# Création Dynamique et Langages Synchrones

## Travail en Cours sur l'Analyse de Réactivité

Louis Mandel  Florence Plateau  Marc Pouzet

Laboratoire de Recherche en Informatique
Université Paris-Sud 11
INRIA Saclay – Ile-de-France

Réunion Partout – 28/05/2009

# Réseau de Kahn : crible d'Ératosthène

```
Process INTEGERS out QO;
    Vars N; 1 → N;
    repeat INCREMENT N; PUT(N,QO) forever
Endprocess;

Process FILTER PRIME in QI out QO;
    Vars N;
    repeat GET(QI) → N;
           if (N MOD PRIME)≠ 0 then PUT(N,QO) close
    forever
Endprocess;

Process SIFT in QI out QO;
    Vars PRIME; GET(QI) → PRIME;
    PUT (PRIME,QO); comment emit a discovered prime;
    doco channels Q;
     FILTER(PRIME,QI,Q): SIFT(Q,QO)
    closeco
Endprocess;

Process OUTPUT in QI; Comment this is a library process;
    repeat PRINT(GET(QI)) forever
Endprocess;

Start doco channels QI Q2;
    INTEGERS(QI); SIFT(QI,Q2); OUTPUT(Q2);
    closeco;
```

Fig.3. Sieve of Eratosthenes.

# Réseau de Kahn : crible d'Ératosthène (sieve.kahn)

```
Process FILTER PRIME in QI out QO;
  repeat
    GET(QI) -> N;
    if (N MOD PRIME) <> 0 then PUT(N, QO) close
  forever
Endprocess;

Process SIFT in QI out QO;
  Vars PRIME;
  GET(QI) -> PRIME;
  PUT (PRIME, QO); comment emit a discovered prime;
  doco channels Q;
    FILTER(PRIME, QI, Q);
    SIFT(Q, QO)
  closeco
Endprocess;
```

# Crible d'Ératosthène en ReactiveML (sieve.rml)

```
let process filter prime s_in s_out =
  loop
    await s_in(n) in
    if (n mod prime) <> 0 then emit s_out n
  end

let rec process sift s_in s_out =
  await s_in(prime) in
  emit s_out prime; (* emit a discovered prime *)
  signal s default 0 gather (+) in
  run (filter prime s_in s)
  ||
  run (sift s s_out)
```

# Crible d'Ératosthène en Lucid Synchrone (sieve.ls)

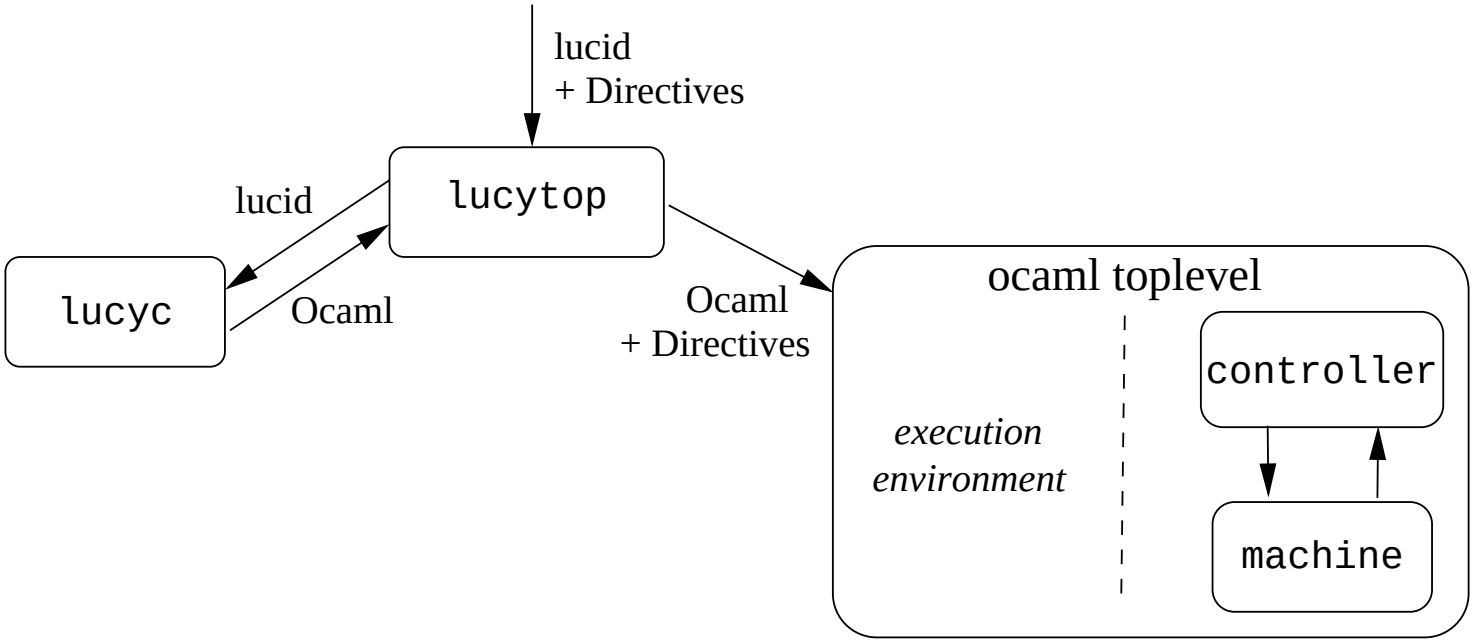
```
let node first x = v where
  rec v = x fby v

let node integers () = n where
  rec n = 1 fby n + 1

let rec node sift n =
  let prime = first n in
  let clock filter = (n mod prime) <> 0 in
  merge filter (sift (n when filter)) (true fby false)
```

Est-il possible de faire un toplevel pour un langage à la LUSTRE ?

# Implantation




- Code de `machine`

```
let react =
  let to_run = ref [] in
  fun to_add ->
    to_run := to_add @ !to_run;
    List.iter (fun step -> step ()) !to_run
```

# Variables globales

- Canaux partagés ≈ Signaux ReactiveML
  - écritures concurrentes
  - lecture avec retard
  - pas de contraintes d'horloges
- Canaux partagés
  - création :

```
share x default 0 gather (+)
```

  - écriture :

```
x <<- 4012
```

  - lecture :

```
last !x
```

# Exemples

- Particules
- Mobilité du $\pi$-calcul

$$
\begin{aligned}
P(x,z) &= \overline{x}\langle z\rangle.P' \\
Q(x) &= x(y).Q' \\
R(z) &= z(a).R' \\
Mob(x,z) &= P(x,z) \mid Q(x) \mid R(z)
\end{aligned}
$$

# Benchmarks

- Crible d'Ératosthène : 10000 premiers nombres premiers
  - Lucid Synchrone $\approx$ 2s
  - ReactiveML $\approx$ 36s

# Conclusion

- Les langages synchrones ne sont pas limités à la programmation de systèmes temps-réel critiques
- La causalité par construction et l'ordonnancement dynamique ne sont pas nécessaires à la création dynamique
- Une causalité peu expressive facilite l'implantation

# Travail en Cours sur l'Analyse de Réactivité

# Exemples

Les deux exemples les plus simples de programmes non réactifs :

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

*Warning: This expression may be an instantaneous loop.*

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

*Warning: This expression may produce an instantaneous recursion.*

Exemple de programme correct :

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

Dans la suite, on s'intéresse uniquement aux processus récursifs

# Sémantique petits pas

$$(\lambda x.e^{b})^{b_1}\ v^{b_2}/S \to_\varepsilon e^{b \vee b_1}[x \leftarrow v^{b_2}]/S \qquad \texttt{rec}\, x = v^{b}/S \to_\varepsilon v^{b}[x \leftarrow (\texttt{rec}\, x = v^{b})^{t}]/S$$

$$v^{b_1}\,;e^{b_2}/S \to_\varepsilon e^{b_2}/S \qquad \texttt{run}\ (\texttt{process}\ e^{b})^{f}/S \to_\varepsilon e^{b}/S$$

$$\texttt{let}\ x_1 = {v_1}^{b_1}\ \texttt{and}\ x_2 = {v_2}^{b_2}\ \texttt{in}\ e^{b_3}/S \to_\varepsilon e^{b_3}[x_1 \leftarrow {v_1}^{b_1}, x_2 \leftarrow {v_2}^{b_2}]/S$$

$$\texttt{emit}\ n^{b_n}/S \to_\varepsilon ()/S + n \qquad \texttt{present}\ n^{b}\ \texttt{then}\ {e_1}^{b_1}\ \texttt{else}\ {e_2}^{b_2}/S \to_\varepsilon {e_1}^{b_1}/S \ \text{si}\ n \in S$$

$$\texttt{signal}\ x\ \texttt{in}\ e^{b}/S \to_\varepsilon e^{b}[x \leftarrow n^{f}]/S \ \text{si}\ n \notin Dom(S)$$

$$\frac{e/S \to_\varepsilon e'/S'}{\Gamma(e)/S \to \Gamma(e')/S'} \qquad \text{avec}\ \Gamma ::= [\,] \mid \Gamma\,;e \mid \texttt{run}\ \Gamma \mid \ldots$$

# Énoncé de la propriétés de réactivité

**Définition 1 (Bonne formation)**

*Un expression $e$ est bien formée ($wf(e)$) si :*

$$\nexists \Gamma \text{ tel que } e = \Gamma[\texttt{run}\ ({e'}^t)]$$

**Théorème 1 (Sûreté du typage)**

*Si $e$ est « bien typée » et $e/S \rightarrow^* e'/S'$ et $e'/S' \not\rightarrow$ alors $wf(e')$*

# Structure de l'analyse actuelle

## 1. Analyse d'instantanéité

- exemples :

```
ackerman 3 10 (* e1 *)
print_int 1; pause; print_int 2 (* e2 *)
if x then pause else () (* e3 *)
```

- typage d'instantanéité : $e : k$ où $k ::= + \mid - \mid \pm$
- propriétés :
  1. Si $e : -$ et $\emptyset \vdash e : \tau$ et $e/S \rightarrow^* e'/S'$ et $e'$ est une forme normale alors $e'$ est une valeur
  2. Si $e : +$ et $\emptyset \vdash e : \tau$ et $e/S \rightarrow^* e'/S'$ et $e'$ est une forme normale alors $e'$ est une expression de fin d'instant mais $e'$ n'est pas une valeur
- preuve en Coq (avec Zaynah Dargaye)

# Structure de l'analyse actuelle

## 2. Détection des récursions instantanées

- système de type basé sur l'accès aux variables dangereuses
- une variable dangereuse est :
  - ▷ introduite par un `rec`
  - ▷ introduite par un `let` qui dépend d'une variable dangereuse
- les règles ont la forme :

$$\Pi \vdash e : \pi \quad \text{où} \quad \Pi, \pi ::= \emptyset \mid x : n, \Pi \ \text{ avec } \ n \in (\mathbb{N} \cup +\infty)$$

  - ▷ $\Pi$ représente l'ensemble des variables potentiellement dangereuses
  - ▷ $\pi$ représente l'ensemble des variables utilisées dans $e$

# Détection des récursions instantanées

$$\frac{x \notin Dom(\Pi)}{\Pi \vdash x : \emptyset} \qquad \frac{n = \Pi(x)}{\Pi \vdash x : \{x : n\}} \qquad \frac{}{\Pi \vdash c : \emptyset}$$

$$\frac{\Pi \vdash e : \pi}{\Pi \vdash \lambda x.e : \pi} \qquad \frac{\Pi \vdash e_1 : \pi_1 \quad \Pi \vdash e_2 : \emptyset}{\Pi \vdash e_1\, e_2 : \pi_1} \qquad \frac{x : 0, \Pi \vdash e : \pi}{\Pi \vdash \texttt{rec}\, x = e : \pi \backslash x}$$

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

# Problèmes

- Faux warnings sur des programmes purement OCaml

```
let rec f x =
  if x <= 0 then 1 else x * f (x-1)
```

- La solution actuelle laisse des réductions infinies de processus !

```
let fix f =
  let g = ref (fun _ -> assert false) in
  g := (fun x -> f !g x);
  !g


let process main =
  let f = fun p -> fun v -> process (run (p v)) in
  run (fix f ())
```

# Conclusion

- Est-ce la bonne propriété qui est exprimée ?
- Est-ce que le système de type n'est pas trop grossier ?
- Il faut faire la preuve de correction !
- En pratique :
  - l'analyse est implantée
  - elle aide à trouver des bogues

`http://rml.lri.fr`