FM Forum, Oxford

Verification Support for the ELLA Hardware Description Language

Howard Barringer   Graham Gough   <u>Alan Williams</u>

Department of Computer Science, University of Manchester

Brian Monahan, Harlequin Ltd.

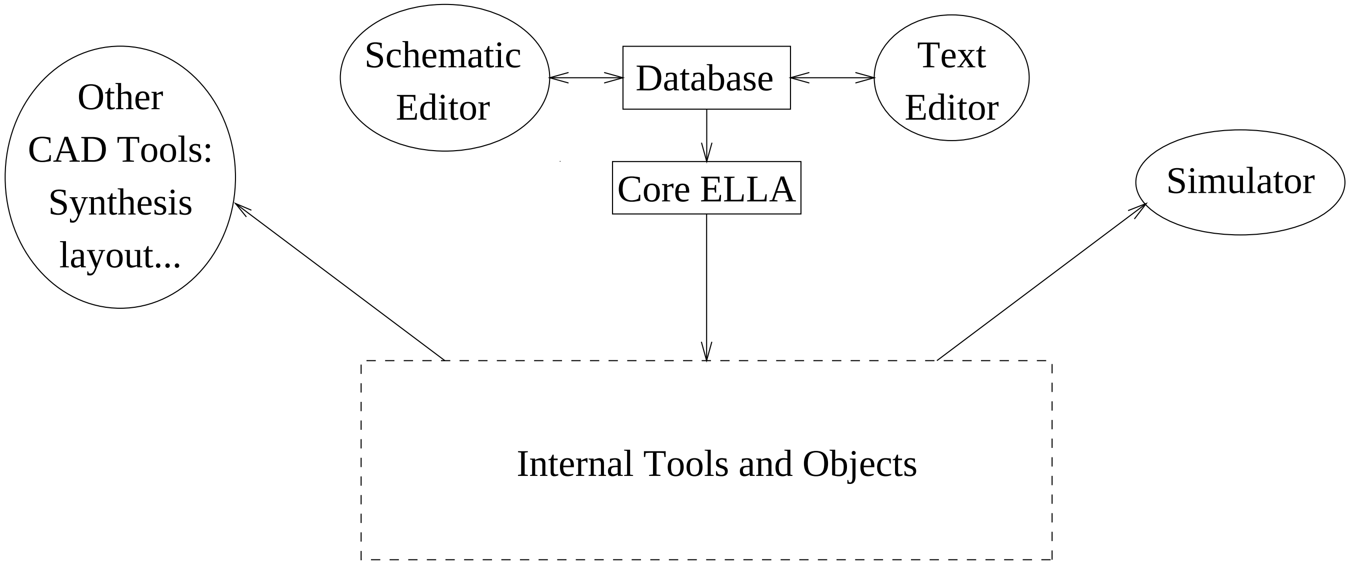
Partners: DRA (Malvern), Harlequin Ltd (Cambridge)

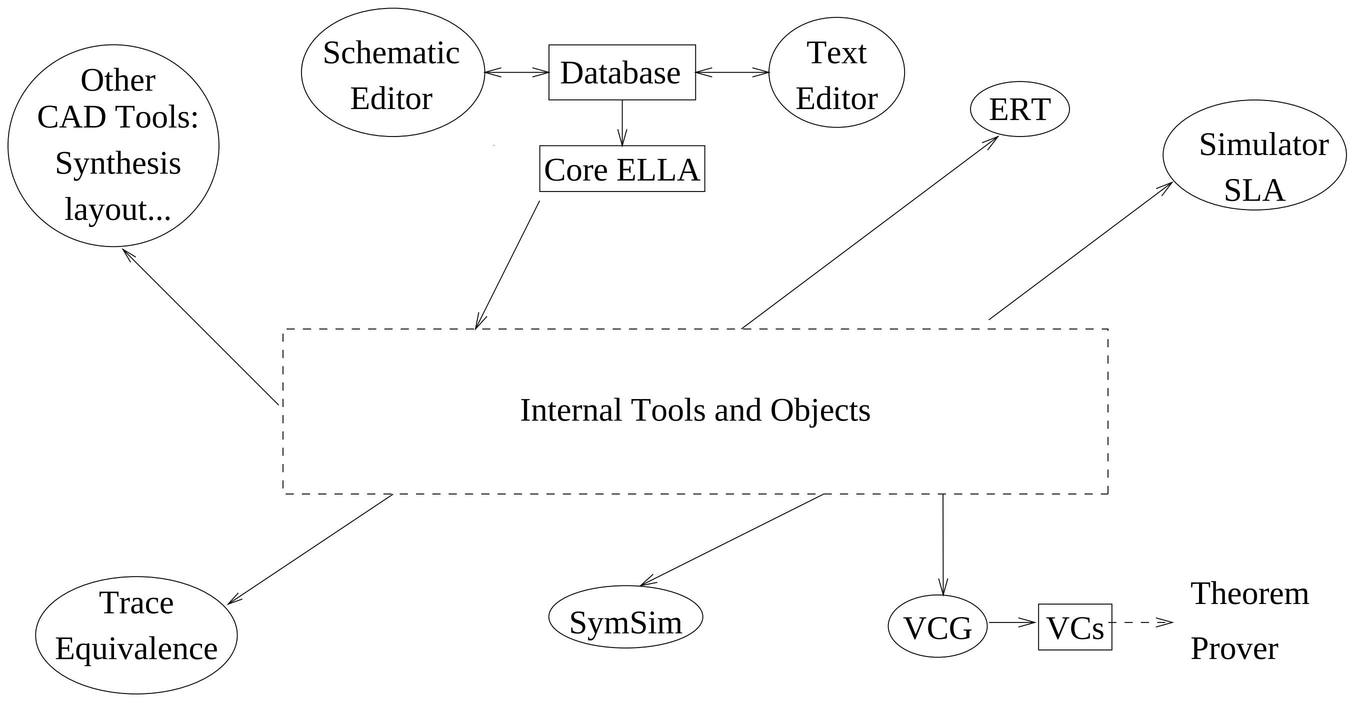
Funded by DTI/SERC

3rd August, 1995

Contents

Background


Schematic Editor
Database
Text Editor
Other CAD Tools: Synthesis layout...
Core ELLA
Simulator
Internal Tools and Objects



ERT: ELLA-level Rewrite Tool

Symsim: Symbolic Simulator

SLA: Soft Logic Analyser

VCG: Verification Condition Generator

Design Representation

ELLA Hardware Description Language

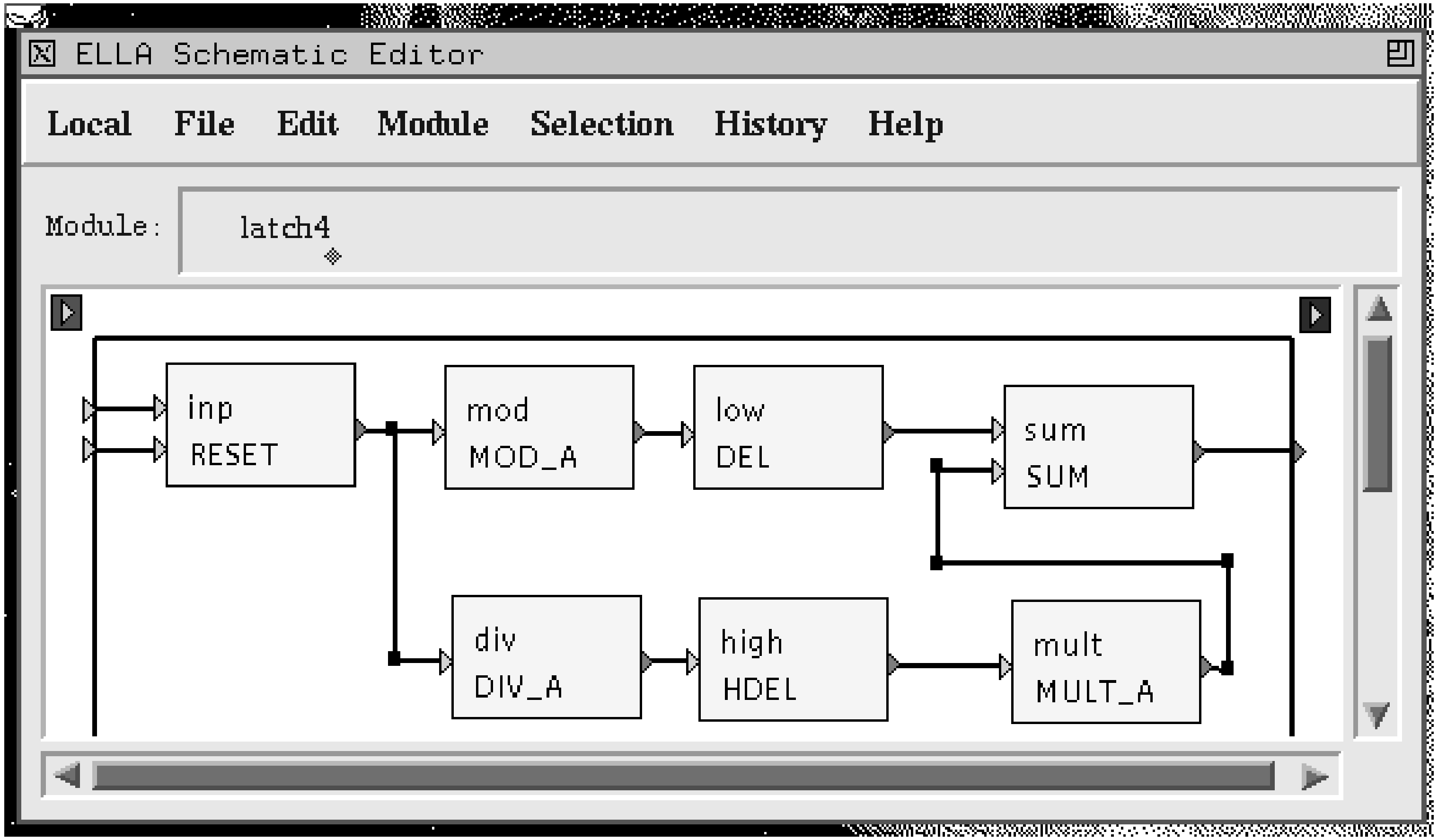
- synchronous communication
- simultaneous input/output
- simple timing model
- small covering subset: Core ELLA

```
TYPE control = NEW ( noreset | reset ).
TYPE data = NEW i/(0..20000).

FN DEL =  (data) -> data: IDELAY(i/0,1).

FN RESET = (control: cntrl, data: in) ->> data:
  CASE cntrl OF
   reset : i/0
  ELSE in ESAC.

FN LATCH = (control: cntrl, data: in) -> data:
BEGIN
   MAKE DEL: m.
   LET newm = RESET(cntrl, in)
   JOIN newm -> m.
   OUTPUT m
END.
```



ELLA Schematic Editor
Local File Edit Module Selection History Help
Module: latch4
inp
RESET
mod
MOD_A
low
DEL
sum
SUM
div
DIV_A
high
HDEL
mult
MULT_A

Semantics for ELLA #I

- set of Input/Output traces as semantic object
- use Input/Output Automaton (IOA)
  - i/o maps as transition labels
  - special IOA product operation
  - compositional semantics for Core ELLA
  - state-based trace-equivalence/containment
  - also BDD encoding
- complete semantics for ELLA

IOA Definition

$$
\begin{aligned}
\mathit{Trans} :: \quad \mathit{from} &: \mathit{State} \\
\mathit{ivals} &: \mathit{Name} \xrightarrow{\text{map}} \mathit{Val} \\
\mathit{ovals} &: \mathit{Name} \xrightarrow{\text{map}} \mathit{Val} \\
\mathit{to} &: \mathit{State}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{IOA} :: \qquad \mathit{states} &: \mathbb{P}(\mathit{State}) \\
\mathit{inputs} &: \mathbb{P}(\mathit{Name}) \\
\mathit{outputs} &: \mathbb{P}(\mathit{Name}) \\
\mathit{transitions} &: \mathbb{P}(\mathit{Trans}) \\
\mathit{start} &: \mathbb{P}(\mathit{State})
\end{aligned}
$$

Causal Product

For two i/o transition labels

- merge all input and output maplets
- *reject* if there are same-name channels holding different values
- *accept* if new map is consistent

New outputs: union of old outputs

New inputs: union of old inputs *minus* outputs

Channels w, x, y, z

	Label 1	Label 2	Result
(a)	$w(3)\ /\ y(1)$	$y(1)\ /\ x(2)$	$w(3)\ /\ (y(1), x(2))$
(b)	$w(3)\ /\ y(1)$	$y(6)\ /\ x(4)$	inconsistent on y

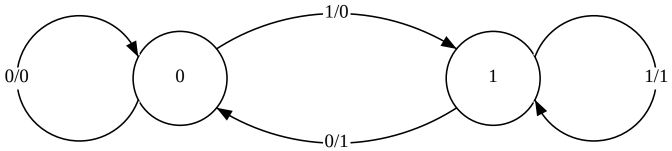
Formal Semantics Examples

$$IDelay \;::\; Init$$

$$MIDelay : Init \to SemObj$$

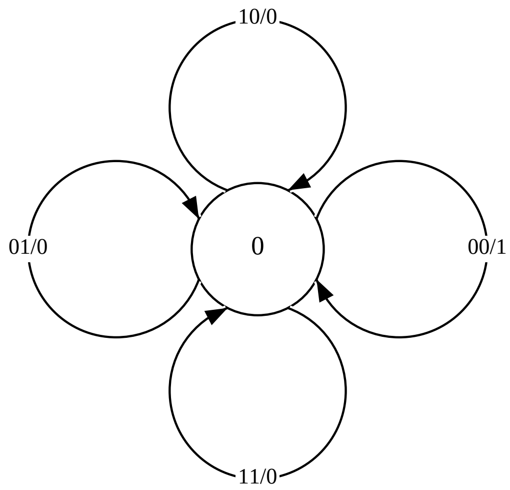
$$MIDelay[\![mk\text{-}IDelay(strt)]\!] \quad \underline{\triangle}$$

$$\begin{aligned}
&\text{let } ts = \{mk\text{-}Trans(0, \{i \mapsto 0\}, \{o \mapsto 0\}, 0)\\
&\qquad\qquad mk\text{-}Trans(0, \{i \mapsto 1\}, \{o \mapsto 0\}, 1)\\
&\qquad\qquad mk\text{-}Trans(1, \{i \mapsto 0\}, \{o \mapsto 1\}, 0)\\
&\qquad\qquad mk\text{-}Trans(1, \{i \mapsto 1\}, \{o \mapsto 1\}, 1)\\
&\qquad\quad \} \text{ in}\\
&mk\text{-}IOA(\{0,1\}, \{i\}, \{o\}, ts, \{strt\})
\end{aligned}$$



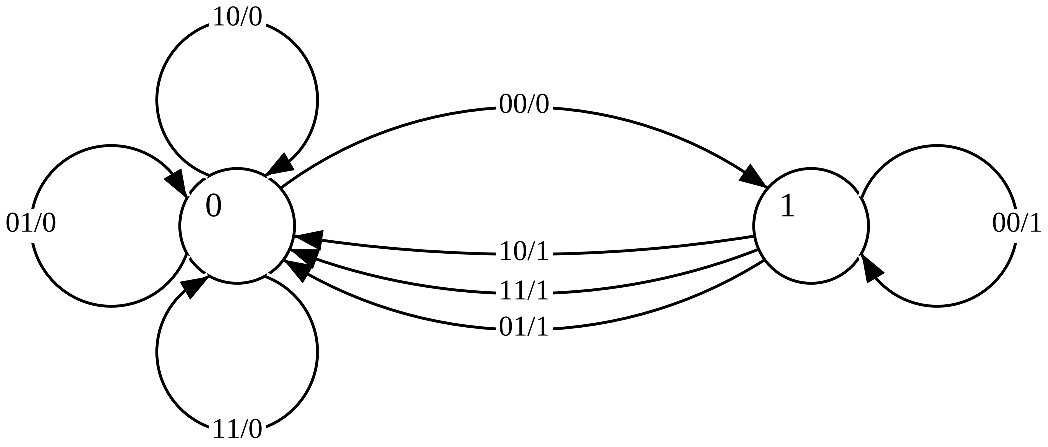
From State	Input i	Output o	To State
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Unit Delay



From	Input		Output	To
State	x	y	o	State
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0

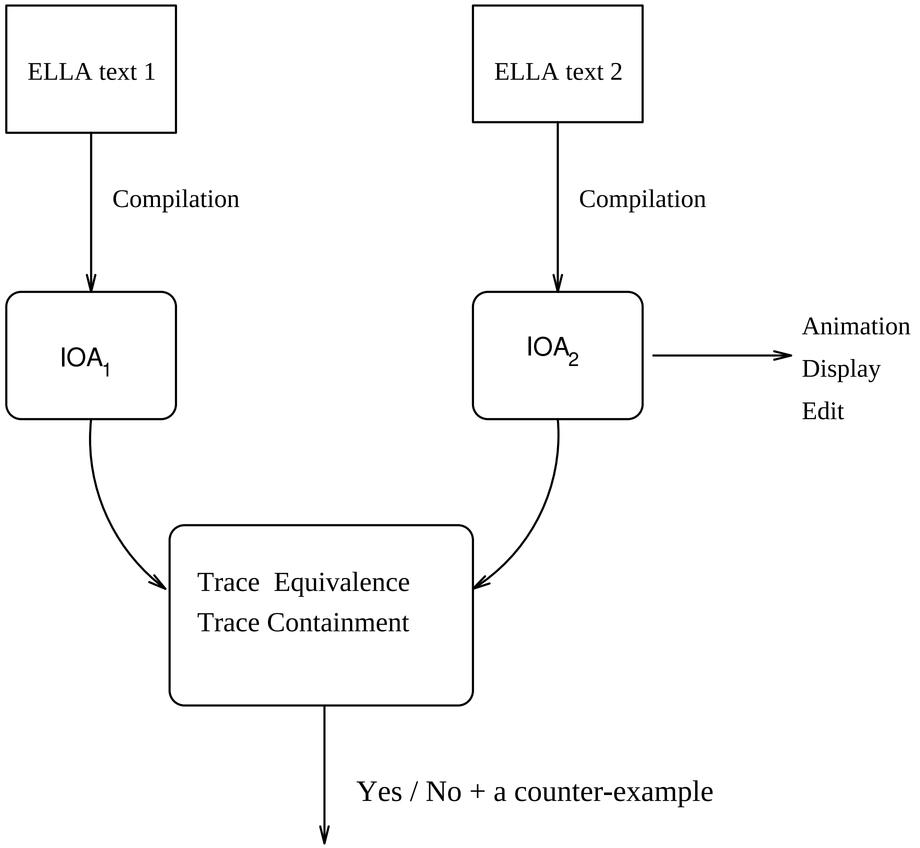
NOR Gate



From State	Input x	Input y	Output o	To State
0	0	0	0	1
0	-	-	0	0
1	0	0	1	1
1	1	1	1	0

Delay NOR Gate

Automata-Theoretic Verification



Problems with State-based Verification

Works well on small systems — of tens of thousands of states (not gates)

Combinatorial state space explosion is inherent

Large(r) designs can be handled through

- BDD encodings
- ELLA source level transformation

Development of process calculus, specifically for IO automata model.

Semantics for ELLA #II

Higher-level algebraic representation of IOAs:

ELLA Process Algebra (EPA):

- inspired by Milner's SCCS
- specialised composition operator
- value-passing process calls
- ELLA action algebra: “rich” actions
  - reduces amount of encoding required

EPA

$pnm(xs_1, \ldots, xs_k)$	$\triangleq$	P_1	Process Definition
$P \in Proc$	$::=$	$P_1 \mid P_2$	Specialised Product
	$\mid$	$\Box xs \cdot P$	Process Choice
	$\mid$	$P \backslash nm$	Hiding
	$\mid$	$\alpha :: P$	Action Prefix
	$\mid$	$pnm(x_1, \ldots, x_k)$	Process Call

$\alpha \in IOAct$, $xs = x$: S for some set S.

action structure: $i_1(x_1) \cdots i_m(x_m) \,/\, o_1(e_1) \cdots o_n(e_n)$

EPA Semantics

$$\textsf{Vacuum}\ \frac{}{P \xrightarrow{\boxtimes}_C P'}$$

$$\textsf{Hiding}\ \frac{P_1 \xrightarrow{a}_C P_1'}{P_1 \backslash ch\text{-}set \xrightarrow{hide\text{-}act(a,\, ch\text{-}set)}_C P_1' \backslash ch\text{-}set}$$

$$\textsf{Prefixing}\ \frac{}{a :: P_1 \xrightarrow{a}_C P_1}$$

$$\textsf{ProcChoice}\ \frac{v \in set, \quad P_1 [\, v / x \,] \xrightarrow{a}_C P_1'}{\square\, x : set \bullet P_1 \xrightarrow{a}_C P_1'}$$

$$\textsf{ProcDefn}\ \frac{P_1 \mathrel{\underline{\triangle}} P_2, \quad P_2 \xrightarrow{a}_C P'}{P_1 \xrightarrow{a}_C P'}$$

$$\textsf{Product}\ \frac{P_1 \xrightarrow{a}_C P_1', \quad P_2 \xrightarrow{b}_C P_2'}{P_1 \mid P_2 \xrightarrow{a \star b}_C P_1' \mid P_2'}$$

Action Product: $\star$

$$\star : (Act \cup \{\boxtimes, \varepsilon\})^2 \rightarrow Act \cup \{\boxtimes, \varepsilon\}$$

$$\forall\, a_1, a_2, b_1, b_2 \in IOAct, \gamma \in Act \cdot$$

$$(a_1 \;/\; b_1) \star (a_2 \;/\; b_2) = ((a_1 \oplus a_2) \ominus (b_1 \oplus b_2)) \;/\; (b_1 \oplus b_2)$$

$$\boxtimes \star \gamma = \boxtimes = \gamma \star \boxtimes$$

Channels w, x, y, z

$$(a) \quad (w(2) \;/\; y(1)) \star (y(1) \;/\; x(3)) = w(2) \;/\; y(1), x(3)$$

$$(b) \quad (w(2) \;/\; y(1)) \star (y(4) \;/\; x(3)) = \boxtimes$$

Formal Semanitcs #II

Unit Delay:

$$\text{process } Del_P(c_1\colon bool \mathbin{/} c_2\colon bool) \mathrel{\underline{\triangle}}$$

$$udel_{[bool]}(\texttt{f} \,;\, c_1 \mathbin{/} c_2)$$

using *generic* process definition :

$$\text{process } udel_{[ty]}(w\colon ty \,;\, c_1\colon ty \mathbin{/} c_2\colon ty) \mathrel{\underline{\triangle}}$$

$$\Box\, v\colon ty \;\boldsymbol{\cdot}\; c_1(v) \mathbin{/} c_2(w) :: udel_{[ty]}(v \,;\, c_1 \mathbin{/} c_2)$$

Product Form:

$$\Big(\; pnm_1(args_1) \; | \; \ldots \; | \; pnm_k(args_k) \; \Big) \backslash \{nm_1, \ldots, nm_m\}$$

e.g. Latch:

$$\begin{aligned}
&(Sel1(inp \;/\; ch1) \;|\; Sel2(inp \;/\; ch2) \;|\; Const0(\;/\; ch3) \;| \\
&\qquad Chooser(ch1, ch2, ch3 \;/\; ch4) \;|\; Udel(0 \;;\; ch4 \;/\; out) \\
&) \setminus \{ch1, ch2, ch3, ch4\}
\end{aligned}$$

Process Evolution

$$(Sel1(inp \ / \ ch1) \ | \ \cdots \ | \ Udel(22 \ ; \ ch4 \ / \ out)) \setminus \{ch1, ch2, ch3, ch4\}$$

$$\xrightarrow{inp(\langle noreset, 33\rangle) \ / \ out(22)}$$

$$(Sel1(inp \ / \ ch1) \ | \ \cdots \ | \ Udel(33 \ ; \ ch4 \ / \ out)) \setminus \{ch1, ch2, ch3, ch4\}$$

$$\xrightarrow{inp(\langle reset, 44\rangle) \ / \ out(33)}$$

$$(Sel1(inp \ / \ ch1) \ | \ \cdots \ | \ Udel(0 \ ; \ ch4 \ / \ out)) \setminus \{ch1, ch2, ch3, ch4\}$$

Deterministic Machines

DFA Form:

$$
\begin{aligned}
& P(\sigma_P \colon St_P \;;\; i \colon I \;/\; o \colon O) \;\underline{\triangle} \\
& \qquad \Box\, x \colon I \;\bullet\; i(x) \;/\; o(r_P(x, \sigma_P)) \\
& \qquad\qquad :: \; P(e_P(x, \sigma_P) \;;\; i \;/\; o)
\end{aligned}
$$

$$i(x) \;/\; o(r_P(x, \sigma_P)) \in IOAct$$

$$P(\sigma_P\colon St_P\ ;\ i\colon I\ /\ o\colon O) \mathrel{\underline{\triangle}}$$
$$\quad \Box\, x\colon I \cdot i(x)\ /\ o(r_P(x,\sigma_P)) :: P(e_P(x,\sigma_P)\ ;\ i\ /\ o)$$

$$D = \langle init_P, r_P, e_P \rangle$$

where

$init_P$	$\in St_P$	— `initial state`
r_P	$: (I \times St_P) \rightarrow O$	— `response function`
e_P	$: (I \times St_P) \rightarrow St_P$	— `evolution function`

$$P(m\colon data\ ;\ inp\colon control \times data\ /\ out\colon data) \triangleq$$
$$\Box\, cntrl\colon contrl, in\colon data\ \cdot$$
$$inp(\langle cntrl, in\rangle)\ /\ out(r_P(\langle cntrl, in\rangle, m))$$
$$::\ P(e_P(\langle cntrl, in\rangle, m)\ ;\ inp\ /\ out)$$

output expression: $r_P(\langle cntrl, in\rangle, m) = m$

state expression: $e_P(\langle cntrl, in\rangle, m) =$ **case** $cntrl$:

$reset \rightarrow 0$

else in

end

$control = \{reset, noreset\}$ $data = \{0, \ldots, 2000\}$

Input domain: $I = control \times data$ Output domain: $O = data$

$St_P = data$ Initial value: $m_0 = 0$

Process Evolution

Process Call: $P(22\ ;\ inp\ /\ out)$

$$P(22\ ;\ inp\ /\ out) \xrightarrow{inp(\langle noreset, 33\rangle)\ /\ out(22)} P(33\ ;\ inp\ /\ out)$$

$$\xrightarrow{inp(\langle reset, 44\rangle)\ /\ out(33)} P(0\ ;\ inp\ /\ out)$$

Symbolic Verification

State-based:

automated ($+$ve)

combinatorial explosion ($-$ve)

Theorem-proving:

high-level structured representation ($+$ve)

proof obligation extraction/formulation ($-$ve)

e.g. verification of VIPER:

– case analysis + theorem-proving

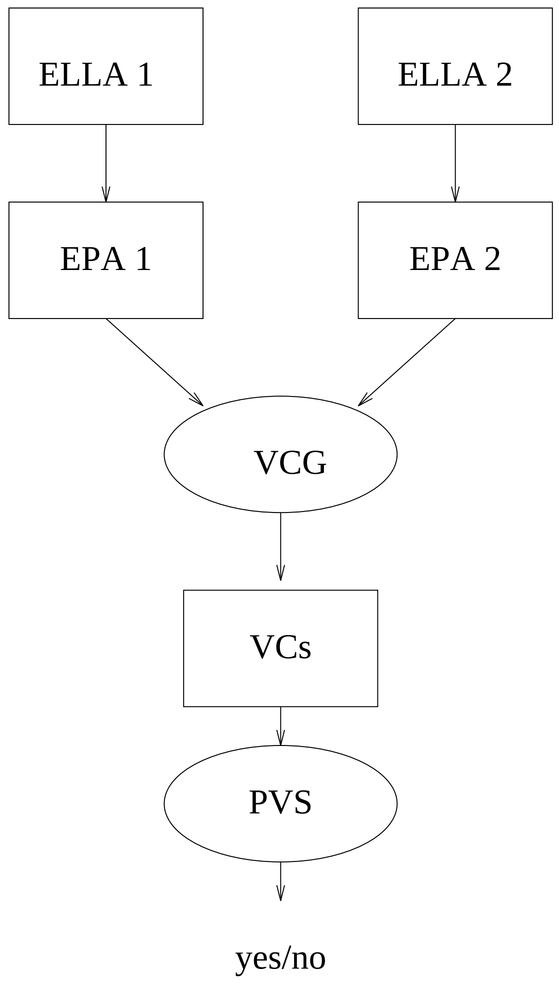
Combined approach:

“merge” approaches

use knowledge of application domain

extract (simple) proof obligations

Verification Approaches #II



Bisimulation

(Park and Milner)

$S \subseteq Proc \times Proc$ is a (strong) bisimulation if $(P,\ Q) \in S$ implies:

For all $\alpha \in IOAct$:

> Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in S$
> Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$ and $(P', Q') \in S$

P and Q are strongly bisimilar — $P \sim Q$ — if $(P,\ Q) \in S$ for some strong bisimulation S.

State Evolution

or... how to check for bisimulation...

$$R_i(\sigma_P, \sigma_Q) \;\Leftrightarrow$$
$$\forall\, x_i,\;\cdots\; x_1 \bullet r_P(x_i,\, sp_i) = r_Q(x_i,\, sq_i)$$

where

$$sp_i = e_P(x_{i\text{-}1}, sp_{i\text{-}1})$$
$$sq_i = e_Q(x_{i\text{-}1}, sq_{i\text{-}1})$$

$$AR_i(\sigma_P, \sigma_Q) \triangleq \bigwedge_{j=1}^{i} R_j(\sigma_P, \sigma_Q)$$

For some fixed $N \in \mathbb{N}$:

(a) $\Gamma \vdash AR_N(i_P, i_Q)$

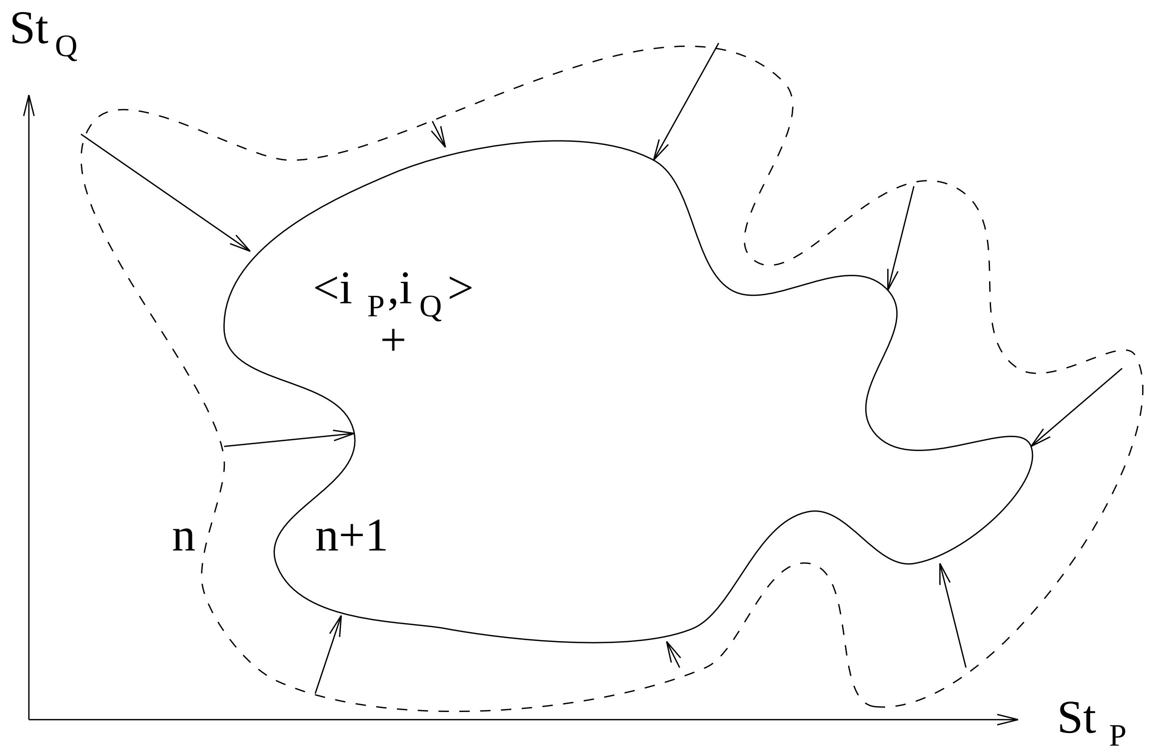
(b) $\Gamma \vdash \forall\, \sigma_P,\ \sigma_Q\ \centerdot$

$$AR_N(\sigma_P, \sigma_Q) \ \Rightarrow\ R_{N+1}(\sigma_P, \sigma_Q)$$

$$\boxed{\text{St-Ev}}\ \frac{\qquad\qquad}{\Gamma \vdash P \sim Q}$$

How to find N?

Can N always be found?


St Q
<i P,i Q>
+
n
n+1
St P

Example

	P	Q
response	$r_P(\langle cntrl, in\rangle, m) = m$	$r_Q(\langle cntrl, in\rangle, \langle low, high\rangle) =$ $low + A * high$
evolution:	$e_P(\langle cntrl, in\rangle, m) =$ **case** $cntrl$: $reset \to 0$ **else** in **end**	$e_Q(\langle cntrl, in\rangle, \langle low, high\rangle) =$ **case** $cntrl$: $reset \to \langle 0, 0\rangle$ **else** $\langle in$ **mod** A, in **div** $A\rangle$ **end**

$$I = control \times data \qquad St_Q = data \times data \qquad A \in data, (\neq 0)$$

$$(R_1(m_1, \langle low_1, high_1 \rangle) \quad \Rightarrow \quad R_2(m_1, \langle low_1, high_1 \rangle))$$

$$\Leftrightarrow \; \forall \, in_2, cntrl_2, in_1, cntrl_1 \cdot$$
$$r_P(\langle cntrl_1, in_1 \rangle, m_1) = r_Q(\langle cntrl_1, in_1 \rangle, \langle low_1, high_1 \rangle)$$
$$\Rightarrow \; r_P(\langle cntrl_2, in_2 \rangle, m_2) = r_Q(\langle cntrl_2, in_2 \rangle, \langle low_2, high_2 \rangle)$$

$$\Leftrightarrow \; ((m_1 = low_1 + A * high_1) \quad \Rightarrow \; (m_2 = low_2 + A * high_2))$$

Verification Condition Generation

Functional Normal Form, and merge:

$$F(x) \quad \triangleq \quad \bigwedge_{i=0}^{k} (A_i[x] \;\Rightarrow\; t_i) \qquad\qquad G(x) \quad \triangleq \quad \bigwedge_{j=0}^{m} (B_j[x] \;\Rightarrow\; u_j)$$

$$\frac{F(x) = G(x)}{\bigwedge_{i,j} (A_i[x] \wedge B_j[x] \;\Rightarrow\; t_i = u_j)}\;\boxed{\text{FNF}}$$

- conditions: A_i, B_j
- (caseless) terms: t_i, u_j.
- case analysis (disjoint, covering)

i.e. use to decompose logical expression for state evolution

$(m_1 = low_1 + A * high_1)$

$\Rightarrow \ (m_2 = low_2 + A * high_2)$

$$
\begin{array}{l}
m_1 = low_1 + A * high_1 \\
\qquad \wedge \ m_2 = \textbf{case } cntrl_1 : \\
\qquad\qquad\qquad reset_1 \rightarrow 0 \\
\qquad\qquad \textbf{else } in_1 \\
\qquad\qquad \textbf{end} \\
\qquad \wedge \ \langle low_2, high_2 \rangle = \textbf{case } cntrl : \\
\qquad\qquad\qquad\qquad reset \rightarrow \langle 0, 0 \rangle \\
\qquad\qquad\qquad \textbf{else } \langle in \textbf{ mod } A, in \textbf{ div } A \rangle \\
\qquad\qquad\qquad \textbf{end} \\
\ \Rightarrow \ m_2 = low_2 + A * high_2
\end{array}
$$

$$
\begin{array}{l}
m_2 = \textbf{case}\ cntrl_1 : \\
\qquad\qquad reset \rightarrow 0 \\
\qquad \textbf{else}\ in_1 \\
\qquad \textbf{end}
\end{array}
$$

$$
\left(\begin{array}{ll} & cntrl_1 \in \{reset\} \\ \wedge & m_2 = 0 \end{array} \right) \tag{1}
$$

$$
\vee
$$

$$
\left(\begin{array}{ll} & cntrl_1 \in control — \{reset\} \\ \wedge & m_2 = in_1 \end{array} \right) \tag{2}
$$

$$\frac{\begin{array}{l} \langle low_2, high_2 \rangle = \textbf{case}\ cntrl : \\ \qquad\qquad reset \rightarrow \langle 0, 0 \rangle \\ \qquad \textbf{else}\ \langle in\ \textbf{mod}\ A, in\ \textbf{div}\ A \rangle \\ \qquad \textbf{end} \end{array}}{\begin{array}{l} \left(\begin{array}{ll} & cntrl_1 \in \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \end{array} \right) \qquad (1) \\ \qquad \vee \\ \left(\begin{array}{ll} & cntrl_1 \in control - \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle in\ \textbf{mod}\ A, in\ \textbf{div}\ A \rangle \end{array} \right) \qquad (2) \end{array}}$$

$$(m_1 = low_1 + A * high_1) \ \wedge$$

$$\begin{pmatrix} & cntrl_1 \in \{reset\} \\ \wedge & m_2 = 0 \end{pmatrix} \wedge \begin{pmatrix} & cntrl_1 \in \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \end{pmatrix} \Rightarrow m_2 = low_2 + A * high_2$$

$$\begin{pmatrix} & cntrl_1 \in control - \{reset\} \\ \wedge & m_2 = in_1 \end{pmatrix} \wedge \begin{pmatrix} & cntrl_1 \in \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \end{pmatrix} \Rightarrow m_2 = low_2 + A * high_2$$

$$\begin{pmatrix} & cntrl_1 \in \{reset\} \\ \wedge & m_2 = 0 \end{pmatrix} \wedge \begin{pmatrix} & cntrl_1 \in control - \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle in \ \textbf{mod} \ A, in \ \textbf{div} \ A \rangle \end{pmatrix} \Rightarrow m_2 = low_2 + A * high_2$$

$$\begin{pmatrix} & cntrl_1 \in control - \{reset\} \\ \wedge & m_2 = in_1 \end{pmatrix} \wedge \begin{pmatrix} & cntrl_1 \in control - \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle in \ \textbf{mod} \ A, in \ \textbf{div} \ A \rangle \end{pmatrix} \Rightarrow m_2 = low_2 + A * high_2$$

Theorem-Proving

$$
\begin{array}{ll}
 & cntrl_1 \in \{reset\} \\
\wedge & m_1 = low_1 + A * high_1 \\
\wedge & m_2 = 0 \\
\wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \\
\Rightarrow & m_2 = low_2 + A * high_2
\end{array}
\quad\Bigg|\quad
\begin{array}{ll}
 & cntrl_1 \in control - \{reset\} \\
\wedge & m_1 = low_1 + A * high_1 \\
\wedge & m_2 = in_1 \\
\wedge & \langle low_2, high_2 \rangle = \langle in_1 \textbf{ mod } A, in_1 \textbf{ div } A \rangle \\
\Rightarrow & m_2 = low_2 + A * high_2
\end{array}
$$

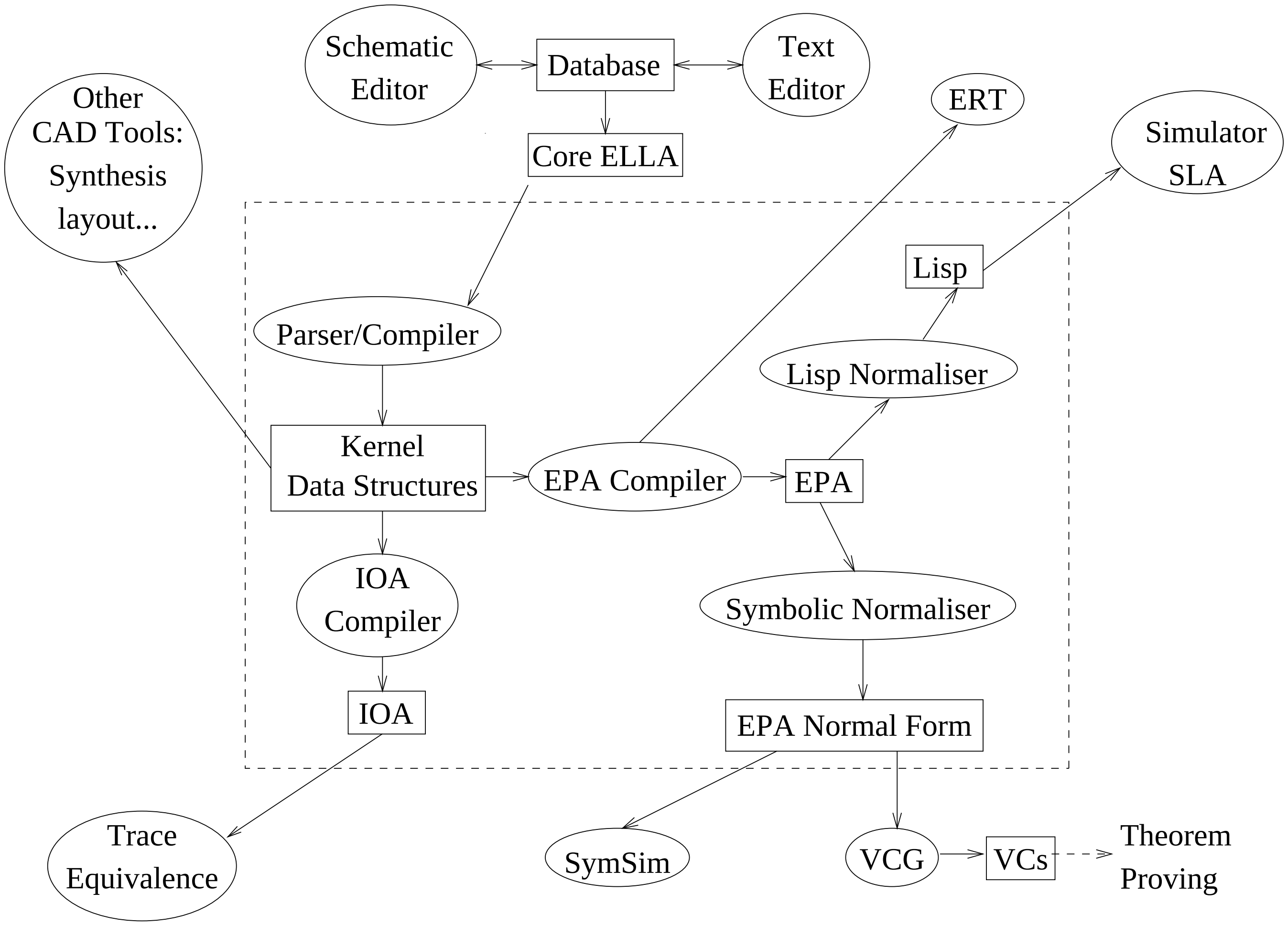
- VC simplification
- add axioms/declarations (from design assumptions):

```
div_ax: AXIOM (FORALL (x: int), (y: int):
   ((y /= 0) IMPLIES (y * div(x, y) + mod(x, y) = x)))
```

- VCs very similar to format of Theorem-Prover (PVS)
- VCG implementation embedded in Design Environment
  - LispWorks/CLOS (Harlequin), GT (Manchester)

(*PVS: from SRI International*)

Summary



Schematic Editor
Database
Text Editor
Core ELLA
ERT
Other CAD Tools: Synthesis layout...
Simulator SLA
Lisp
Parser/Compiler
Lisp Normaliser
Kernel Data Structures
EPA Compiler
EPA
IOA Compiler
Symbolic Normaliser
IOA
EPA Normal Form
Trace Equivalence
SymSim
VCG
VCs
Theorem Proving

Embedded verification tools

Full semantics for commercial HDL:

- IOA
- EPA

Bisimulation Equvalence via

- State Evolution
- Verification Conditions

Semi-automated approach:

- automatic theorem generation
- ‘simple’ conditions to prove

WWW: `http://www.cs.man.ac.uk/fmethods/projects/ella-project.html`