Towards a Coq-verified compiler from Esterel to circuits: 2 years later

Lionel Rieg

Yale University

Synchron 2016
December 5th, 2016

Objective: prove the compilation scheme for Esterel

- Esterel
  - Synchronous dataflow language
  - Control-oriented, imperative-flavored — unlike Lustre
- Verified compilation to circuit
  - Draft book by Gérard Berry
    [The Constructive Semantics of Pure Esterel]
  - Modular compilation
  - Same spirit as Compcert:
    semantics is ~~refined~~/preserved by compilation
- Restrictions
  - Compilation toward digital circuits only
  - No data, only Pure Esterel v.5
  - No reincarnation, left for future work

Syntax of Kernel Esterel (instructions)

$p, q :=$			
	0	`nothing`	
	1	`pause`	
	$s??$	`await (immediate) s`	
	$!s$	`emit s`	
	$s\,?\,p\,,\,q$	`if s then p else q end`	
	$s \supset p$	`suspend p when s`	
	$p\,;\,q$	`p ; q`	
	$p \mid q$	`p || q`	
	$p*$	`loop p end`	
	$k \quad k \geqslant 2$	`exit` T^k	k is the level
	$\{p\}$	`trap T in p end`	
	$\uparrow p$		
	$p \backslash s$	`signal s in p end`	

+ macros:

`halt` $:= 1*$

`await` $s := \{(s\,?\,2\,,\,1)*\}$

`abort` p `when` $s := \{(s\,?\,2\,,\,1)* \mid (\uparrow p\,;\,2)\}$

Hello world in Esterel: ABRO

Idea:

- as soon as both *A* and *B* are received, emit *O*
- reinitialize when *R* is received

```
halt              := loop pause end
abort p when s    := trap T in
                         loop (if s then exit T else pause end) end
                       ||
                         (p ; exit T)
```

Hello world in Esterel: ABRO

Idea:

- as soon as both *A* and *B* are received, emit *O*
- reinitialize when *R* is received

```
(await A || await B) ;
emit O ;
halt
```

```
halt              :=  loop pause end
abort p when s    :=  trap T in
                          loop (if s then exit T else pause end) end
                        ||
                          (p ; exit T)
```

Hello world in Esterel: ABRO

Idea:

- as soon as both *A* and *B* are received, emit *O*
- reinitialize when *R* is received

```
abort
    (await A || await B);
    emit O;
    halt
when R
```

```
halt             := loop pause end
abort p when s   := trap T in
                        loop (if s then exit T else pause end) end
                      ||
                        (p; exit T)
```

Hello world in Esterel: ABRO

Idea:

- as soon as both *A* and *B* are received, emit *O*
- reinitialize when *R* is received

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

```
halt              := loop pause end
abort p when s    := trap T in
                         loop (if s then exit T else pause end) end
                       ||
                         (p; exit T)
```

Semantics of an Esterel Program

At each instant, either:

- One (macro-)step $p \xrightarrow[E]{E',k} p'$ with:
  - Inputs E
  - Outputs E'
  - A return code k 0 = done, 1 = pending, 2+ = exceptions
- Several microsteps
  - No E' and k: they can be read from p'
  - No Can/Must functions

Semantics of an Esterel Program

At each instant, either:

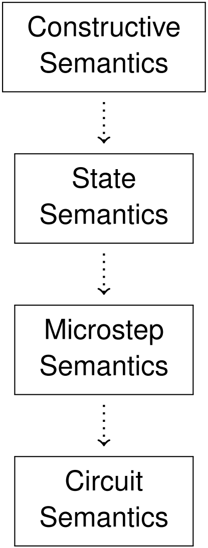
- One (macro-)step $p \xrightarrow[E]{E', k} p'$ with:
  - Inputs E
  - Outputs E'
  - A return code k 0 = done, 1 = pending, 2+ = exceptions
- Several microsteps
  - No E' and k: they can be read from p'
  - No Can/Must functions

Some remarks:

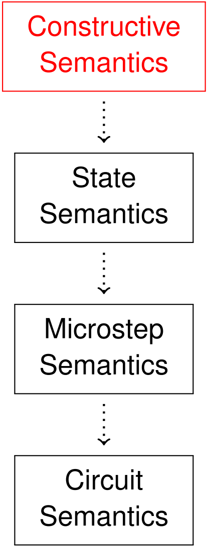
- E and E' are maps from declared signals to $\{-, \perp, +\}$
- Instantaneous communication: $E' \subseteq E$

Not compositional if not done carefully

Global diagram of semantics



Global diagram of semantics



Constructive Semantics

- Rewrite the program
  - Erase dead code & only keep active parts
  - Duplicate loop bodies $\quad$ `loop p end` $\equiv$ `p ; loop p end`
- Use Can/Must for local signals
  - s^+ if s must be emitted
  - s^- if s cannot be emitted
  - Avoid causality problems & non-determinism

    forbid “`if s then emit s else nothing end`”
- Usual style of programming language semantics

  $\rightsquigarrow$ convenient for high-level reasoning about programs
- The if-then rule: $$\frac{s^+ \in E \qquad p \xrightarrow[E]{E',k} p'}{s\,?\,p\,,\,q \xrightarrow[E]{E',k} p'}$$

Execution of ABRO

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

Execution of ABRO

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

$\{B\}$

Execution of ABRO

```
abort
   (await A || await B);
   emit O ;
   halt
when R ;
loop
   abort
      (await A || await B) ;
      emit O ;
      halt
   when R
end
```

$\{B\}$

Execution of ABRO

```
abort
   (await A || nothing);
    emit O ;
    halt
when R ;
loop
    abort
         (await A || await B) ;
         emit O ;
         halt
    when R
end
```

$\{B\}$

Execution of ABRO

```
abort
   (await A || nothing);
   emit O ;
   halt
when R ;
loop
   abort
      (await A || await B) ;
      emit O ;
      halt
   when R
end
```

$\{B\} \Longrightarrow \{A,$

Execution of ABRO

```
abort
   (nothing || nothing);
    emit O ;
    halt
when R ;
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

$\{B\} \Longrightarrow \{A,$

Execution of ABRO

```
abort

    emit O ;
    halt
when R ;
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
```

$\{B\} \Longrightarrow \{A,$

Execution of ABRO

```
abort


   halt
when R ;
loop
   abort
      (await A || await B);
      emit O ;
      halt
   when R
end
```

$\{B\} \Longrightarrow \{A, O\}$

Execution of ABRO

```
abort



    halt
when R ;
loop
    abort
        (await A || await B);
        emit O ;
        halt
    when R
end
```

$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\}$

Execution of ABRO

```
abort


   halt
when R ;
loop
   abort
      (await A || await B);
      emit O ;
      halt
   when R
end
```

$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\}$

Execution of ABRO

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\}$$

Execution of ABRO

```
abort
   (await A || await B);
   emit O ;
   halt
when R ;
loop
   abort
      (await A || await B) ;
      emit O ;
      halt
   when R
end
```

$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\}$$

Execution of ABRO

```
abort
   (await A || await B);
    emit O ;
    halt
when R ;
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
```

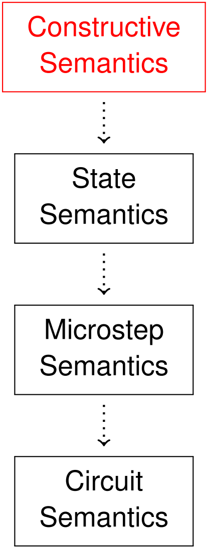
$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\} \Longrightarrow \{A, B,$$

Execution of ABRO

```
abort


   halt
when R ;
loop
   abort
      (await A || await B);
      emit O ;
      halt
   when R
end
```

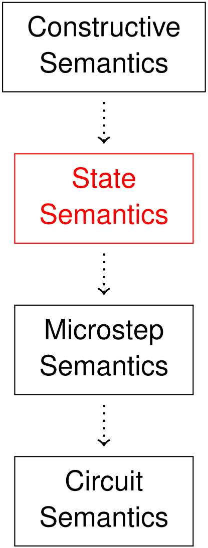
$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\} \Longrightarrow \{A, B, O\}$$

Global diagram of semantics



- $\oplus$ closest to PL semantics
- $\oplus$ one small set of rules
- $\ominus$ modifies the program

Global diagram of semantics



- ⊕ closest to PL semantics
- ⊕ one small set of rules
- ⊖ modifies the program

State Semantics

- Evaluation as moving annotations on the source code
  - The underlying program never changes
  - Pointers indicate where the execution is
    $\rightsquigarrow$ several pointers because of parallelism
- Close to circuits:
  activated pause $=$ activated register
- Two types of programs:
  - Inert program p
  - State $\widehat{p}$ = program under evaluation
  - Term $\overline{p}$ = either $\widehat{p}$ or p
- Two sets of rules:
  - Start: program $\rightarrow$ term
  - Resume: state $\rightarrow$ term

Constructive *vs.* State: the if-then Rule

- Constructive Semantics

$$\frac{s^{+} \in E \qquad p \xhookrightarrow[E]{E',k} p'}{s \,?\, p \,,\, q \xhookrightarrow[E]{E',k} p'}$$

- State Semantics
  - Start rule

$$\frac{s^{+} \in E \qquad p \xhookrightarrow[E]{E',k}\!\!\twoheadrightarrow_s \overline{p'}}{s \,?\, p \,,\, q \xhookrightarrow[E]{E',k}\!\!\twoheadrightarrow_s s \,?\, \overline{p'} \,,\, q}$$

  - Resume rule

$$\frac{\widehat{p} \xhookrightarrow[E]{E',k}\!\!\twoheadrightarrow_r \overline{p'}}{s \,?\, \widehat{p} \,,\, q \xhookrightarrow[E]{E',k}\!\!\twoheadrightarrow_r s \,?\, \overline{p'} \,,\, q}$$

ABRO again

Constructive Semantics

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

State Semantics

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

ABRO again

Constructive Semantics

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

$\{B\}$

State Semantics

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

ABRO again

Constructive Semantics

```
abort
   (await A || nothing);
    emit O ;
    halt
when R ;
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
              {B}
```

State Semantics

```
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
```

ABRO again

Constructive Semantics

```
abort
   (await A || nothing);
    emit O ;
    halt
when R ;
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
```

$$\{B\} \Longrightarrow \{A, O\}$$

State Semantics

$\mathtt{loop}$

$\quad\mathtt{abort}$

$\qquad(\widehat{\mathtt{await}}\ A\ \ ||\ \ \mathtt{await}\ B)\ ;$

$\qquad\mathtt{emit}\ O\ ;$

$\qquad\mathtt{halt}$

$\quad\widehat{\mathtt{when}}\ R$

$\mathtt{end}$

ABRO again

Constructive Semantics

```
abort



    halt
when R ;
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
 end
```

$$\{B\} \Longrightarrow \{A, O\}$$

State Semantics

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
 end
```

ABRO again

Constructive Semantics

```
abort


   halt
when R ;
loop
   abort
      (await A || await B) ;
      emit O ;
      halt
   when R
end
```

$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\}$$

State Semantics

```
loop
   abort
      (await A || await B) ;
      emit O ;
      halt
   when R
end
```

ABRO again

Constructive Semantics

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

State Semantics

```
loop
    abort
        (await A || await B);
        emit O;
        halt
    when R
end
```

$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\}$$

ABRO again

Constructive Semantics

```
abort
   (await A || await B);
    emit O ;
    halt
when R ;
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
```

State Semantics

$$
\begin{array}{l}
\texttt{loop} \\
\quad \texttt{abort} \\
\quad\quad (\widehat{\texttt{await}}\ A\ \texttt{||}\ \widehat{\texttt{await}}\ B)\,; \\
\quad\quad \texttt{emit}\ O\,; \\
\quad\quad \texttt{halt} \\
\quad \widehat{\texttt{when}}\ R \\
\texttt{end}
\end{array}
$$

$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\}$$

ABRO again

Constructive Semantics

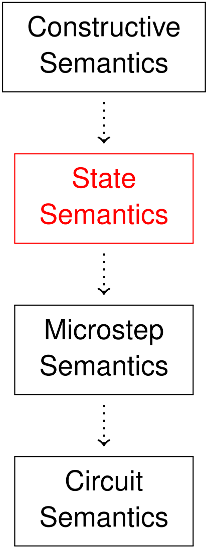
```
abort


    halt
when R ;
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
```

State Semantics

```
loop
    abort
        (await A || await B) ;
        emit O ;
        halt
    when R
end
```

$$\{B\} \Longrightarrow \{A, O\} \Longrightarrow \{B\} \Longrightarrow \{R\} \Longrightarrow \{A, B, O\}$$

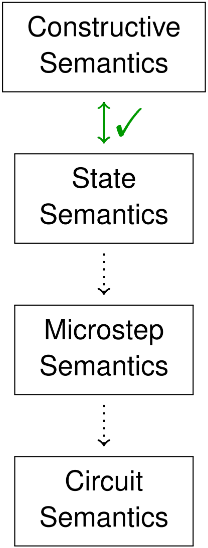
Global diagram of semantics



- $\oplus$ closest to PL semantics
- $\oplus$ one small set of rules
- $\ominus$ modifies the program

- $\oplus$ execution as annotations
- $\oplus$ correspondance with circuit states
- $\ominus$ two sets of rules

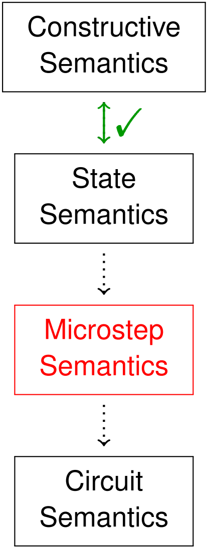
Global diagram of semantics



- $\oplus$ closest to PL semantics
- $\oplus$ one small set of rules
- $\ominus$ modifies the program

- $\oplus$ execution as annotations
- $\oplus$ correspondance with circuit states
- $\ominus$ two sets of rules

Global diagram of semantics



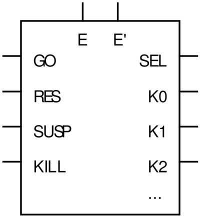
- ⊕ closest to PL semantics
- ⊕ one small set of rules
- ⊖ modifies the program

- ⊕ execution as annotations
- ⊕ correspondance with circuit states
- ⊖ two sets of rules

Microstep Semantics: Entering the Instant

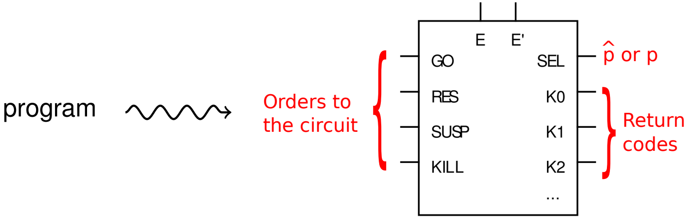
- Key idea:
  - Atomic steps on source code
    that match electric propagation through gates
  - No more cheating with Must/Can!
- Inspiration:
  - Fixpoint semantics: increase the information
  - Circuit translation

program



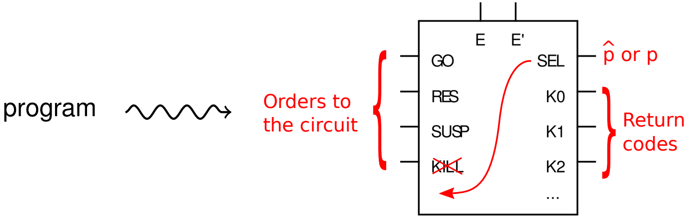
Microstep Semantics: Entering the Instant

- Key idea:
  - Atomic steps on source code
    that match electric propagation through gates
  - No more cheating with Must/Can!
- Inspiration:
  - Fixpoint semantics: increase the information
  - Circuit translation



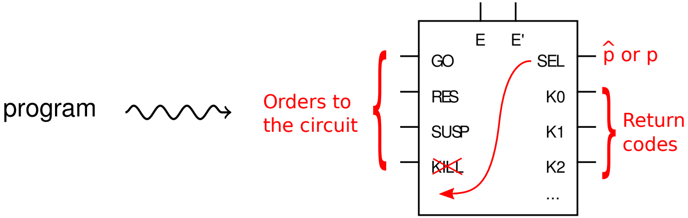
Microstep Semantics: Entering the Instant

- Key idea:
  - Atomic steps on source code
    that match electric propagation through gates
  - No more cheating with Must/Can!
- Inspiration:
  - Fixpoint semantics: increase the information
  - Circuit translation



Microstep Semantics: Entering the Instant

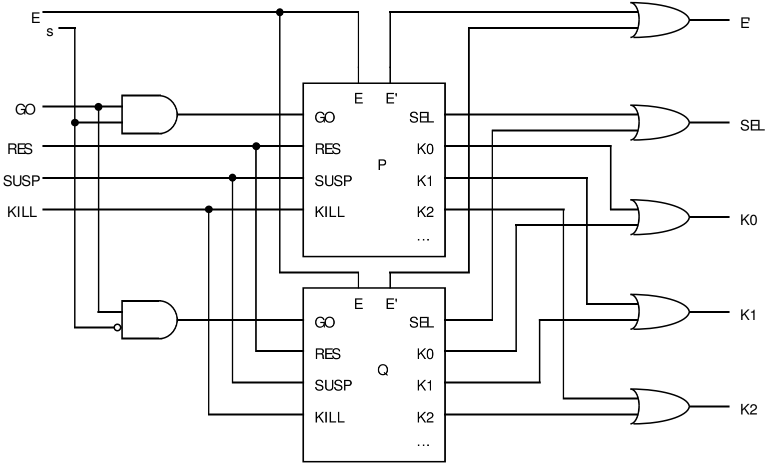
- Key idea:
  - Atomic steps on source code
    that match electric propagation through gates
  - No more cheating with Must/Can!
- Inspiration:
  - Fixpoint semantics: increase the information
  - Circuit translation



- In practice:
  - Recursively add input/output colors to programs
  - Microstep rules update them
  - Until all colors are totally defined

Translation of $s \mathbin{?} p \, , \; q$

`if` s `then` P `else` Q `end`



Microsteps Rules for if-then

$\square$ = input $\qquad\qquad \circ$ = output

$$\frac{s^b \in E \qquad (\mathrm{Go}\,\square) < (\mathrm{Go}\,\square) \wedge b \qquad \square = \square[\mathrm{Go} \leftarrow (\mathrm{Go}\,\square) \wedge b]}{\square(s\,?\,(\square p \circ)\,,\,(\square q \circ))\circ \xrightarrow[E]{E',k} \square(s\,?\,(\square p \circ)\,,\,(\square q \circ))\circ}$$

$$\frac{\square p \circ \xrightarrow[E]{E',k} \square p' \circ}{\square(s\,?\,(\square p \circ)\,,\,(\square q \circ))\circ \xrightarrow[E]{E',k} \square(s\,?\,(\square p' \circ)\,,\,(\square q \circ))\circ}$$

$$\frac{\circ < (\circ \vee \circ)}{\square(s\,?\,(\square p \circ)\,,\,(\square q \circ))\circ \xrightarrow[E]{E',k} \square(s\,?\,(\square p \circ)\,,\,(\square q \circ))(\circ \vee \circ)}$$

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (     (□ await A ○{0,1})
                ||
                    (□ await B ○{0,1})
            )○{0,1} ;
           □ ( (□ emit O ○{0}) ;
               (□ halt ○{0})
            ) ○{0}
          } ○{0,1}
      when R ○{0,1}
  end ○{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (     (□ await A ∘{0,1})
                  ||
                    (□ await B ∘{0,1})
              )∘{0,1} ;
            □ ( (□ emit O ∘{0}) ;
                (□ halt ∘{0})
              ) ∘{0}
          } ∘{0,1}
      when R ∘{0,1}
  end ∘{1}
```

{ B }

First Instant of ABRO in Microsteps

$$
\begin{array}{l}
\square\ \texttt{loop} \\
\quad \square\ \texttt{abort} \\
\quad\quad \square \left\{ \begin{array}{l}
\square\ (\quad (\square\ \texttt{await}\ A\ \circ_{\{0,1\}}) \\
\qquad\quad || \\
\qquad\quad (\square\ \texttt{await}\ B\ \circ_{\{0,1\}}) \\
\quad)\circ_{\{0,1\}}\ ; \\
\square\ (\ (\square\ \texttt{emit}\ O\ \circ_{\{0\}})\ ; \\
\quad (\square\ \texttt{halt}\ \circ_{\{0\}}) \\
\quad)\ \circ_{\{0\}}
\end{array} \right\} \circ_{\{0,1\}} \\
\quad\ \ \texttt{when}\ R\ \circ_{\{0,1\}} \\
\ \ \texttt{end}\ \circ_{\{1\}}
\end{array}
$$

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (     (□ await A ∘{1})
                ||
                    (□ await B ∘{0,1})
            )∘{0,1} ;
            □ ( (□ emit O ∘{0}) ;
                (□ halt ∘{0})
            ) ∘{0}
          } ∘{0,1}
    when R ∘{0,1}
  end ∘{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (    (□ await A ∘{1})
                ||
                   (□ await B ∘{0,1})
              )∘{0,1} ;
            □ ( (□ emit O ∘{0}) ;
                (□ halt ∘{0})
              ) ∘{0}
          } ∘{0,1}
      when R ∘{0,1}
  end ∘{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (     (□ await A ∘{1})
                ||
                    (□ await B ∘{0,1})
            )∘{0,1} ;
            □ ( (□ emit O ∘{0}) ;
                (□ halt ∘{0})
            ) ∘{0}
        } ∘{0,1}
    when R ∘{0,1}
end ∘{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (    (□ await A ●_1)
                ||
                   (□ await B ○_{0,1})
              )○_{0,1} ;
            □ ( (□ emit O ○_{0}) ;
                (□ halt ○_{0})
              ) ○_{0}
          } ○_{0,1}
      when R ○_{0,1}
  end ○_{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (    (□ await A ●₁)
                  ||
                   (□ await B ○{0,1})
              )○{1} ;
              □ ( (□ emit O ○{0}) ;
                  (□ halt ○{0})
              ) ○{0}
          } ○{0,1}
      when R ○{0,1}
  end ○{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (     (□ await A ●_1)
                 ||
                    (□ await B ○_{0,1})
              )○_{1} ;
            □ ( (□ emit O ○_{0}) ;
                (□ halt ○_{0})
              ) ○_{0}
          } ○_{0,1}
      when R ○_{0,1}
  end ○_{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (     (□ await A ●₁)
                 ||
                    (□ await B ○{0,1})
              )○{1} ;
            □ ( (□ emit O ○{0}) ;
                (□ halt ○{0})
              ) ○{0}
          } ○{0,1}
      when R ○{0,1}
  end ○{1}
```

{ B }

First Instant of ABRO in Microsteps

$\square$ `loop`
  $\square$ `abort`
    $\square$ $\{$ $\square$ `(`   `(`$\square$ `await` A $\bullet_1$`)`
        `||`
          `(`$\square$ `await` B $\circ_{\{0,1\}}$`)`
      `)`$\circ_{\{1\}}$ `;`
      $\square$ `( (`$\square$ `emit` O $\circ_{\emptyset}$`);`
        `(`$\square$ `halt` $\circ_{\{0\}}$`)`
      `)` $\circ_{\{0\}}$
    $\}$ $\circ_{\{0,1\}}$
  `when` R $\circ_{\{0,1\}}$
`end` $\circ_{\{1\}}$

{ B }

First Instant of ABRO in Microsteps

$\square$ `loop`

  $\square$ `abort`

    $\square \{$ $\square$ (($\square$ `await` A $\bullet_1$)

          `||`

          ($\square$ `await` B $\circ_{\{0,1\}}$)

)$\circ_{\{1\}}$ `;`

       $\square$ (($\square$ `emit` O $\circ_{\varnothing}$) `;`

          ($\square$ `halt` $\circ_{\{0\}}$)

) $\circ_{\{0\}}$

    $\}$ $\circ_{\{0,1\}}$

  `when` R $\circ_{\{0,1\}}$

`end` $\circ_{\{1\}}$

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (    (□ await A ●₁)
                 ||
                   (□ await B ○{0,1})
              )○{1} ;
              □ ( (□ emit O ○∅) ;
                  (□ halt ○∅)
              ) ○{0}
          } ○{0,1}
      when R ○{0,1}
  end ○{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (    (□ await A •_1)
                  ||
                   (□ await B ∘_{0,1})
              )∘_{1} ;
            □ ( (□ emit O ∘_∅) ;
                (□ halt ∘_∅)
              ) ∘_∅
          } ∘_{0,1}
    when R ∘_{0,1}
end ∘_{1}
```

{ B }

First Instant of ABRO in Microsteps

```
□ loop
    □ abort
        □ { □ (    (□ await A •₁)
                 ||
                   (□ await B ○{0,1})
              )○{1} ;
            □ ( (□ emit O ○∅);
                (□ halt ○∅)
              ) ○∅
          } ○{1}
      when R ○{0,1}
  end ○{1}
```

{ B }

First Instant of ABRO in Microsteps

$$
\begin{array}{l}
\square\ \texttt{loop} \\
\quad \square\ \texttt{abort} \\
\quad\quad \square \left\{ \begin{array}{l}
\square\ (\quad (\square\ \texttt{await}\ A\ \bullet_1) \\
\quad\quad\quad \texttt{||} \\
\quad\quad\quad (\square\ \texttt{await}\ B\ \circ_{\{0,1\}}) \\
\quad)\circ_{\{1\}}\ \texttt{;} \\
\quad \square\ (\ (\square\ \texttt{emit}\ O\ \circ_{\varnothing})\ \texttt{;} \\
\quad\quad (\square\ \texttt{halt}\ \circ_{\varnothing}) \\
\quad)\ \circ_{\varnothing}
\end{array} \right\} \circ_{\{1\}} \\
\quad \texttt{when}\ R\ \circ_{\{1\}} \\
\texttt{end}\ \circ_{\{1\}}
\end{array}
$$

{ B }

First Instant of ABRO in Microsteps

$$
\begin{array}{l}
\square\ \texttt{loop} \\
\quad \square\ \texttt{abort} \\
\quad\quad \square \left\{ \begin{array}{l}
\square\ (\quad (\square\ \texttt{await}\ A\ \bullet_1) \\
\qquad\quad \texttt{||} \\
\qquad\quad (\square\ \texttt{await}\ B\ \bullet_0) \\
\quad)\circ_{\{1\}}\ ; \\
\square\ (\ (\square\ \texttt{emit}\ O\ \circ_{\varnothing})\ ; \\
\quad\ (\square\ \texttt{halt}\ \circ_{\varnothing}) \\
\quad)\ \circ_{\varnothing}
\end{array} \right\} \circ_{\{1\}} \\
\quad \texttt{when}\ R\ \circ_{\{1\}} \\
\texttt{end}\ \circ_{\{1\}}
\end{array}
$$

{ B }

First Instant of ABRO in Microsteps

□ loop

□ abort

□ { □ ((□ await A $\bullet_1$)

||

(□ await B $\bullet_0$)

)$\bullet_1$;

□ ((□ emit O $\circ_{\emptyset}$) ;

(□ halt $\circ_{\emptyset}$)

) $\circ_{\emptyset}$

} $\circ_{\{1\}}$

when R $\circ_{\{1\}}$

end $\circ_{\{1\}}$

{ B }

$$
\begin{array}{l}
\square\ \texttt{loop} \\
\quad \square\ \texttt{abort} \\
\qquad \square \left\{ \begin{array}{l}
\square\ (\quad (\square\ \texttt{await}\ A\ \bullet_1) \\
\qquad\quad \texttt{||} \\
\qquad\quad (\square\ \texttt{await}\ B\ \bullet_0) \\
\quad)\bullet_1\ ; \\
\square\ (\ (\square\ \texttt{emit}\ O\ \circ_\emptyset)\ ; \\
\qquad (\square\ \texttt{halt}\ \circ_\emptyset) \\
\quad)\ \circ_\emptyset
\end{array} \right\} \bullet_1 \\
\quad \texttt{when}\ R\ \circ_{\{1\}} \\
\texttt{end}\ \circ_{\{1\}}
\end{array}
$$

{ B }

First Instant of ABRO in Microsteps

$\square$ loop
  $\square$ abort
    $\square$ { $\square$ (($\square$ await A $\bullet_1$)
      ||
      ($\square$ await B $\bullet_0$)
)$\bullet_1$;
    $\square$ (($\square$ emit O $\circ_\varnothing$) ;
      ($\square$ halt $\circ_\varnothing$)
) $\circ_\varnothing$
    } $\bullet_1$
  when R $\bullet_1$
end $\circ_{\{1\}}$

{ B }

First Instant of ABRO in Microsteps

$\square$ `loop`

$\square$ `abort`

$\square$ $\{$ $\square$ (($\square$ `await` A $\bullet_1$)

`||`

($\square$ `await` B $\bullet_0$)

)$\bullet_1$;

$\square$ (($\square$ `emit` O $\circ_\varnothing$) ;

($\square$ `halt` $\circ_\varnothing$)

) $\circ_\varnothing$

$\}$ $\bullet_1$

`when` R $\bullet_1$

`end` $\bullet_1$

{ B }

Global diagram of semantics

Constructive Semantics

State Semantics

Microstep Semantics

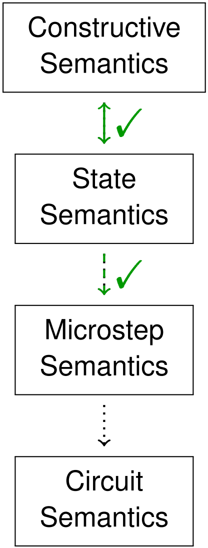
Circuit Semantics

- ⊕ closest to PL semantics
- ⊕ one small set of rules
- ⊖ modifies the program

- ⊕ execution as annotations
- ⊕ correspondance with circuit states
- ⊖ two sets of rules

- ⊕ low-level local semantics
- ⊕ very close to circuits
- ⊕ no Can/Must
- ⊕ one set of rules
- ⊖ a lot of rules
- ⊖ no loop yet

Global diagram of semantics



- ⊕ closest to PL semantics
- ⊕ one small set of rules
- ⊖ modifies the program

- ⊕ execution as annotations
- ⊕ correspondance with circuit states
- ⊖ two sets of rules

- ⊕ low-level local semantics
- ⊕ very close to circuits
- ⊕ no Can/Must
- ⊕ one set of rules
- ⊖ a lot of rules
- ⊖ no loop yet

Current state of the proofs

- Microsteps are still work in progress
  - Non deterministic but confluent
  - Hardest parts by far:
    - Avoiding Can/Must
    - Invariants to approximate valid microsteps executions
  - Having the right design takes time: 1 iteration $\approx$ 1–2 months
- 15 admits left
  - 5: Link Can/Must with microsteps
  - 5: Coinduction with up-to techniques
  - 2: Technical changes (setoid rewriting)
  - 2: A bug with weak suspend?
  - 1: Unused property
- What about reincarnation?
  - = Avoid using twice the same wires/gates with different values
  - Fixpoint semantics already avoid reuse

Conclusion

- Same as last year
  - All important elements are in place
  - No real mistake found yet — weak suspend?
  - Still work to do — who?
  - Fixpoint semantics for other synchronous languages
    $\leadsto$ Can we reuse the same proof ideas?

- Two years of formal proofs with Coq
  - What is easy/hard?
    - Easy part: just formalization to do — constructive $\rightarrow$ state
    - Hard part: good design for formal proofs — state $\rightarrow$ micro
  - Best representation is not obvious
    - Sometimes different from the paper one!
    - Months for each try

Conclusion

- Same as last year
  - All important elements are in place
  - No real mistake found yet — weak suspend?
  - Still work to do — who?
  - Fixpoint semantics for other synchronous languages
    $\rightsquigarrow$ Can we reuse the same proof ideas?

- Two years of formal proofs with Coq
  - What is easy/hard?
    - Easy part: just formalization to do — constructive $\rightarrow$ state
    - Hard part: good design for formal proofs — state $\rightarrow$ micro
  - Best representation is not obvious
    - Sometimes different from the paper one!
    - Months for each try

The End