Coiterative Synchronous Semantics
Part II: Control Structures

Marc Pouzet

Ecole normale supérieure
Paris

Marc.Pouzet@ens.fr

Course notes, MPRI, Nov. 2024

Language Extensions

Extension of the language kernel: local variables, last, default value, by-case definition of streams, mutually recursive definitions, hierarchical automata

The language kernel we have considered is similar to Lustre.

- It is first-order as Lustre but adds type polymorphism, a reset and an elementary control-structure to execute a block conditionally.
- All functions are length preserving: there is no `when/merge` or `current` operation.

We consider now an extended language that incorporates programming construct that exists in Lucid Synchrone, Zélus and Scade 6.

Details in the paper [Colaco et al., 2023] and the implementation of `ZRun` at:

`https://github.com/marcpouzet/zrun`

Mutually Recursive Equations

Equations are extended with local definitions:

$$
\begin{array}{rcl}
E & ::= & p = e \mid E \;\texttt{and}\; E \\
  &   & \quad\quad\;\; \mid \texttt{local}\; v \;\texttt{in}\; E \\
v & ::= & x \mid x \;\texttt{init}\; e \mid x \;\texttt{default}\; e \\
p & ::= & x \mid (p, ..., p)
\end{array}
$$

Expressions are extended with a construct to access the last value of a stream:

$$e \quad ::= \quad ... \mid \texttt{last}\; x$$

Environment

The construct `local` x `in` E declares x to be local in E.

The construct `local` x `init` e `in` E declares x to be local and the *last computed value of* x to be initialized with the value of e.

The construct `local` x `default` e `in` E declares x to be local and the *default value of* x to be the value of e, at instants where no definition of x is given.

By case definition of streams

If e is an expression whose type is a sum type $t = C_1 \mid \dots \mid C_n$,

- $\texttt{match}\ e\ \texttt{with}\ C_{i_1} \to E_1 \ \mid\ \dots\ \mid\ C_{i_n} \to E_n$ activates equation E_j such that i_j is the first index such that $e = C_{i_j}$, with $1 \leq i_1, \dots, i_n \leq n$.

$$E \quad ::= \quad \dots \mid \texttt{match}\ e\ \texttt{with}\ C \to E \dots C \to E$$

$\texttt{if}\ c\ \texttt{then}\ E_1\ \texttt{else}\ E_2$ is a short-case for
$\texttt{match}\ e\ \texttt{with}\ (\texttt{true} \to E_1) \mid (\texttt{false} \to E_2)$.

Hierarchical Automata

A automaton describes a system with several modes and transitions between them.
Such an automaton is characterized by:

- A finite set of states.
- In every state, a set of equations with variables that are possibly local to the state.
- A set (possibly empty) of "weak transitions" (keyword `until`) which define the active state for the next reaction.
- A set (possibly empty) of "strong transitions" (keyword `unless`) which define the active set of equations for the current reaction.
- Transitions can be by "reset" or by "history".

Rmq: Contrary to Scade 6 and Lucid Synchrone, weak and strong transitions cannot be mixed inside an automaton. This choice was introduced in Zélus.

The syntax is extended in the following way.

$$
\begin{array}{lcl}
E & ::= & \ldots \mid \texttt{automaton}\ (S(p) \rightarrow u\ wt)^{+} \\
 & & \quad \mid \texttt{automaton}\ (S(p) \rightarrow u\ st)^{+} \\
u & ::= & \texttt{local}\ v\ \texttt{in}\ u \mid \texttt{do}\ E \\
st & ::= & \texttt{unless}\ t^{*} \\
wt & ::= & \texttt{until}\ t^{*} \\
t & ::= & e\ \texttt{then}\ S(e, \ldots, e) \mid e\ \texttt{continue}\ S(e, \ldots, e)
\end{array}
$$

Examples in Zelus

```
type t = Incr | Decr | Idle

let f(c) =
   local o init 0
   do
    match c with
    | Idle -> (* o keeps its previous value, i.e., o = last o *)
              do done
    | Incr -> do o = last o + 1 done
    | Decr -> do o = last o - 1 done
   in o
```

Examples in Zelus

```
let node controller(auto, error, input) = output where rec
  automaton
  | Manual -> do output = input unless auto then Auto
  | Auto -> do output = run pid(p, i, d, error)
            unless (not auto) then Manual

let node await(a) = go where rec
  automaton
  | Await -> do go = false unless a then Run
  | Go -> do go = true done

let node abro(a, b, r) = go where rec
  reset automaton
        | Await -> do go = false
                   unless (run await(a) && run await(b))
                   then Go
        | Go -> do go = true done
  every r
```

Semantics

Environment

The environement is complemented to possibly associate a default or initial value to a variable.

$$\rho ::= \rho + [v/x] \mid \rho + [v/\texttt{default}\, x] \mid [v/\texttt{last}\, x] \mid []$$

If ρ and ρ' are two environments, we write $\rho\, \texttt{by}\, \rho'$ the completion of ρ with default or initial values from ρ'.

This operation is used to define the value of a variable in

$$\begin{array}{lcl}
\rho\, \texttt{by}\, [] & = & \rho \\
\rho\, \texttt{by}\, (\rho' + [v/\texttt{default}\, x]) & = & (\rho + [v/x])\, \texttt{by}\, \rho' \\
\rho\, \texttt{by}\, (\rho' + [v/\texttt{last}\, x]) & = & (\rho + [v/x])\, \texttt{by}\, \rho' \\
\rho\, \texttt{by}\, (\rho' + [v/x]) & = & \rho\, \texttt{by}\, \rho'
\end{array}$$

If p is a pattern and v is a value, $\texttt{match}\ v\ \texttt{with}\ p$ builds the environment by matching v by p such that:

$$\begin{array}{lcl}
[v|x] & = & [v/x] \\
[(v_1, v_2)|(p_1, p_2)] & = & [v_1|p_1] + [v_2|p_2]
\end{array}$$

If E is an equation, ρ is an environment, $[\![E]\!]_\rho^{Init}$ is the initial state and $[\![E]\!]_\rho^{State}$ is the step function. The semantics of an equation E is:

$$[\![E]\!]_\rho = [\![E]\!]_\rho^{Init}, [\![E]\!]_\rho^{State}$$

$$\begin{array}{lcl}
[\![p = e]\!]_\rho^{Init} & = & [\![e]\!]_\rho^{Init} \\
[\![p = e]\!]_\rho^{State} & = & \lambda s.let\ v, s = [\![e]\!]_\rho^{State}(s)\ in\ [v|p], s \\
[\![E_1 \texttt{ and } E_2]\!]_\rho^{Init} & = & ([\![E_1]\!]_\rho^{Init}, [\![E_2]\!]_\rho^{Init}) \\
[\![E_1 \texttt{ and } E_2]\!]_\rho^{State} & = & \lambda(s_1, s_2).let\ \rho_1, s_1 = [\![E_1]\!]_\rho^{State}(s_1)\ in \\
 & & \qquad let\ \rho_2, s_2 = [\![E_2]\!]_\rho^{State}(s_2)\ in \\
 & & \qquad \rho_1 + \rho_2, (s_1, s_2)
\end{array}$$

Notation: If $\rho = \rho' + [v/x]$, $\rho\backslash x = \rho'$.

$$
\begin{aligned}
[\![\texttt{local}\, x\, \texttt{in}\, E]\!]^{Init}_{\rho} &= [\![E]\!]^{Init}_{\rho} \\
[\![\texttt{local}\, x\, \texttt{in}\, E]\!]^{State}_{\rho}(s) &= \mathit{let}\ \rho', s = \mathit{fix}\left(\lambda s, \rho'.[\![E]\!]^{State}_{\rho+\rho'}(s)\right)(s)\ \mathit{in} \\
&\quad\ \rho'\backslash x, s \\
[\![\texttt{local}\, x\, \texttt{default}\, v\, \texttt{in}\, E]\!]^{Init}_{\rho} &= [\![E]\!]^{Init}_{\rho} \\
[\![\texttt{local}\, x\, \texttt{init}\, v\, \texttt{in}\, E]\!]^{Init}_{\rho} &= (v, [\![E]\!]^{Init}_{\rho})
\end{aligned}
$$

$$
\begin{aligned}
&[\![\texttt{local}\, x\, \texttt{default}\, v\, \texttt{in}\, E]\!]^{State}_{\rho}(s) = \\
&\qquad \mathit{let}\ \rho', s = \mathit{fix}\left(\lambda \rho', s.[\![E]\!]^{State}_{\rho+\rho'+[v/\texttt{default}\, x]}(s)\right)\ \mathit{in} \\
&\qquad \rho'\backslash x, s
\end{aligned}
$$

$$
\begin{aligned}
&[\![\texttt{local}\, x\, \texttt{init}\, v\, \texttt{in}\, E]\!]^{State}_{\rho}(w, s) = \\
&\qquad \mathit{let}\ \rho', s = \mathit{fix}\left(\lambda \rho', s.[\![E]\!]^{State}_{\rho+\rho'+[w/\texttt{last}\, x]}(s)\right)\ \mathit{in} \\
&\qquad \rho'\backslash x, (\rho'(x), s)
\end{aligned}
$$

Semantics for conditionals

The semantics for a conditional must consider the case where a branch defines a value for a variable `x` in one branch but not the other branch. We take the following convention:

- If a variable x is declared with a default value v, then a missing equation for x in a branch means that $x = v$ in that branch.
- Otherwise, $x = \texttt{last}\, x$, that is, x keeps its previous value.
- If x is declared with an initial value for $\texttt{last}\, x$, this means that x has a definition in every branch. Otherwise, there is a potential initialisation issue which has to be checked by other means.

Semantics for Conditionals

$$[\![\texttt{if } e \texttt{ then } E_1 \texttt{ else } E_2]\!]_\rho^{Init} = ([\![e]\!]_\rho^{Init}, [\![E_1]\!]_\rho^{Init}, [\![E_2]\!]_\rho^{Init})$$

$$\begin{aligned}
&[\![\texttt{if } e \texttt{ then } E_1 \texttt{ else } E_2]\!]_\rho^{State}(s, s_1, s_2) = \\
&\qquad \textit{let } v, s = [\![e]\!]_\rho^{State}(s) \textit{ in} \\
&\qquad \textit{if } v \textit{ then let } \rho_1, s_1 = [\![E_1]\!]_\rho^{State}(s_1) \textit{ in} \\
&\qquad\qquad \rho_1 \texttt{ by } \rho[N \backslash N_1], (s, s_1, s_2) \\
&\qquad \textit{else let } \rho_2, s_2 = [\![E_2]\!]_\rho^{State}(s_2) \textit{ in} \\
&\qquad\qquad \rho_2 \texttt{ by } \rho[N \backslash N_2], (s, s_1, s_2)
\end{aligned}$$

where $N = N1 \cup N_2$and $N_1 = Def(E_1)$and $N_2 = Def(E_2)$

$$[\![\texttt{match } e \texttt{ with } (C_i \to E_i)_{i \in [1..n]}]\!]_\rho^{Init} = ([\![e]\!]_\rho^{Init}, [\![E_1]\!]_\rho^{Init}, ..., [\![E_n]\!]_\rho^{Init})$$

The Transition Function:

$$\llbracket \texttt{match}\ e\ \texttt{with}\ (C_i \rightarrow E_i)_{i\in[1..n]} \rrbracket_\rho^{State}(s, s_1, ..., s_n) =$$
$$let\ v, s = \llbracket e \rrbracket_\rho^{State}(s)\ inmatch\ v\ with$$
$$\left(\begin{array}{l} C_i \rightarrow let\ \rho_i, s_i = \llbracket E_i \rrbracket_\rho^{State}(s_i)\ in \\ \qquad\quad \rho_i\ \text{by}\ \rho[N \backslash N_i], (s, s_1, ..., s_n) \end{array} \right)_{i\in[1..n]}$$

where $N = \cup_{i\in[1..n]}(N_i)$and $N_i = Def(E_i)$

The Last Computed Value:

$$\begin{array}{lcl} \llbracket \texttt{last}\ x \rrbracket_\rho^{Init} = () & & \\ \llbracket \texttt{last}\ x \rrbracket_\rho^{State} & = & \lambda s.\rho(\texttt{last}\ x), s \end{array}$$

Initial state of the transition function

$$
\begin{array}{l}
[\![\texttt{automaton}\ (S_i(p_i) \to u_i\ wt_i)_{i\in[1..n]}]\!]^{Init}_\rho \quad = \\
\quad let\ (s_i = [\![u_i]\!]^{Init}_\rho)_{i\in[1..n]}\ in \\
\quad let\ (s'_i = [\![wt_i]\!]^{Init}_\rho)_{i\in[1..n]}\ in \\
\quad (S_0(), false, (s_1,\ldots,s_n), (s'_1,\ldots,s'_n)) \\
[\![\texttt{automaton}\ (S_i(p_i) \to u_i\ st_i)_{i\in[1..n]}]\!]^{Init}_\rho \quad = \\
\quad let\ (s_i = [\![u_i]\!]^{Init}_\rho)_{i\in[1..n]}\ in \\
\quad let\ (s'_i = [\![st_i]\!]^{Init}_\rho)_{i\in[1..n]}\ in \\
\quad (S_0(), false, (s_1,\ldots,s_n), (s'_1,\ldots,s'_n))
\end{array}
$$

$$
\begin{array}{l}
[\![\texttt{automaton}\ (S_i(p_i) \to u_i\ wt_i)_{i\in[1..n]}]\!]^{State}_\rho(v,r,s,s') = \\
\quad let\ (\rho, v, r), (s, s') = [\![(S_i(p_i) \to u_i\ wt_i)_{i\in[1..n]}]\!]^{v,r}_\rho(s,s')\ in \\
\quad \rho, (v, r, s, s') \\
[\![\texttt{automaton}\ (S_i(p_i) \to u_i\ st_i)_{i\in[1..n]}]\!]^{State}_\rho(v,r,s,s') = \\
\quad let\ (\rho, v, r), (s, s') = [\![(S_i(p_i) \to u_i\ st_i)_{i\in[1..n]}]\!]^{v,r}_\rho(s,s')\ in \\
\quad \rho, (v, r, s, s')
\end{array}
$$

$$
\begin{array}{l}
[\![(S_i(p_i) \rightarrow u_i\ wt_i)_{i\in[1..n]}]\!]^{v,r}_{\rho}((s_1,...,s_n),(s'_1,...,s'_n)) = \\
\quad match\ v\ with \\
\quad\quad \left(\begin{array}{l} S_i(p_i) \rightarrow let\ \rho, s_i = [\![u_i]\!]^{r}_{\rho}(s_i)\ in \\ \qquad\qquad\quad let\ (v,r), s'_i = [\![wt_i]\!]^{v,r}_{\rho}(s'_i)\ in \\ \qquad\qquad\quad \rho,(v,r,(s_1,...,s_n),(s'_1,...,s'_n)) \end{array} \right)_{i\in[1..n]}
\end{array}
$$

$$
\begin{array}{l}
[\![(S_i(p_i) \rightarrow u_i\ st_i)_{i\in[1..n]}]\!]^{v,r}_{\rho}((s_1,...,s_n),(s'_1,...,s'_n)) = \\
\quad let\ (v,r,(s'_1,...,s'_n) = \\
\quad\quad match\ v\ with \\
\quad\quad\quad \left(\begin{array}{l} S_i(p_i) \rightarrow let\ (v,r), s'_i = [\![st_i]\!]^{v,r}_{\rho}(s'_i)\ in \\ \qquad\qquad\quad (v,r,(s'_1,...,s'_n)) \end{array} \right)_{i\in[1..n]} \\
\quad in\, match\ v\ with \\
\quad\quad\quad \left(\begin{array}{l} S_i(p_i) \rightarrow let\ \rho, s_i = [\![u_i]\!]^{r}_{\rho}(s_i)\ in \\ \qquad\qquad\quad \rho,(v,r,(s_1,...,s_n),(s'_1,...,s'_n)) \end{array} \right)_{i\in[1..n]}
\end{array}
$$

$$
\begin{array}{lcl}
[\![\texttt{until}\ t^*]\!]_\rho^{Init} & = & [\![t^*]\!]_\rho^{Init} \\
[\![\texttt{unless}\ t^*]\!]_\rho^{Init} & = & [\![t^*]\!]_\rho^{Init} \\
[\![\texttt{until}\ t^*]\!]_\rho^{v,r}(s) & = & [\![t^*]\!]_\rho^{v,r}(s) \\
[\![\texttt{unless}\ t^*]\!]_\rho^{v,r}(s) & = & [\![t^*]\!]_\rho^{v,r}(s) \\
[\![\epsilon]\!]_\rho^{Init} & = & \texttt{()} \\
[\![e\ \texttt{then}\ se\ t^*]\!]_\rho^{Init} & = & ([\![e]\!]_\rho^{Init}, [\![se]\!]_\rho^{Init}) \\
[\![e\ \texttt{continue}\ se\ t^*]\!]_\rho^{Init} & = & ([\![e]\!]_\rho^{Init}, [\![se]\!]_\rho^{Init}) \\
[\![\epsilon]\!]_\rho^{v,r}(s) & = & (v, r), s
\end{array}
$$

$$
\begin{aligned}
&[\![e\ \texttt{then}\ se\ t^*]\!]_\rho^{v,r}((s_1, s_2), s_3) = \\
&\qquad \mathit{let}\ s_1 = \mathit{if}\ r\ \mathit{then}\ [\![e]\!]_\rho^{\mathit{Init}}\ \mathit{else}\ s_1\ \mathit{in} \\
&\qquad \mathit{let}\ s_2 = \mathit{if}\ r\ \mathit{then}\ [\![se]\!]_\rho^{\mathit{Init}}\ \mathit{else}\ s_2\ \mathit{in} \\
&\qquad \mathit{let}\ s_3 = \mathit{if}\ r\ \mathit{then}\ [\![t^*]\!]_\rho^{\mathit{Init}}\ \mathit{else}\ s_3\ \mathit{in} \\
&\qquad \mathit{let}\ c, s_1 = [\![e]\!]_\rho^{\mathit{State}}(s_1)\ \mathit{in} \\
&\qquad \mathit{if}\ c\ \mathit{then}\ \mathit{let}\ v, s_2 = [\![se]\!]_\rho^{\mathit{State}}(s_2)\ \mathit{in}\ (v, \mathit{true}), ((s_1, s_2), s_3) \\
&\qquad \mathit{else}\ \mathit{let}\ (v, r), s_2 = [\![t^*]\!]_\rho^{v,r}(s)\ \mathit{in}\ (v, r), (s_1, s_2)
\end{aligned}
$$

$$
\begin{aligned}
&[\![e\ \texttt{continue}\ se\ t^*]\!]_\rho^{v,r}((s_1, s_2), s_3) = \\
&\qquad \mathit{let}\ s_1 = \mathit{if}\ r\ \mathit{then}\ [\![e]\!]_\rho^{\mathit{Init}}\ \mathit{else}\ s_1\ \mathit{in} \\
&\qquad \mathit{let}\ s_2 = \mathit{if}\ r\ \mathit{then}\ [\![se]\!]_\rho^{\mathit{Init}}\ \mathit{else}\ s_2\ \mathit{in} \\
&\qquad \mathit{let}\ s_3 = \mathit{if}\ r\ \mathit{then}\ [\![t^*]\!]_\rho^{\mathit{Init}}\ \mathit{else}\ s_3\ \mathit{in} \\
&\qquad \mathit{let}\ c, s_1 = [\![e]\!]_\rho^{\mathit{State}}(s_1)\ \mathit{in} \\
&\qquad \mathit{if}\ c\ \mathit{then}\ \mathit{let}\ v, s_2 = [\![se]\!]_\rho^{\mathit{State}}(s_2)\ \mathit{in}\ (v, \mathit{false}), ((s_1, s_2), s_3) \\
&\qquad \mathit{else}\ \mathit{let}\ (v, r), s_2 = [\![t^*]\!]_\rho^{v,r}(s)\ \mathit{in}\ (v, r), (s_1, s_2)
\end{aligned}
$$

$$
\begin{aligned}
[\![S(e_1, \ldots, e_n)]\!]_\rho^{\mathit{Init}} &= [\![e_1]\!]_\rho^{\mathit{Init}}, \ldots, [\![e_n]\!]_\rho^{\mathit{Init}} \\
[\![S(e_1, \ldots, e_n)]\!]_\rho^{\mathit{State}} &= \mathit{let}\ (v_i, s_i = [\![e_i]\!]_\rho^{\mathit{State}}(s_i))_{i \in [1..n]}\ \mathit{in} \\
&\ S(v_1, \ldots, v_n), (s_1, \ldots, s_n)
\end{aligned}
$$

Interpretation

- The transition function associated with the automaton construct is executed in an initial state.
- This state if of the form (ps, pr, s, s'). ps is the current state of the automaton. It is initialised with the initial state of the automaton. pr is the reset status. It is initialized with the value false. s is the state to execute the code of the strong transitions; s' is the state to execute the body of the automaton; s' is the state to execute the transitions.
- For an automaton with weak transition, the body is executed, then the transitions.
- For an automaton with strong transitions, the code of transitions of the current state are executed. This determines the active state. Then, the corresponding body is executed.

Exercices/questions

- Defines the semantics of e_1 `fby` e_2.
- Based on the previous definitions, write a small interpretor in Haskell or OCaml for a tiny language.
- Express the transition function and initial state directly as values in Haskell or OCaml where fix-point computation is replaced by lazy evaluation.
- Compare the efficiency between the two approaches.

References I

Colaco, J.-L., Mendler, M., Pauget, B., and Pouzet, M. (2023).
A Constructive State-based Semantics and Interpreter for a Synchronous Data-flow Language with State machines.
In *International Conference on Embedded Software (EMSOFT'23)*, Hamburg, Germany. ACM.