Modular Compilation of a Synchronous Language

Annie Ressouche[1] and Daniel Gaffé[2] and Valérie Roy[3]

[1]Inria Sophia-Antipolis Méditerranée

[2]Nice Sophia Antipolis University and CNRS(LEAT)

[3]Ecole des Mines-CMA

SERA 2008

LABORATOIRE D'ELECTRONIQUE ANTENNES ET TELECOMMUNICATIONS

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

centre de recherche SOPHIA ANTIPOLIS - MÉDITERRANÉE

Motivation

+ Synchronous languages are model-driven $\Longrightarrow$

- Efficiency and reusability of system design
- Formal verification of system behavior

- Large size of models

Modular compilation

Motivation

+ Synchronous languages are model-driven $\Longrightarrow$
  - Efficiency and reusability of system design
  - Formal verification of system behavior

- Large size of models

Modular compilation

Motivation

+ Synchronous languages are model-driven $\Longrightarrow$
  - Efficiency and reusability of system design
  - Formal verification of system behavior

- Large size of models

Modular compilation

Motivation

+ Synchronous languages are model-driven $\Longrightarrow$
  - Efficiency and reusability of system design
  - Formal verification of system behavior

- Large size of models

➡ Modular compilation

Motivation

+ Synchronous languages are model-driven $\Longrightarrow$

- Efficiency and reusability of system design
- Formal verification of system behavior

- Large size of models

➡ Modular compilation

model-driven + modularity $\Longrightarrow$ global causality checking

- synchronous hypothesis $\Longrightarrow$ responsiveness.
- modularity
- global causality checking

Motivation

+ Synchronous languages are model-driven $\Longrightarrow$
  - Efficiency and reusability of system design
  - Formal verification of system behavior

- Large size of models

➡ Modular compilation

We introduce :

- an **equational semantic** allowing modular compilation
- an efficient way to check causality
- a synchronous language LE

Outline

LE Language

LE language allows 3 kinds of design :

1. Event driven application design
    - synchronous parallel
    - Run module operator to achieve separated compilation
2. Automata (State Chart like) design
3. Data flow application design

LE Language

LE language allows 3 kinds of design :

1. Event driven application design
    - synchronous parallel
    - Run module operator to achieve separated compilation
2. Automata (State Chart like) design
3. Data flow application design

LE Language

LE language allows 3 kinds of design :

1. Event driven application design
   - synchronous parallel
   - Run module operator to achieve separated compilation
2. Automata (State Chart like) design
3. Data flow application design

LE Language

LE language allows 3 kinds of design :

1. Event driven application design
    - synchronous parallel
    - Run module operator to achieve separated compilation
2. Automata (State Chart like) design
3. Data flow application design

Mathematical Context

- $\xi = \{\perp, 1, 0, \top\}$;
- notion of environment (E, $\preceq$)

Mathematical Context

- $\xi = \{\bot, 1, 0, \top\}$;
- notion of environment (E, $\preceq$)

Notion of Circuit

- W : wires ; R : registers ; S : signals (input, output, locals)
- $\mathcal{C} =_{def} \xi$ equation system
- $p \longrightarrow \mathcal{C}(p)$ with 3 wires :
  1. Set_p : control flow propagation
  2. $Reset_p$: reinitialisation flow propagation
  3. RTL_p : ready to leave
  4. $ACTIVE$: register (for some instruction only)
- $E \vdash w \hookrightarrow bb$: a constructive propagation law

Mathematical Context

- $\xi = \{\bot, 1, 0, \top\}$;
- notion of environment (E, $\preceq$)

Notion of Circuit

- W : wires ; R : registers ; S : signals (input, output, locals)
- $\mathcal{C} =_{def} \xi$ equation system
- $p \longrightarrow \mathcal{C}(p)$ with 3 wires :
  1. Set_p : control flow propagation
  2. $Reset_p$: reinitialisation flow propagation
  3. RTL_p : ready to leave
  4. $ACTIVE$: register (for some instruction only)
- $E \vdash w \hookrightarrow bb$: a constructive propagation law

Mathematical Context

- $\xi = \{\bot, 1, 0, \top\}$;
- notion of environment (E, $\preceq$)

Notion of Circuit

- W : wires ; R : registers ; S : signals (input, output, locals)
- $\mathcal{C} =_{def} \xi$ equation system
- $p \longrightarrow \mathcal{C}(p)$ with 3 wires :
  1. Set_p : control flow propagation
  2. $Reset_p$: reinitialisation flow propagation
  3. RTL_p : ready to leave
  4. $ACTIVE$: register (for some instruction only)
- $E \vdash w \hookrightarrow bb$: a constructive propagation law

Mathematical Context

- $\xi = \{\bot, 1, 0, \top\}$;
- notion of environment (E, $\preceq$)

Notion of Circuit

- W : wires; R : registers; S : signals (input, output, locals)
- $\mathcal{C} =_{def} \xi$ equation system
- $p \longrightarrow \mathcal{C}(p)$ with 3 wires :
  1. Set_p : control flow propagation
  2. $Reset_p$: reinitialisation flow propagation
  3. RTL_p : ready to leave
  4. $ACTIVE$: register (for some instruction only)
- $E \vdash w \hookrightarrow bb$: a constructive propagation law

Equational Semantic Definition

- p a LE statement, E : an environment
  $\mathcal{S}_e(p, E) = E'$ iff $E \vdash \mathcal{C}(p) \hookrightarrow E'$. (notation : $\langle p \rangle_E$)
- P :LE program.
  $(P, E) \longmapsto E'$ iff $\mathcal{S}_e(\Gamma(P), E) = E'$, where $\Gamma(P)$ is the LE statement body of program P

Equational Semantic Definition

- p a LE statement, E : an environment
  $\mathcal{S}_e(p, E) = E'$ iff $E \vdash \mathcal{C}(p) \hookrightarrow E'$. (notation : $\langle p \rangle_E$)
- P :LE program.
  $(P, E) \longmapsto E'$ iff $\mathcal{S}_e(\Gamma(P), E) = E'$, where $\Gamma(P)$ is the LE statement body of program P

Parallel Operator($P_1 \| P_2$) Circuit Definition

$$\mathcal{C}(P_1 \| P_2) = \mathcal{C}(P_1) \cup \mathcal{C}(P_2) \cup \mathcal{C}_{P_1 \| P_2}$$

$$\mathcal{C}_{P_1 \| P_2} = \left\{ \begin{array}{ll} \mathrm{Set}_{P_1} = & \mathrm{Set}_{P_1 \| P_2} \\ \mathrm{Set}_{P_2} = & \mathrm{Set}_{\mathrm{P}_1 \| \mathrm{P}_2} \\ \mathrm{Reset}_{P_1} = & \mathrm{Reset}_{P_1 \| P_2} \\ \mathrm{Reset}_{P_2} = & \mathrm{Reset}_{P_1 \| P_2} \\ \mathrm{ACTIVE}_1{}^+ = & (\mathrm{RTL}_{P_1} \sqcup \mathrm{ACTIVE}_1) \sqcap \neg \mathrm{Reset}_{P_1 \| P_2} \\ \mathrm{ACTIVE}_2{}^+ = & (\mathrm{RTL}_{P_2} \sqcup \mathrm{ACTIVE}_2) \sqcap \neg \mathrm{Reset}_{P_1 \| P_2} \\ \mathrm{RTL}_{P_1 \| P_2} = & (\mathrm{RTL}_{P_1} \sqcup \mathrm{ACTIVE}_1) \sqcap (\mathrm{RTL}_{P_2} \sqcup \mathrm{ACTIVE}_2) \end{array} \right\}$$

Parallel Operator($P_1 \| P_2$) Semantic Computation

$$\langle P_1 \rangle_E \sqcup \langle P_2 \rangle_E \vdash \mathcal{C}(P_1 \| P_2) \hookrightarrow \langle P_1 \| P_2 \rangle_E$$

Behavioral Semantic

P program, E input environment, E' output environment :

Rule-based specification : $p \xrightarrow[E]{E', TERM} p'$

$$(P, E) \longmapsto (P', E') \quad \text{iff} \quad \Gamma(P) \xrightarrow[E]{E',\ TERM} \Gamma(P')$$

Behavioral Semantic

P program, E input environment, E' output environment :

Rule-based specification : $p \xrightarrow[E]{E', TERM} p'$

$$(P, E) \longmapsto (P', E') \quad \text{iff} \quad \Gamma(P) \xrightarrow[E]{E',\ TERM} \Gamma(P')$$

Theorem

Let P be a LE statement, O its output signal set, and $E_{\mathcal{C}}$ an input environment, the following property holds :

$P \xrightarrow[E]{E', \mathrm{RTL}_P} P'$ *and* $\langle P \rangle_{E_{\mathcal{C}}}|_O = E'|_O$

where $E = \{S^x | S^x \in E_{\mathcal{C}}$ *and* $S \notin W \cup R\}$.

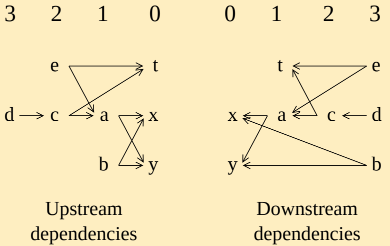
- Equational semantic offers a means to compile LE programs.
- Behavioral semantic ensures model-checking techniques apply.

Causality Checking

- Problem : the composition of 2 causal systems may introduce causality cycle
- Solution : preserve signal independance

Sorting Algorithm : a PERT family

$$
\begin{aligned}
a &= x \sqcup y \\
b &= x \sqcup not\ y \\
c &= a \sqcup t \\
d &= a \sqcup c \\
e &= a \sqcup t
\end{aligned}
$$



CanDate and MustDate

a	b	c	d	e	x	y	t
$(1,1)$	$(1,3)$	$(2,2)$	$(3,3)$	$(2,3)$	$(0,0)$	$(0,0)$	$(0,1)$

Partial Orders Link

A	B
$a = x \sqcup y$	
$b = x \sqcup not\ y$	$y = m$
$c = a \sqcup t$	$z = a$
$d = a \sqcup c$	$v = w$
$e = a \sqcup t$	

A :	a	b	c	d	e	x	y	t
	$(1,1)$	$(1,3)$	$(2,2)$	$(3,3)$	$(2,3)$	$(0,0)$	$(0,0)$	$(0,1)$

B :	a	m	v	w	y	z
	$(0,0)$	$(0,0)$	$(1,1)$	$(0,0)$	$(1,1)$	$(1,1)$

Common variables : a y

A :	a	b	c	d	e	x	y	t
	$(1,1)$	$(1,3)$	$(2,2)$	$(3,3)$	$(2,3)$	$(0,0)$	$(0,0)$	$(0,1)$

B :	a	m	v	w	y	z
	$(0,0)$	$(0,0)$	$(1,1)$	$(0,0)$	$(1,1)$	$(1,1)$

Dates Propagation

	a	b	c	d	e	x	y
$\Delta_c(a)$:	$(1,1)$	$(1,3)$	$(2,2)$	$(3,3)$	$(2,3)$	$(0,0)$	$(1,1)$
$\Delta_m(a)$:	$(1,1)$	$(1,3)$	$(2,2)$	$(3,3)$	$(2,3)$	$(0,0)$	$(1,1)$
$\Delta_c(y)$:	$(2,2)$	$(2,4)$	$(3,3)$	$(4,4)$	$(3,4)$	$(0,0)$	$(1,1)$
$\Delta_m(y)$:	$(2,2)$	$(2,4)$	$(3,3)$	$(4,4)$	$(3,4)$	$(0,0)$	$(1,1)$

	t	m	v	w	z
$\Delta_c(a)$:	$(0,1)$	$(0,0)$	$(1,1)$	$(0,0)$	$(2,2)$
$\Delta_m(a)$:	$(0,1)$	$(0,0)$	$(1,1)$	$(0,0)$	$(2,2)$
$\Delta_c(y)$:	$(0,1)$	$(0,0)$	$(1,1)$	$(0,0)$	$(3,3)$
$\Delta_m(y)$:	$(0,1)$	$(0,0)$	$(1,1)$	$(0,0)$	$(3,3)$

a	b	c	d	e	x	y	t	m	v	w
$(2,2)$	$(2,4)$	$(3,3)$	$(4,4)$	$(3,4)$	$(0,0)$	$(1,1)$	$(0,1)$	$(0,0)$	$(1,1)$	$(0,0)$

Two Valid Sorts

sorting − 1	sorting − 2
$0: \; m \quad x \quad v \quad t$	$0: \; m \quad x \quad v \quad t$
$1: \; y = m$	$1: \; y = m$
$v = w$	$v = w$
$2: \; b = x \sqcup not\ y$	$2: \; a = x \sqcup y$
$a = x \sqcup y$	$3: \; c = a \sqcup t$
$3: \; c = a \sqcup t$	$z = a$
$z = a$	$4: \; b = x \sqcup not\ y$
$e = a \sqcup t$	$e = a \sqcup t$
$4: \; d = a \sqcup c$	$d = a \sqcup c$

Effective Compilation

1. P is associated with a ξ equation system ($\mathcal{C}(P)$)
2. $\xi \longrightarrow \mathcal{B}$ (BDD implementation)
3. compilation = $\hookrightarrow$ propagation law implementation
4. separated compilation relies on LEC internal format

Effective Compilation

1. P is associated with a ξ equation system ($\mathcal{C}(P)$)
2. $\xi \longrightarrow \mathcal{B}$ (BDD implementation)
3. compilation $= \hookrightarrow$ propagation law implementation
4. separated compilation relies on LEC internal format

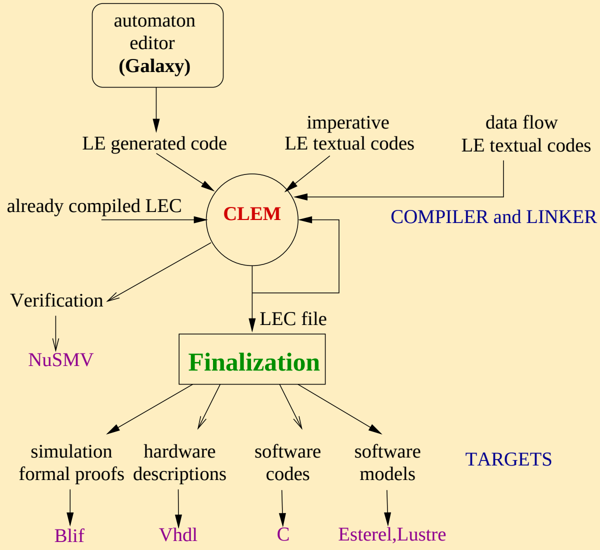
Effective Compilation

1. P is associated with a ξ equation system ($\mathcal{C}(P)$)
2. $\xi \longrightarrow \mathcal{B}$ (BDD implementation)
3. compilation $= \hookrightarrow$ propagation law implementation
4. separated compilation relies on LEC internal format

Effective Compilation

1. P is associated with a ξ equation system ($\mathcal{C}(P)$)
2. $\xi \longrightarrow \mathcal{B}$ (BDD implementation)
3. compilation $= \hookrightarrow$ propagation law implementation
4. separated compilation relies on LEC internal format

CLEM Toolkit ://http :www.inria.fr/sophia/pulsar/projects/Clem



Conclusion

1. LE language with 2 semantics :
   - the equational semantic offers separated compilation means
   - the behavioral semantic allows NuSMV model-checker usage
2. We define the CLEM toolkit around LE language modular compilation

Conclusion

1. LE language with 2 semantics :
   - the equational semantic offers separated compilation means
   - the behavioral semantic allows NuSMV model-checker usage
2. We define the CLEM toolkit around LE language modular compilation

Conclusion

1. LE language with 2 semantics :
   - the equational semantic offers separated compilation means
   - the behavioral semantic allows NuSMV model-checker usage
2. We define the CLEM toolkit around LE language modular compilation

Conclusion

1. LE language with 2 semantics :
   - the equational semantic offers separated compilation means
   - the behavioral semantic allows NuSMV model-checker usage
2. We define the CLEM toolkit around LE language modular compilation

Future Work

1. large industrial application development
2. extension of LE to deal with data :
   - language improvement
   - semantics extension
   - rely on Abstract Interpretation methods (like polyhedron intersection) to still apply model-checking techniques
3. improve LE verification :
   - provide facilities to define safety properties as observers.
   - prove that modular and “assume-guarantee” model-checking techniques apply

Future Work

1. large industrial application development
2. extension of LE to deal with data :
    - language improvement
    - semantics extension
    - rely on Abstract Interpretation methods (like polyhedron intersection) to still apply model-checking techniques
3. improve LE verification :
    - provide facilities to define safety properties as observers.
    - prove that modular and "assume-guarantee" model-checking techniques apply

Future Work

1. large industrial application development
2. extension of LE to deal with data :
   - language improvement
   - semantics extension
   - rely on Abstract Interpretation methods (like polyhedron intersection) to still apply model-checking techniques
3. improve LE verification :
   - provide facilities to define safety properties as observers.
   - prove that modular and "assume-guarantee" model-checking techniques apply

Future Work

1. large industrial application development
2. extension of LE to deal with data :
   - language improvement
   - semantics extension
   - rely on Abstract Interpretation methods (like polyhedron intersection) to still apply model-checking techniques
3. improve LE verification :
   - provide facilities to define safety properties as observers.
   - prove that modular and "assume-guarantee" model-checking techniques apply

$$E \vdash bb \hookrightarrow bb \qquad \frac{E(w) = bb}{E \vdash w \hookrightarrow bb}$$

$$\frac{E \vdash e \hookrightarrow bb}{E \vdash (w = e) \hookrightarrow bb} \qquad \frac{E \vdash e \hookrightarrow \neg bb}{E \vdash \neg e \hookrightarrow bb}$$

$$\frac{E \vdash e \hookrightarrow \top \text{ or } E \vdash e' \hookrightarrow \top}{E \vdash e \sqcup e' \hookrightarrow \top}$$

$$\frac{E \vdash e \hookrightarrow \bot \text{ or } E \vdash e' \hookrightarrow \bot}{E \vdash e \sqcap e' \hookrightarrow \bot}$$

$$\frac{E \vdash e \hookrightarrow 1[0] \text{ and } E \vdash e' \hookrightarrow 0[1]}{E \vdash e \sqcup e' \hookrightarrow \top \text{ and } E \vdash e \sqcap e' \hookrightarrow \bot}$$

$$\frac{E \vdash e \hookrightarrow 1[\bot] \text{ and } E \vdash e' \hookrightarrow \bot[1]}{E \vdash e \sqcup e' \hookrightarrow 1 \; and \; E \vdash e \sqcap e' \hookrightarrow \bot}$$

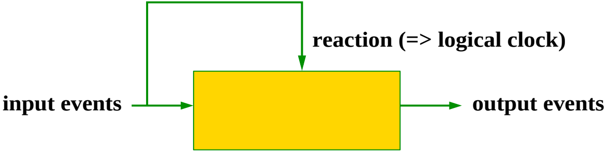
$$\frac{E \vdash e \hookrightarrow 0[\bot] \text{ and } E \vdash e' \hookrightarrow \bot[0]}{E \vdash e \sqcup e' \hookrightarrow 0}$$

$$\frac{E \vdash e \hookrightarrow 0[\top] \text{ and } E \vdash e' \hookrightarrow \top[0]}{E \vdash e \sqcap e' \hookrightarrow 0}$$

$$\frac{E \vdash e \hookrightarrow x \text{ and } E \vdash e' \hookrightarrow x (x = \bot, 0, 1, \top)}{E \vdash e \sqcup e' \hookrightarrow x \text{ and } E \vdash e \sqcap e' \hookrightarrow x}$$

$$\frac{E \vdash e \hookrightarrow 1[T] \text{ and } E \vdash e' \hookrightarrow T[1]}{E \vdash e \sqcap e' \hookrightarrow 1}$$

Synchronous languages rely on the Synchronous hypothesis



Synchronous Hypothesis

Model of event driven systems

- **Broadcasting** of events (non blocking communication)
- Reaction is **atomic** : input and resulting output events are simultaneous
- Succession of reactions $\Longrightarrow$ logical time
- Synchronous systems are deterministic

Event driven Application Design

Event driven Application Design

LE Operators

- *emit speed*
- *present S { P1} else { P2}*
- $P_1 \gg P_2$: perform P_1 then P_2
- $P_1 || P_2$: synchronous parallel : start P_1 and P_2 simultaneously and stop when both have terminated
- *abort P when S* : perform P until S presence
- *loop* $\{P\}$
- *local* S $\{P\}$: encapsulation, the scope of S is restricted to P
- *Run M* : call of module M
- *pause* : stop until the next reaction
- *wait S* : stop until the next reaction in which S is present

LE Program Example

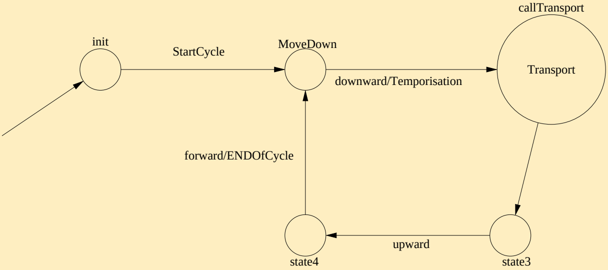
```
module R2WIEO :
Input: I0,I1;
Output: O0,O1;
Run:"/home/ar/GnuStrl/CLEM_SRC/TEST/" : WIEO;
{
  run WIEO[I0 \ i, O0 \ o] || run WIEO[I1 \ i, O1 \ o]
}
end

module WIEO :
Input: i;
Output: o;
wait i >> emit o
end
```

State Chart like Design

State Chart like Design

Automata Design

- $\mathcal{A}(\mathcal{M}, \mathcal{T}, \mathcal{C}ond, M_f, \mathcal{O}, \lambda)$: automata specification



Data flow application Design

Data flow application Design

Equation Design

- $\mathcal{E}(\mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{D})$: equation system definition

```
module ADDMM:
Input: Xi,Yi,Rin;
Output: Si, Rout;

Mealy Machine

Si = (Xi xor Yi) xor Rin;
Rout = (Xi and Yi) or (Xi and Rin) or (Yi and Rin);
end
```

Causality Problem Illustration

```
module first:
Input: I1,I2;
Output: O1,O2;
loop {
 pause >>
 {
 present I1 {emit O1}
 ||
 present I2 {emit O2}
 }
end
```

O1 = I1
O2 = I2

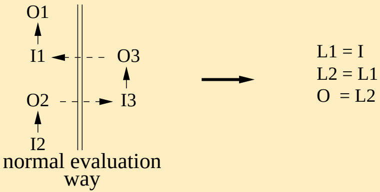
```
module second:
Input: I3;
Output: O3;
loop {
 pause >>  present I3 {emit O3}
}
end
```

O3 = I3

```
module final:
Input: I;
Output O;
local  L1,L2 {
  run first[ L2\I1,O\O1,I\I2,L1\O2]
  ||
  run second[ L1\I3,L2\O3]
}
end
```

O = L2
L1 = I

L2 = L1



Causality Problem Illustration

```
module first:
Input: I1,I2;
Output: O1,O2;
loop {
  pause >>
  {
  present I1 {emit O1}
  ||
  present I2 {emit O2}
  }
end
```

O1 = I1
O2 = I2

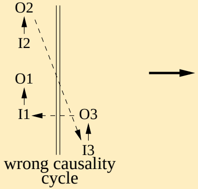
```
module second:
Input: I3;
Output: O3;
loop {
 pause >>  present I3 {emit O3}
}
end
```

O3 = I3

```
module final:
Input: I;
Output O;
local  L1,L2 {
  run first[ L2\I1,O\O1,I\I2,L1\O2]
  ||
  run second[ L1\I3,L2\O3]
}
end
```

O = L2
L1 = I

L2 = L1



L2 = L1
O = L2
L1 = I