$$
c \quad ::= \quad \texttt{true} \mid \texttt{false} \mid \texttt{()} \mid \texttt{0} \mid \ldots \mid \texttt{+} \mid \texttt{-} \mid \ldots
$$

Reactive ML

Opérateurs dérivés

`emit` s	$\stackrel{def}{=}$	`emit` s `()`
`present` s `then` p	$\stackrel{def}{=}$	`present` s `then` p `else ()`
`present` s `else` p	$\stackrel{def}{=}$	`present` s `then ()` `else` p
`await immediate` s	$\stackrel{def}{=}$	`do ()` `when` s
`pause`	$\stackrel{def}{=}$	`signal` s `in present` s `else ()`
...		

`signal` s `in` $p \stackrel{def}{=}$ `signal` s `default [] gather fun x y -> x::y in` p

Objectifs

- Avoir un langage de programmation
  - Integration de l'approche réactive dans ML
  - Sémantique de la partie réactive et combinatoire
  - Signaux valués avec une notion de portée
  - Analyses statiques
  - Scheduling efficace

Exemple

```
let moyenne_list l =
  let sum, nb =
    List.fold_left (fun (sum,nb) n -> (sum+n, nb+1)) (0,0) l
  in
  sum / nb

let process moyenne_sig s =
  loop
    await s (l) in
    print_int (moyenne_list l)
  end
```

Exemple d'ordre supérieur

```
let rec process dynapar add =
  await add (p) in
  run p
  ||
  run (dynapar add)
```

Exemple d'ordre supérieur

```
let process main =
  signal add default (process ())
              gather (fun x y -> process (run x || run y)) in
  run (dynapar add)
  ||
  signal s in
  do
    ...
    let process p = ... in emit add p;
    ...
  until s done
```

Scope Extrusion

```
let process send add p1 p2 =
  signal ack in
  emit add (process (run p1; emit ack));
  await immediate ack;
  run p2
```

Sémantique Comportementale

à la ESTEREL

- Sémantique de la partie combinatoire

$$e \Downarrow v$$

- Sémantique de la partie reactive

  La sémantique de e est donnée par le plus petit S tel que $E \sqsubseteq S$ et

$$N \vdash e \xrightarrow[S]{E, b} e'$$

Sémantique Comportementale

$$\frac{e \Downarrow n \quad n \in S \quad N \vdash e_1 \xrightarrow[S]{E,\, b} e_1'}{N \vdash \texttt{present}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \xrightarrow[S]{E,\, b} e_1'}$$

$$\frac{e \Downarrow n \quad n \notin S}{N \vdash \texttt{present}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \xrightarrow[S]{\emptyset,\, false} e_2}$$

$$\frac{e_1 \Downarrow n \quad e_2 \Downarrow v}{N \vdash \texttt{emit}\ e_1\ e_2 \xrightarrow[S]{[\{v\}/n],\, true} \texttt{()}}$$

Séparation combinatoire/réactif

$$\vdash e : k$$

Les règles :

$$\frac{\vdash e : 0}{\vdash \lambda x.e : k} \qquad \frac{\vdash e : k_1}{\vdash \texttt{proc}\ e : k_2}$$

$$\frac{\vdash e_1 : 0 \quad \vdash e_2 : 0}{\vdash \texttt{emit}\ e_1\ e_2 : 1} \qquad \frac{\vdash e_1 : 0 \quad \vdash e_2 : k}{\vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : k}$$

$\Rightarrow$ Les programmes OCaml sont executés sans surcout.

Typage

$$\frac{H \vdash e_1 : (\tau_1, \tau_2)\,\texttt{event} \quad H \vdash e_2 : \tau_1}{H \vdash \texttt{emit}\ e_1\ e_2 : \texttt{unit}}$$

$$\frac{H \vdash e_1 : (\tau_1, \tau_2)\,\texttt{event} \quad H[x : \tau_2] \vdash e : \tau}{H \vdash \texttt{let}\ e_1(x)\ \texttt{in}\ e : \texttt{unit}}$$

$$\frac{H \vdash e_1 : \tau_2 \quad H \vdash e_2 : \tau_1 \to \tau_2 \to \tau_2 \quad H[s : (\tau_1, \tau_2)\,\texttt{event}] \vdash e : \tau}{H \vdash \texttt{signal}\ s\ \texttt{default}\ e_1\ \texttt{gather}\ e_2\ \texttt{in}\ e : \texttt{unit}}$$

Conclusion

- Langage dédié à la programmation de systèmes interactifs :
  Spirale, Spirale++
- Scheduling efficace :
  Fredkin

% of active cells	0 %	4 %	42 %
OCAML	0.74 s	0.75 s	0.76 s
REACTIVEML	0.05 s	0.08 s	0.89 s

- Integration avec efficace avec OCAML :
  Simulateur
- Disponible: `http://www-spi.lip6.fr/~mandel/rml`

Perspectives

- Identification des programmes temps réels
- Complilation