Towards a Coq-Verified Compiler from Esterel to Circuits

Lionel Rieg

Collège de France

December 4th, 2014

Long-term goal

Follow the ideas of CompCert:

- Esterel kernel is simpler than C
  $\leadsto$ I only consider Pure Esterel v5
- Coq allows extracting the compiler
- I could also go to C code and link with CompCert
- several transformation passes
  $\leadsto$ semantics is preserved across passes

Here, only the formalization of several semantics of Esterel
(it's just the beginning!)

High-level view of semantics

A reaction $P \longrightarrow^* P'$ of P into P' in several ticks

One reduction $p \rightarrow \ldots \rightarrow q$ for each clock tick, having:

- inputs I
- outputs O
- return code k

I only consider reductions, written $p \xrightarrow[I]{O,k} p'$

$$\begin{aligned} \text{emit s; pause; present i then pause else emit s} &\xrightarrow[I]{\{s\},1} \\ \text{present i then pause else emit s} &\xrightarrow[I]{\{s\},0} \\ \text{nothing} & \end{aligned}$$

Pure Esterel kernel syntax (commands)

$p, q :=$	0	nothing	
	1	pause	
	$T + 2$	exit T	T a de Bruijn integer
	$!s$	emit s	
	$s?p \cdot q$	present s then p else q end	
	$s \supset p$	suspend p when s	
	$p; q$		
	$p \,\|\, q$		
	$p*$	loop p end	
	$\uparrow p$		does not exist in the language
	$\{p\}$	trap p end	
	$p \backslash s$	signal s in p end	

+ macros, ex: $s \mathbin{\underset{\cdot}{\supset}} p := \{(s?1 \cdot 2)*\}; s \supset p$

Classical Behavioral Semantics

Notation for reductions: $p \xrightarrow[I]{O,k} p'$

- LBS (Logical Behavioral Semantics) [Gonthier]
  $\rightsquigarrow$ the reference semantics
- DS (Deterministic Semantics): [Tardieu]
  avoids non-determinism in $p \backslash s$
  $\rightsquigarrow$ same value of s in both branches
- RDS (Reactive Deterministic Semantics): [Tardieu]
  adds error cases ($k \in \mathbb{N} \cup \{\infty\}$)
  $\rightsquigarrow$ instantaneous loop — looperror
  $\rightsquigarrow$ $p \backslash s$: different values de s dans p — signalMP / signalPM
  $\rightsquigarrow$ + propagation (only $p \backslash s$) — signal∞

Difference for signal with s present

$$\frac{p \xrightarrow[I \cup \{s\}]{O_+, k_+} p' \qquad s \in O_+}{p \backslash s \xrightarrow[I]{O_+ \backslash \{s\}, k_+} p' \backslash s}\ \text{LBS}$$

$$\frac{p \xrightarrow[I \cup \{s\}]{O_+, k_+} p' \qquad p \xrightarrow[I \backslash \{s\}]{O_-, k_-} p' \qquad s \in O_+, O_-}{p \backslash s \xrightarrow[I]{O_+ \backslash \{s\}, k_+} p' \backslash s}\ \text{DS}$$

$$\frac{p \xrightarrow[I \cup \{s\}]{O_+, k_+} p' \qquad p \xrightarrow[I \backslash \{s\}]{O_-, k_-} p' \qquad s \in O_+, O_- \qquad k_+, k_- < \infty}{p \backslash s \xrightarrow[I]{O \backslash \{s\}, k_+} p' \backslash s}\ \text{RDS}$$

Properties

	deterministic	reactive (*i.e.* total)
LBS	✗	✗
DS	✓	✗
RDS	✓	✓

  $\leadsto$ RDS can be a function

- If $k < \infty$, RDS $\iff$ DS $\implies$ LBS

- In $p \xrightarrow[I]{O,\,k} p'$, if $k \neq 1$, then $p' = \textit{nothing}$.

  $\leadsto$ requires to add δ $\qquad$ $\delta\, k\, p$:= if $k = 1$ then p else nothing

- $p \xrightarrow[I]{O,\,\infty}_{\text{RDS}} p' \iff$ we use looperror or signalMP/signalPM

  $\leadsto$ requires to formally define “use”

  $\leadsto$ (p, I) error-free := they are not used in the reduction

What about Coq?

All results in this talk are proved in Coq

Feedback from using Coq:

- I, O = events or sets of signals?
  $\leadsto$ Gérard Berry *vs.* Olivier Tardieu
- higher-order logic: one definition for determinism

```
Definition deterministic {T : Type} (R : semantics T) p :=
  forall I O1 k1 p1 O2 k2 p2, R p I O1 k1 p1 → R p I O2 k2 p2 →
    S.Equal O1 O2 ∧ k1 = k2 ∧ p1 = p2.
```

- compatibility w.r.t. set equality on I and O
- find bugs: $\cancel{p \xrightarrow[I]{O,\infty}_{\mathrm{RDS}} 0 \implies O = \varnothing}$!s || 0*   [Tardieu]
  $\leadsto$ should we change the semantics?

Why a constructive semantics?

- remove all problems of:
  - non-determinism
  - backward dependency across sequence
- closer to circuit semantics [Berry, Mendler, Shiple]
  constructive circuit = stabilizes for all delays
- based on what Must/Can be done
  $\leadsto$ only affects $p \backslash s$
  - s true in $p \iff s \in \mathrm{Must}(p, E)$
  - s false in $p \iff s \notin \mathrm{Can}(p, E)$
  - otherwise, we block!

The Constructive Case

CBS = Constructive Behavioral Semantics
cf. Esterel Constructive Book

Must: what must be done

- Must_s: set of signals that must be emitted
- Must_k: the return code that p must return (at most 1)

Can: what can be done

- Can_s: set of signals that can be emitted
- Can_k: set of return codes that p can return

Again I, O: events or sets?
$\leadsto$ sets but also requires A, set of absent signals

Notation: $$p \xhookrightarrow[A,I]{O,k}_{\text{CBS}} p'$$

Properties of Can/Must

- Max K L defined by Gonthier's formula

$$\begin{aligned}\mathrm{Max}\,K\,L &= \{n \in \mathbb{N} \mid \exists kl, k \in K \wedge l \in L \wedge n = \max k\, l\} \\ &= \{n \in K \cup L \mid n \geqslant \min K \wedge n \geqslant \min L\}\end{aligned}$$

  \+ its specification:

  $\mathrm{Max}\,K\,L := \mathrm{filter}\,(\geqslant (\min L))\,(\mathrm{filter}\,(\geqslant (\min K))\,(K \cup L))$

  with $x \in (\mathrm{Max}\,K\,L) \iff \exists kl, k \in K \wedge l \in L \wedge x = \max k\, l$

- Must $p\,E \subseteq \mathrm{Can}^+\,p\,E$ (separate proof)

- monotony of Can and Must

  $\rightsquigarrow$ big mutual induction: 400 l. of proof!

Properties of CBS

- Deterministic but not reactive
  $\leadsto$ Must/Can may get stuck
  $\leadsto$ we can use ∞ to track errors
- In $p \xhookrightarrow[A,I]{O,k} p'$, if $k \neq 1$, then p' = nothing (requires some δ)
- CBS et Can/Must: if $p \xhookrightarrow[A,I]{O,k} p'$, then
  - $s \in \mathrm{Must}_s\, p\, E_{A,I} \implies s \in O$ + idem for k
  - $s \in O \implies s \in \mathrm{Can}_s^m\, p\, E_{A,I}$ + idem for k
- Same for LBS (mutually recursive proof of 200 l.)
- CBS $\implies$ LBS mais CBS $\not\Longrightarrow$ RDS
- If p error-free, CBS $\implies$ RDS
  $\leadsto$ CBS ignores errors inside unreachable code ($p \backslash s$)

Why a state semantics?

Intermediate step between Esterel and circuits:

- on source code but the program does not change (unlike LBS)
- program counters indicate where we are
- state semantics very close to circuit semantics
- pause statements are mapped to registers

$\downarrow$ emit s; pause; present i then pause else emit s $\xrightarrow[I]{\{s\},1}$

emit s; pause; $\downarrow$ present i then pause else emit s $\xrightarrow[I]{\{s\},0}$

emit s; pause; present i then pause else emit s$\downarrow$

Commands and states

state = command being evaluated

$$
\begin{array}{r|l}
\hat{p}, \hat{q} := & 0 \\
& 1 \\
& T + 2 \\
& !s \\
& s?p \cdot q \\
& s \supset p \\
& p; q \\
& p \,\|\, q \\
& p* \\
& \uparrow p \\
& \{p\} \\
& p \backslash s
\end{array}
$$

Commands and states

state = command being evaluated

$$
\hat{p}, \hat{q} := \left|\begin{array}{l} \\ 1 \\ \\ \\ s?p \cdot q \\ s \supset p \\ p; q \\ p \parallel q \\ p* \\ \uparrow p \\ \{p\} \\ p\backslash s \end{array}\right.
$$

Commands and states

state = command being evaluated

$\hat{p}, \hat{q} :=$

| $\hat{1}$ activated pause
| $s?p \cdot q$
| $s \supset p$
| $p; q$
| $p \parallel q$
| $p*$
| $\uparrow p$
| $\{p\}$
| $p \backslash s$

Commands and states

state = command being evaluated

$$
\hat{p}, \hat{q} := \left|
\begin{array}{l}
\hat{1} \qquad\qquad \text{activated pause} \\
\\
s?\hat{p} \cdot q \quad | \quad s?p \cdot \hat{q} \\
s \supset \hat{p} \\
\hat{p}; q \quad | \quad p; \hat{q} \\
\hat{p} \,||\, q \quad | \quad p \,||\, \hat{q} \quad | \quad \hat{p} \,||\, \hat{q} \\
\hat{p}* \\
\uparrow \hat{p} \\
\{\hat{p}\} \\
\hat{p} \backslash s
\end{array}
\right.
$$

Commands and states

state = command being evaluated

$$
\begin{array}{r|ll}
\hat{p}, \hat{q} := & & \\
& \hat{1} & \text{activated pause} \\
& & \\
& s?\hat{p} \cdot q \quad | \quad s?p \cdot \hat{q} & \\
& s \supset \hat{p} & \\
& \hat{p}; q \quad | \quad p; \hat{q} & \\
& \hat{p} \,\|\, q \quad | \quad p \,\|\, \hat{q} \quad | \quad \hat{p} \,\|\, \hat{q} & \\
& \hat{p}* & \\
& \uparrow \hat{p} & \\
& \{\hat{p}\} & \\
& \hat{p} \backslash s &
\end{array}
$$

$\hat{p}$ = state
(computation pending)

p = command
(computation done)

$\overline{p} := \hat{p} \quad | \quad p$ (term)

State Semantics

LSBS = Logical State Behavioral Semantics

- 2 types of rules: s-rules (start) and r-rules (resume)
  $\rightsquigarrow$ 2 distinct inductive definitions:
  first sLSBS — from commands to terms
  then rLSBS — from states to terms
  $\rightsquigarrow$ LSBS = sLSBS $\cup$ rLSBS — from terms to terms

- main theorem:

$$\hat{p} \xrightarrow[I]{O,k}_{\text{rLSBS}} \overline{p'} \implies \exists q, \mathcal{E}(\hat{p}) \xrightarrow[I]{O,k}_{\text{LBS}} q \wedge \delta\, k\, (\mathcal{E}(\overline{p'})) \equiv q$$

where $p \equiv q$:= "p and q equivalent"
$\mathcal{E}(\hat{p})$:= expansion of $\hat{p}$
= what will be executed in the next tick

Bugs corrected

- $p \equiv q$: immediate equivalence *vs.* bisimulation

$$\begin{tikzcd} p \arrow[dr] \arrow[d, dash, dashed, "\equiv_{imm}"] & \\ q \arrow[r] & p' \end{tikzcd} \qquad vs. \qquad \begin{tikzcd} p \arrow[r] \arrow[d, dash, dashed, "\equiv"] & p' \arrow[r] \arrow[d, dash, dashed, "\equiv"] & \cdots \\ q \arrow[r] & q' \arrow[r] & \cdots \end{tikzcd}$$

  - $\equiv_{imm}$ not compatible with the kernel:
    $p \equiv_{imm} q \;\not\Longrightarrow\; s \supset p \equiv_{imm} s \supset q$ (when s is present)
  - bisimulation requires coinduction

- small mistake in the definition of states:
  $\overline{p} \,||\, \overline{q}$ allows two commands $(p \,||\, q)$

- $\hat{p} \xrightarrow[I]{O,k}_{\text{rLSBS}} p' \implies (k = 1 \iff p' \text{ state})$
  $\rightsquigarrow$ was wrong for $p \,||\, q$ (when $k_p = 1$ and $k_q > 1$)

Constructive State Behavioral Semantics

Yet another semantics: CSBS

- Same restriction on LSBS as going from LBS to CBS (signal declaration must be constructive)
  $\rightsquigarrow$ Must/Can extended to states
- proofs work exactly the same

Conclusion & Next Steps

Summary:

- formal proofs of “obvious results” (except the wrong ones)
- iron out a few bugs
- only semantics here
  $\rightsquigarrow$ necessary to compare Esterel and circuits

Next steps:

- get to circuits!
  $\rightsquigarrow$ write their semantics
  $\rightsquigarrow$ (finally) write the compiler
- prove that compilation preserve semantics
- handle schizophrenia & verify optimization
- toward full Esterel?

Conclusion & Next Steps

Summary:

- formal proofs of “obvious results” (except the wrong ones)
- iron out a few bugs
- only semantics here
  $\rightsquigarrow$ necessary to compare Esterel and circuits

Next steps:

- get to circuits!
  $\rightsquigarrow$ write their semantics
  $\rightsquigarrow$ (finally) write the compiler
- prove that compilation preserve semantics
- handle schizophrenia & verify optimization
- toward full Esterel?

Thank you
for your attention