Encoding Esterel in Synpatick

Claude Stolze, joint work with Luigi Liquori and Michael Mendler

Some Synpatick aims ...

- to define *confluence* constraint properties
- to define the *semantics* of synchronous languages
- to act as a *typed intermediate language* in the compilation process
- to act as a *bridge* with λ-calculus
- to relate *scheduling* issues in SL with CBV/CBN, and CPS style in functional languages
- ...

Synpatick syntax

- Channel names $a, b, c \in \mathcal{A}$, co-names $\overline{a}, \overline{b}, \overline{c} \in \overline{\mathcal{A}}$; $\mathcal{R} = \mathcal{A} \cup \overline{\mathcal{A}}$
- Clock names $\sigma \in \mathcal{C}$
- Action labels $\alpha \in \mathcal{L} \overset{def}{=} \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{C}$
- Process names $A \in \mathcal{I}$ defined by user as $A := P$ (possibly recursive)

P, Q	$::=$	0	stop (inaction)
	$\mid$	A	name, $A \in \mathcal{I}$
	$\mid$	$\alpha{:}H.P$	action, $\alpha \in \mathcal{L}$, $H \subseteq \mathcal{L}$
	$\mid$	$P + Q$	choice
	$\mid$	$P \mid Q$	parallel
	$\mid$	$P \setminus L$	restriction
	$\mid$	$P \,/\, L$	hiding

Process (structural) equivalence as usual

- $P \mid Q \equiv Q \mid P$
- $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
- ...

Semantics: admissible transitions (1/2)

$P \xrightarrow[R]{\alpha}_H Q$ means:

- P emits α and reduces to Q
- The actions in H have priority over α
- R is the concurrent subterms of P that may compete with α
- If R wants to do some action in $\overline{H}$, the transition is admissible but not enabled (we *could*, but we *shall* not)

$$\frac{}{\alpha{:}H.P \xrightarrow[0]{\alpha}_H P}\;(Act) \qquad \frac{P \equiv P' \quad P' \xrightarrow[R']{\alpha}_H Q' \quad Q' \equiv Q \quad R' \equiv R}{P \xrightarrow[R]{\alpha}_H Q}\;(Struct)$$

$$\frac{A := P \quad P \xrightarrow[R]{\alpha}_H P'}{A \xrightarrow[R]{\alpha}_H P'}\;(Con) \qquad \frac{P \xrightarrow[R]{\alpha}_H Q \quad L'{=}L \cup \overline{L} \quad \alpha \notin L' \quad H'{=}H - L'}{P \setminus L \xrightarrow[R \setminus L]{\alpha}_{H'} Q \setminus L}\;(Restr)$$

$$\frac{P \xrightarrow[R]{\alpha}_H P'}{P + Q \xrightarrow[R]{\alpha}_H P'}\;(Sum) \qquad \frac{P \xrightarrow[R]{\alpha}_H P' \quad \alpha \notin \mathcal{C}}{P \mid Q \xrightarrow[R \mid Q]{\alpha}_H P' \mid Q}\;(Par)$$

Semantics: admissible transitions (2/2)

$$\frac{P \xrightarrow[R]{\alpha}_H Q \quad H'{=}H - L}{P \,/\, L \xrightarrow[R\,/\,L]{\alpha\,/\,L}_{H'} Q \,/\, L} \;(Hide) \quad where \quad \alpha \,/\, L \stackrel{def}{=} \begin{cases} \tau & \text{if } \alpha \in L \\ \alpha & \text{otherwise} \end{cases}$$

$$\frac{P \xrightarrow[R_1]{\alpha}_{H_1} P' \quad Q \xrightarrow[R_2]{\overline{\alpha}}_{H_2} Q' \quad H = race(P, Q, H_1, H_2)}{P \mid Q \xrightarrow[R_1 \mid R_2]{\alpha \mid \overline{\alpha}}_{H_1 \cup H_2 \cup H} P' \mid Q'} \;(Com)$$

$$race(P, Q, H_1, H_2) \stackrel{def}{=} \begin{cases} \{\tau\} & \text{if } H_1 \cap \overline{\mathrm{iA}}(Q) \not\subseteq \{\alpha\} \text{ or } H_2 \cap \overline{\mathrm{iA}}(P) \not\subseteq \{\overline{\alpha}\} \\ \{\,\} & \text{otherwise} \end{cases}$$

$$\alpha \mid \overline{\alpha} \stackrel{def}{=} \begin{cases} \tau & \text{if } \alpha \in \mathcal{A} \cup \overline{\mathcal{A}} \\ \alpha & \text{if } \alpha \in \mathcal{C} \end{cases}$$

Constructively enabled transitions

- Only transitions which are constructively enabled, or c-enabled for short, are legal
- A transition $P \xrightarrow[R]{\alpha}_H Q$ is c-enabled if no possible future transition of R in the current clock cycle belongs to $\overline{H}$
- It means we have to find a scheduling policy before executing processes
- The initial actions of a process R are

$$\mathrm{iA}(R) = \{\alpha \in \mathcal{L} \cup \{\tau\} \mid \exists Q.\, R \xrightarrow{\alpha} Q\}$$

- The set $\mathrm{iA}^*(P) \subseteq \mathcal{L}$ of potential actions is the smallest extension $\mathrm{iA}(P) \subseteq \mathrm{iA}^*(P)$ such that if $\ell \in \mathrm{iA}^*(Q)$ and $P \xrightarrow{\alpha} Q$ for $\alpha \in \mathcal{R} \cup \{\tau\}$, then $\ell \in \mathrm{iA}^*(P)$
- Formally, a transition $P \xrightarrow[R]{\alpha}_H Q$ is c-enabled, if

$$H \cap (\overline{\mathrm{iA}}^*(R) \cup \{\tau\}) = \{\,\}$$

Coherence (see MM)

- Two transitions $Q \xrightarrow[E_1]{\alpha_1}_{H_1} Q_1$ and $Q \xrightarrow[E_2]{\alpha_2}_{H_2} Q_2$ are independent if
  - $\alpha_1 = \alpha_2$ and $Q_1 \not\equiv Q_2$, or
  - $\{\alpha_1, \alpha_2\} \neq \{\tau\}$, $\alpha_1 \notin H_2$ and $\alpha_2 \notin H_1$

- A process P is (structurally) coherent if for all its derivatives Q the following holds: For any two independent transitions

$$Q \xrightarrow[E_1]{\alpha_1}_{H_1} Q_1 \text{ and } Q \xrightarrow[E_2]{\alpha_2}_{H_2} Q_2$$

  there exist Q' s.t.

$$Q_1 \xrightarrow[E_2']{\alpha_2}_{H_2'} Q' \text{ and } Q_2 \xrightarrow[E_1']{\alpha_1}_{H_1'} Q'$$

  are c-enabled

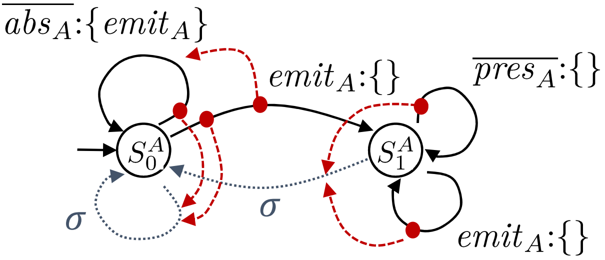
- Coherent processes are determinate under c-enabled reductions, i.e., $P \Downarrow Q_1$ and $P \Downarrow Q_2$ implies $Q_1 \equiv Q_2$

- (Ongoing work) Finding a typing system to ensure processes are coherent

Esterel signals

$$S_0^A \quad := \quad \overline{abs_A}{:}emit_A.S_0^A + emit_A.S_1^A + \sigma{:}\{\overline{abs_A}, emit_A\}.S_0^A$$

$$S_1^A \quad := \quad \overline{pres_A}.S_1^A + emit_A.S_1^A + \sigma{:}\{\overline{pres_A}, emit_A\}.S_0^A$$



Encoding Esterel in Synpatick: core ideas

- We use only one clock σ
- Action $\overline{\mathit{done}}$ when Esterel process is terminating
- Priorities between actions correspond to a scheduling
- Actions which may abort the process have higher priorities. The process to deal with those is:

$$Exc_E \stackrel{def}{=} \sum_{e \in E} e.\overline{done}$$

- A local Esterel program `prog` is translated to $[\![\mathtt{prog}]\!]_E$
- A global Esterel program `prog` with signals $A_1, \ldots, A_n$ is translated to

$$[\![\mathtt{prog}]\!]_{\{\}} \mid S_0^{A_1} \mid \cdots \mid S_0^{A_n}$$

Some instructions

- `nothing` temrinates immediately

$$[\![\texttt{nothing}]\!]_E \stackrel{def}{=} \overline{done}$$

- `emit x` emits a signal `x`, which is managed either by S_0^x or S_1^x in our encoding

$$[\![\texttt{emit x}]\!]_E \stackrel{def}{=} Exc_E + \overline{emit_x}{:}E.\overline{done}$$

- `present x then P else Q end` does `P` if there is $pres_x$, and `Q` if there is abs_x

$$[\![\texttt{present x then P else Q end}]\!]_E \stackrel{def}{=}$$

$$Exc_E + pres_x{:}E.[\![\mathrm{P}]\!]_E + abs_x{:}\{pres_x\} \cup E.[\![\mathrm{Q}]\!]_E$$

Sequence and parallelism

- Following Milner, we use a local action d to force a process to execute before another

$$[\![\mathtt{P};\mathtt{Q}]\!]_E \stackrel{def}{=} ([\![\mathtt{P}]\!]_E[d/done] \mid next_{\mathtt{Q},E,d}) \setminus d$$

$$next_{\mathtt{Q},E,d} \stackrel{def}{=} Exc_E + d{:}E.[\![\mathtt{Q}]\!]_E + \sigma{:}\{d\} \cup E.next_{\mathtt{Q},E,d}$$

- We also use a local action d to ensure two parallel processes are both done
- When we count d twice, we are done

$$[\![\mathtt{P} \mid\mid \mathtt{Q}]\!]_E \stackrel{def}{=} ([\![\mathtt{P}]\!][d/done] \mid [\![\mathtt{Q}]\!][d/done] \mid count2_{E,d}) \setminus d$$

$$count2_{E,d} \stackrel{def}{=} Exc_E + d{:}E.count1_{E,d} + \sigma{:}\{d\} \cup E.count2_{E,d}$$

$$count1_{E,S,d} \stackrel{def}{=} Exc_E + d{:}E.\overline{done} + \sigma{:}\{d\} \cup E.count1_{E,S,d}$$

Properties (1/2)

$$\texttt{P} \overset{def}{=} \texttt{present A then nothing else emit A}$$

The process `P` is non-constructive in Esterel. What happens in the translation?

$$\llbracket \mathrm{P} \rrbracket_{\{\}} \overset{def}{=} pres_A.\overline{done} + abs_A{:}pres_A.\overline{emit_A}.\overline{done}$$

$$\llbracket \mathrm{P} \rrbracket_{\{\}} \mid S_0^A \overset{def}{=} (pres_A.\overline{done} + abs_A{:}pres_A.\overline{emit_A}.\overline{done}) \mid (\overline{abs_A}{:}emit_A.S_0^A + \ldots)$$

$$\llbracket \mathrm{P} \rrbracket_{\{\}} \mid S_1^A \overset{def}{=} (pres_A.\overline{done} + abs_A{:}pres_A.\overline{emit_A}.\overline{done}) \mid (\overline{pres_A}.S_1^A + \ldots)$$

- $\llbracket \mathrm{P} \rrbracket_{\{\}} \mid S_0^A$ is stuck (the $\overline{abs_A}$-transition is not c-enabled)
- $\llbracket \mathrm{P} \rrbracket_{\{\}} \mid S_1^A$ can progress, and reduces to $\overline{done} \mid S_1^A$

Properties (2/2)

$$\texttt{Q} \stackrel{\mathit{def}}{=} (\texttt{present A then nothing else emit A end}) \mid\mid \texttt{emit A}$$

Q is a constructive Esterel program

$$\begin{aligned}[\![\texttt{Q}]\!]_{\{\}} \mid S_0^A \quad \stackrel{\mathit{def}}{=} \quad & ((\mathit{pres}_A.\overline{d} + \mathit{abs}_A{:}\mathit{pres}_A.\overline{\mathit{emit}_A}.\overline{d}) \mid \overline{\mathit{emit}_A}.\overline{d} \mid \ldots) \setminus d \\ & \mid (\overline{\mathit{abs}_A}{:}\mathit{emit}_A.S_0^A + \mathit{emit}_A.S_1^A + \ldots)\end{aligned}$$

- The only c-enabled transitions are the emit_A and $\overline{\mathit{emit}_A}$
- $[\![\texttt{Q}]\!]_{\{\}} \mid S_0^A$ reduces deterministically to $[\![\mathrm{P}]\!]_{\{\}} \mid S_1^A$, then to S_1^A
- Conjecture: this encoding of Esterel in Synpatick is correct
- Proof: in progress ... (any suggestion is welcome)

Await

- `await` waits a cycle before calling `await immediate`

$$\llbracket \texttt{await immediate x} \rrbracket_E \stackrel{def}{=} Exc_E + pres_x{:}E.\overline{done} + \sigma{:}\{pres_x\} \cup E.\llbracket \texttt{await x} \rrbracket_E$$

$$\llbracket \texttt{await x} \rrbracket_E \stackrel{def}{=} Exc_E + \sigma{:}E.\llbracket \texttt{await immediate x} \rrbracket_E$$

- `loop P each x` restarts `P` each time the signal `x` appears, so $pres_x$ should be added to the set E
- We remove the *done* action because a loop is never done

$$\llbracket \texttt{loop P each x} \rrbracket_E \stackrel{def}{=} \llbracket \texttt{P} \rrbracket_{\{pres_x\} \cup E}[\tau/done] \mid endloop_{x,E,\texttt{P}}$$

$$endloop_{x,E,\texttt{P}} \stackrel{def}{=} pres_x.\sigma{:}E.\llbracket \texttt{loop P each x} \rrbracket_E + \sigma{:}\{pres_x\} \cup E.endloop_{x,E,\texttt{P}}$$

ABRO (automatic encoding)

```
loop
  [ await A || await B ];
  emit O
each R
```

$$
\begin{aligned}
ABRO &:= ABO \mid R \\
ABO &:= (AB \mid O) \setminus d' \\
AB &:= ((pres_R + \sigma{:}pres_R.A) \mid (pres_R + \sigma{:}pres_R.B \mid AB') \setminus d \\
A &:= pres_R + pres_A{:}pres_R.\overline{d} + \sigma{:}\{pres_A, pres_R\}.A \\
B &:= pres_R + pres_B{:}pres_R.\overline{d} + \sigma{:}\{pres_B, pres_R\}.B \\
AB' &:= pres_R + d{:}pres_R.AB'' + \sigma{:}\{pres_R, d\}.AB' \\
AB'' &:= pres_R + d{:}pres_R.\overline{d'} + \sigma{:}\{pres_R, d\}.AB'' \\
O &:= pres_R + d'{:}pres_R.(pres_R + \overline{emit_O}{:}pres_R) \\
R &:= pres_R.\sigma.ABRO + \sigma{:}pres_R.R
\end{aligned}
$$

ABRO (optimised)

```
loop
  [ await A || await B ];
  emit O
each R
```

$$
\begin{aligned}
ABRO &:= ABO \mid R \\
ABO &:= ((pres_R + \sigma{:}pres_R.(A \mid B)) \mid AB') \setminus d \\
A &:= pres_R + pres_A{:}pres_R.\overline{d} + \sigma{:}\{pres_A, pres_R\}.A \\
B &:= pres_R + pres_B{:}pres_R.\overline{d} + \sigma{:}\{pres_B, pres_R\}.B \\
AB' &:= pres_R + d{:}pres_R.AB'' + \sigma{:}\{pres_R, d\}.AB' \\
AB'' &:= pres_R + d{:}pres_R.\overline{emit_O} + \sigma{:}\{pres_R, d\}.AB'' \\
R &:= pres_R.\sigma.ABRO + \sigma{:}pres_R.R
\end{aligned}
$$

Ongoing work

- Find statically a scheduling of c-enabled actions
- Ensure the process is determinated
- Idea: a scheduling policy should act as a type
- Finding a good scheduling policy = type inference
- Issue: finding a type system where typing is
    - compositional
    - decidable
    - not adding too many constraints
- paper submitted at FoSSaCS

Thank you for listening