Modular Compilation of a Synchronous Language

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

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

[2]Inria Sophia-Antipolis Méditerranée

[3]Ecole des Mines-CMA

Synchron 2010

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

- Large size of models

Modular compilation

Motivation

\+ Synchronous languages are model-driven ⇒ see

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

\- Large size of models

Modular compilation

Motivation

\+ Synchronous languages are model-driven ⇒ see

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

\- Large size of models

Modular compilation

Motivation

\+ Synchronous languages are model-driven ⇒ see

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

\- Large size of models

➡ Modular compilation

Motivation

\+ Synchronous languages are model-driven ⇒ ▸ see

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

\- Large size of models

➡ Modular compilation

model-driven + modularity ⇒ global causality checking

- synchronous hypothesis ⇒ responsiveness.
- modularity
- global causality checking

Motivation

+ Synchronous languages are model-driven ⇒ see
  - Efficiency and reusability of system design
  - Formal verification of system behavior

- Large size of models

➜ Modular compilation

We introduce :

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

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

▸ detail

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

detail

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

detail

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

▸ detail

Mathematical Context

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

Mathematical Context

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

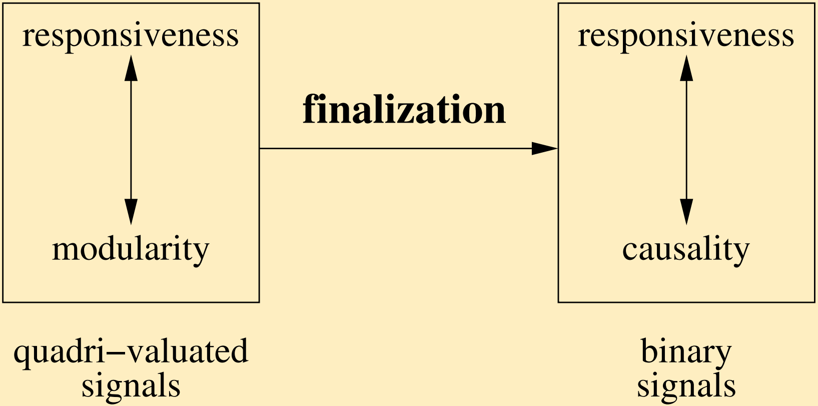
ξ Rules

$\sqcup$	1	0	$\top$	$\bot$
1	1	$\top$	$\top$	1
0	$\top$	0	$\top$	0
$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	1	0	$\top$	$\bot$

$\sqcap$	1	0	$\top$	$\bot$
1	1	$\bot$	1	$\bot$
0	$\bot$	0	0	$\bot$
$\top$	1	0	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

x	$\neg\, x$
1	0
0	1
$\top$	$\bot$
$\bot$	$\top$

Modularity versus Causality



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 : starts p
  2. $Reset_p$: stops and reinits p
  3. RTL_p : p is ready to leave
  4. registers (for some instruction only)
- $E \vdash w \hookrightarrow bb$: a constructive propagation law ▸ prop-law

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 : starts p
  2. $Reset_p$: stops and reinits p
  3. RTL_p : p is ready to leave
  4. registers (for some instruction only)
- $E \vdash w \hookrightarrow bb$: a constructive propagation law prop-law

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 : starts p
  2. $Reset_p$: stops and reinits p
  3. RTL_p : p is ready to leave
  4. registers (for some instruction only)
- $E \vdash w \hookrightarrow bb$: a constructive propagation law prop-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

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

Environment $\mathcal{P}$re Operation

$$\mathcal{P}re(E) = \{S^{\perp} \mid S^x \in E\} \cup \{S^x_{pre} \mid S^x \in E\}$$

Wait operator Circuit Definition

$$\mathcal{C}_{wait\ \ S} = \left[\begin{array}{lcll} R+ & = & (Set_{wait\ \ S} \sqcap \neg Reset_{wait\ \ S}) \sqcup & \\ & & (R \sqcap \neg Reset_{wait\ \ S} \sqcap \neg S) & (1) \\ RTL_{wait\ \ S} & = & R \sqcap S & (2) \end{array}\right]$$

Wait Semantics

$$\langle P_{wait\ \ S} \rangle_E = \mathcal{P}re(E') \text{ and } E \vdash \mathcal{C}(P_{wait\ \ S}) \hookrightarrow E'$$

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

$$\mathcal{C}_{P_1\|P_2} = \left[\begin{array}{lcl} Set_{P_1} & = & Set_{P_1\|P_2} \\ Set_{P_2} & = & Set_{P_1\|P_2} \\ Reset_{P_1} & = & Reset_{P_1\|P_2} \\ Reset_{P_2} & = & Reset_{P_1\|P_2} \\ R_1^+ & = & R_1 \sqcap \neg RTL_{P_2} \sqcap \neg Reset_{P_1\|P_2} \\ & & \sqcup \neg R_2 \sqcap RTL_{P_1} \sqcap \neg RTL_{P_2} \sqcap \neg Reset_{P_1\|P_2} \\ R_2^+ & = & R_2 \sqcap \neg RTL_{P_1} \sqcap \neg Reset_{P_1\|P_2} \\ & & \sqcup \neg R_1 \sqcap \neg RTL_{P_1} \sqcap RTL_{P_2} \sqcap \neg Reset_{P_1\|P_2} \\ RTL_{P_1\|P_2} & = & R_1 \sqcap \neg R_2 \sqcap RTL_{P_2} \sqcup \\ & & (\neg R_1 \sqcap RTL_{P_1} \sqcap (R_2 \sqcup RTL_{P_2})) \end{array}\right]$$

Parallel Semantics

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

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 $E|_X = \{S^x | S^x \in E, S \in X\}$.

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

New Causality Checking Method

- Problem : the composition of 2 causal systems may introduce causality cycle ▸ causality
- Solution :
  1. compute partial orders instead of total orders (thanks to equational semantics)
  2. finalization phase : to generate effective output code

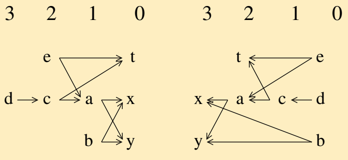
Computing Partial Orders

For each equation system, we compute the earliest and latest dates at which each variable can and must be valuated :

1. 2 dependencies graphs : from system inputs (upstream dependencies graph) and from system outputs (downstream dependencies graph) ;
2. the earliest date of each system variable v is the length of the maximal path from v to system inputs ;
3. the latest date of each system variable v is the length of the maximal path from v to system outputs ;

Example

$$
\begin{aligned}
a &= f_1(x, y) \\
b &= f_2(x, y) \\
c &= f_3(a, t) \\
d &= f_4(a, c) \\
e &= f_5(a, t)
\end{aligned}
$$



Upstream dependencies

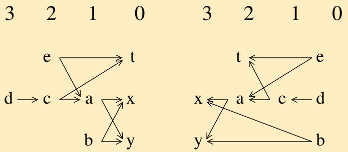
Downstream dependencies

Earliest and Latest Dates

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

Example

$$
\begin{aligned}
a &= f_1(x, y)\\
b &= f_2(x, y)\\
c &= f_3(a, t)\\
d &= f_4(a, c)\\
e &= f_5(a, t)
\end{aligned}
$$



Upstream dependencies

Downstream dependencies

Earliest and Latest Dates

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

3 Algorithms

1. apply PERT method : inputs (resp. outputs) have date 0 and recursively increase of dates for each vertice in the upstream (resp downstream) dependencies graph.
2. apply graph theory :
   - compute the adjency matrix $\mathcal{U}$ of upstream (resp. downstream) dependencies graph.
   - the length of the maximal path from a variable v to system inputs is characterized by the maximal k such that $\mathcal{U}^k[v, i] \neq 0$ for all inputs i.
3. apply fix point theory : the vector of earliest (resp. lastest) dates can be computed as the least fix point of a momotonic increasing function.

Partial Orders Composition

To compose two already sorted systems A and B :

- only interface variables may be common ; thus we memorize the upstream dependencies of output variables and the dowmstream dependencies of input for each equation systems.
- two algorithms :
  1. propagation of commom variables dates adjustement
  2. fix point characterisation starting with the vectors of already computed dates and considering only the variables in the dependencies (upstream and downstream) of common variables

Partial Orders Link

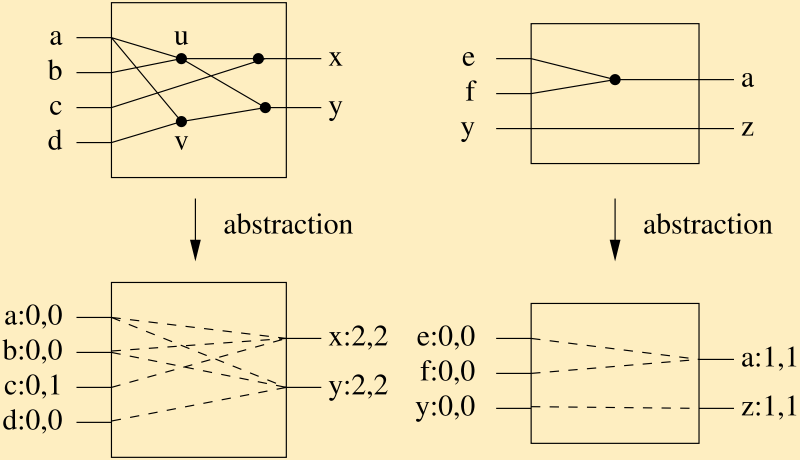
A	B
$a = f_1(x, y)$	
$b = f_2(x, y)$	$y = g_1(m)$
$c = f_3(a, t)$	$z = g_2(d)$
$d = f_4(a, c)$	$v = g_3(w)$
$e = f_5(a, t)$	

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

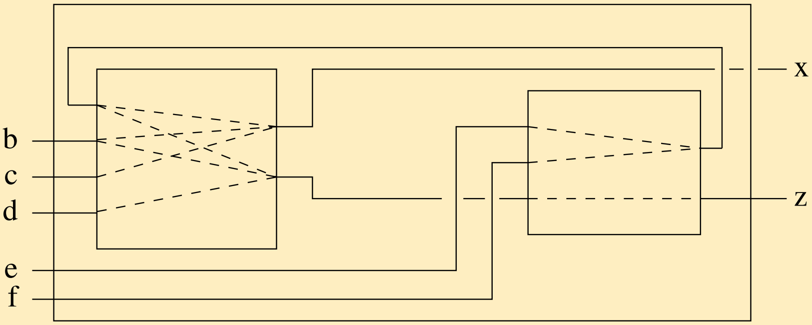
Common variables : d y

	Equations1	*Equations2*	*upstreamdep*	*downstreamdep*	**d**	**y**
a	$(1,2)$	–	$\{c,e,d\}$	$\{x,y\}$	$(1,\mathbf{3})$	$(\mathbf{2},3)$
b	$(1,0)$	–	$\emptyset$	$\{x,y\}$	$(1,0)$	$(\mathbf{2},0)$
c	$(2,1)$	–	$\{d\}$	$\{a,t\}$	$(2,\mathbf{2})$	$(\mathbf{3},2)$
d	$(3,0)$	$(0,1)$	(z)	$\{a,c\}$	$(3,\mathbf{1})$	$(\mathbf{4},1)$
e	$(2,0)$	–	$\emptyset$	$\{a,t\}$	$(2,0)$	$(\mathbf{3},0)$
x	$(0,3)$	–	$\{a,b\}$	$\emptyset$	$(0,\mathbf{4})$	$(0,4)$
y	$(0,3)$	$(1,0)$	$\{a,b\}$	$\{m\}$	$(0,\mathbf{4})$	$(\mathbf{1},4)$
t	$(0,2)$	–	$\{c,e\}$	$\emptyset$	$(0,\mathbf{3})$	$(0,3)$
m	–	$(0,1)$	$\{y\}$	$\emptyset$	$(0,1)$	$(0,\mathbf{5})$
v	–	$(1,0)$	$\emptyset$	$\{w\}$	$(1,0)$	$(1,0)$
w	–	$(0,1)$	$\{v\}$	$\emptyset$	$(0,1)$	$(0,1)$
z	–	$(1,0)$	$\emptyset$	$\{d\}$	$(\mathbf{4},0)$	$(\mathbf{5},0)$

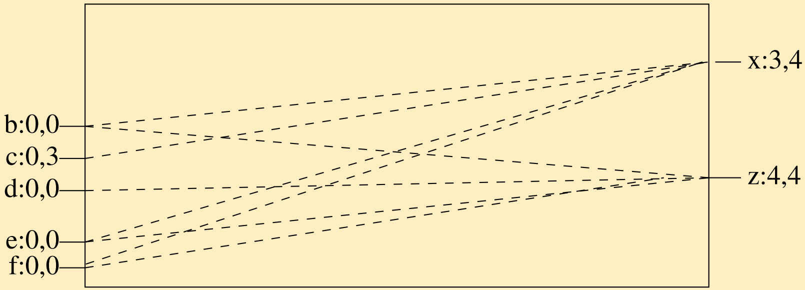
First Compilation Level



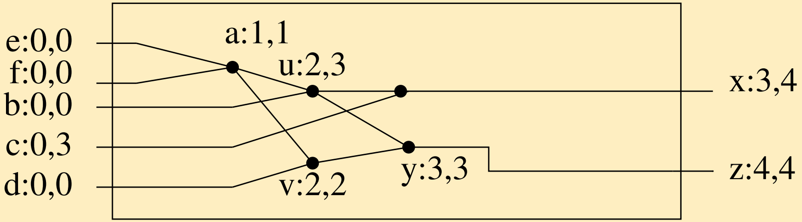
Second Compilation Level



Second Compilation Level



Finalization



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
    1. LEC internal format
    2. Finilization operation

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
    1. LEC internal format
    2. Finilization operation

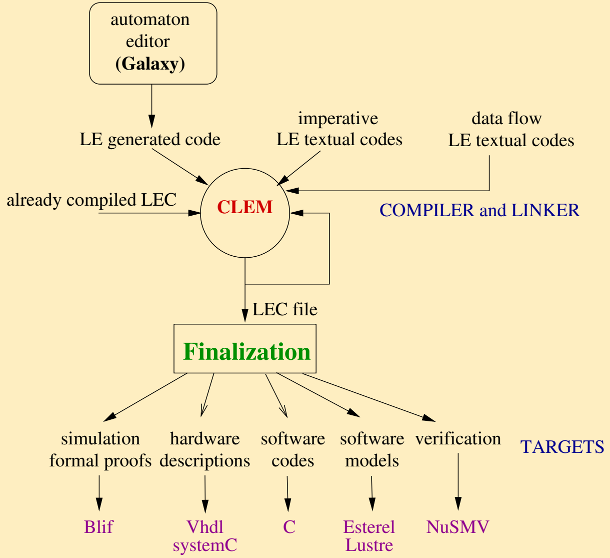
Effective Compilation

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
    1. LEC internal format
    2. Finilization operation

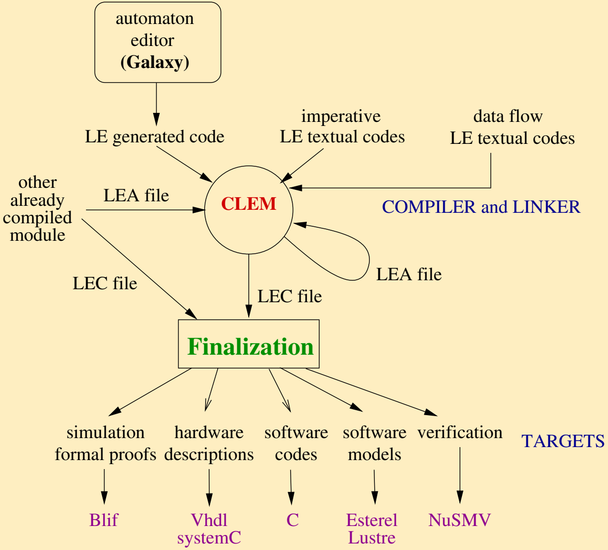
Effective Compilation

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
    1. LEC internal format
    2. Finilization operation

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



The Future CLEM Toolkit



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

Work in Progress

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

Work in Progress

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

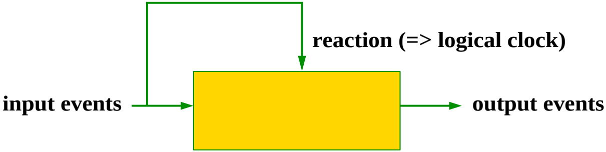
Work in Progress

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

Work in Progress

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

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 $\Rightarrow$ logical time
- Synchronous systems are deterministic

return

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

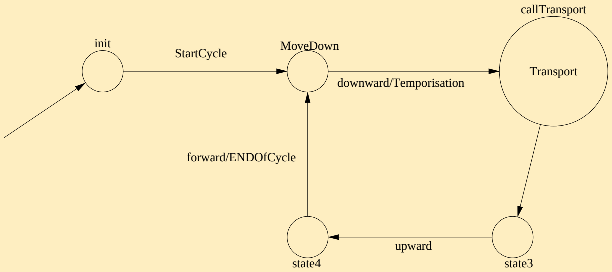
```
module R2WIEO :
Input: I0,I1;
Output: O0,O1;
Run:"/home/ar/GnuStrl/CLEM_SRC/TEST/" : WIEO;
{
  run WIEO[I0 \ i, O0 \ o] || run WIEO[I1 \ i, O1 \ o]
}
end

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

return

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
```

return

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

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

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

$$\frac{(E \vdash e \hookrightarrow 1 \text{ and } E \vdash e' \hookrightarrow \bot) \quad \text{or} \quad (E \vdash e \hookrightarrow \bot \text{ and } E \vdash e' \hookrightarrow 1)}{E \vdash e \sqcup e' \hookrightarrow 1 \text{ and } E \vdash e \sqcap e' \hookrightarrow \bot}$$

$$\frac{(E \vdash e \hookrightarrow 0 \text{ and } E \vdash e' \hookrightarrow \bot) \quad \text{or} \quad (E \vdash e \hookrightarrow \bot \text{ and } E \vdash e' \hookrightarrow 0)}{E \vdash e \sqcup e' \hookrightarrow 0}$$

$$\frac{(E \vdash e \hookrightarrow 0 \text{ and } E \vdash e' \hookrightarrow \top) \quad \text{or} \quad (E \vdash e \hookrightarrow \top \text{ and } E \vdash e' \hookrightarrow 0)}{E \vdash e \sqcap e' \hookrightarrow 0}$$

$$\frac{E \vdash e \hookrightarrow v \text{ and } E \vdash e' \hookrightarrow v}{E \vdash e \sqcup e' \hookrightarrow v \text{ and } E \vdash e \sqcap e' \hookrightarrow v}$$

$$\frac{(E \vdash e \hookrightarrow \top \text{ and } E \vdash e' \hookrightarrow 1) \quad \text{or} \quad (E \vdash e \hookrightarrow 1 \text{ and } E \vdash e' \hookrightarrow \top)}{E \vdash e \sqcap e' \hookrightarrow 1}$$

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

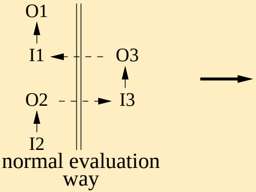
```
module second:
Input: I3;
Output: O3;
loop {
 pause >>  present I3 {emit O3}
}
end
```

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



L1 = I
L2 = L1
O = L2

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

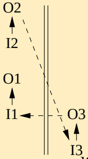
```
module second:
Input: I3;
Output: O3;
loop {
 pause >>  present I3 {emit O3}
}
end
```

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



wrong causality cycle

→

L2 = L1
O = L2
L1 = I

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

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