CHDL'95, Tokyo

C-6.3: Symbolic verification of hardware systems

Howard Barringer Graham Gough, Brian Monahan, Alan Williams

Department of Computer Science, University of Manchester

Partners: Harlequin Ltd, DRA (Malvern)

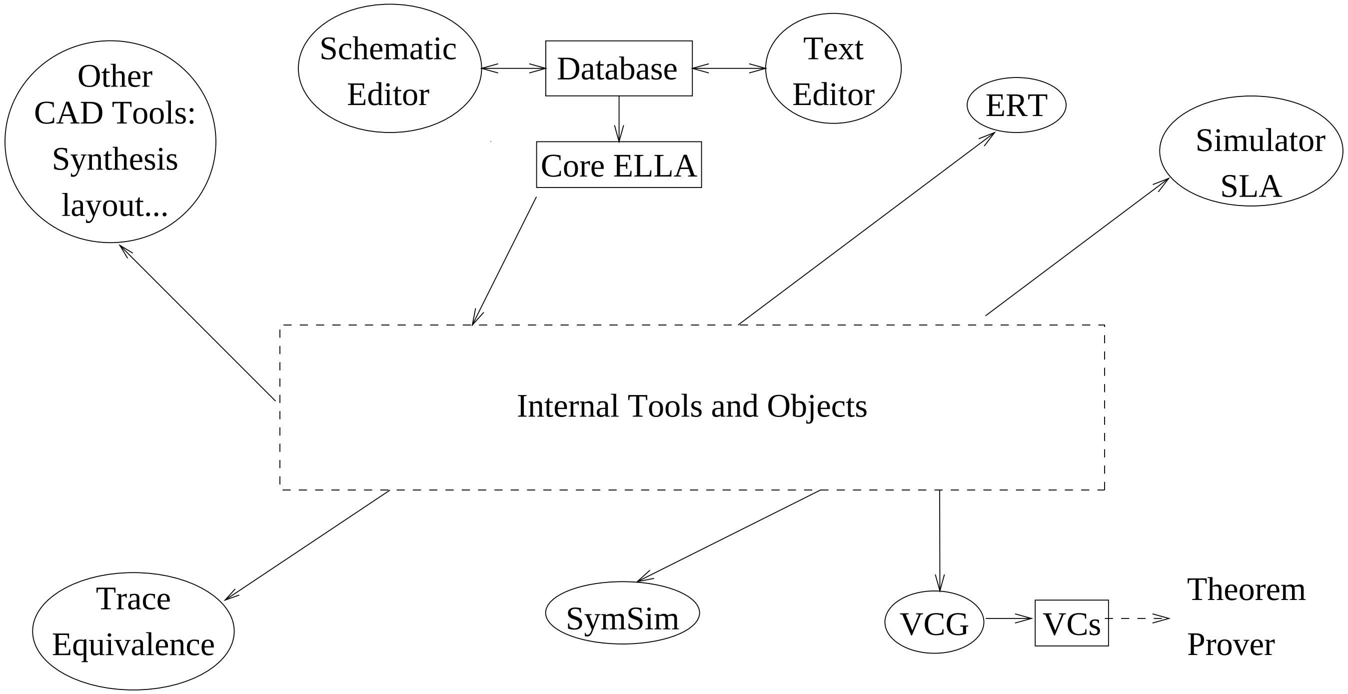
Funded by DTI/SERC (UK)

31st August, 1995

Contents

Background

- utilisation of formal methods in design cycle
- tractable verification techniques:
  - avoiding state explosion
  - semi-automatic
- feedback to designer



ERT: ELLA-level Rewrite Tool

Symsim: Symbolic Simulator

SLA: Soft Logic Analyser

VCG: Verification Condition Generator

Formal Verification

- trace equivalence
- state-space exploration of Finite State Machines
- standard automatic algorithms
- ‘state-explosion’ problem
- BDD encoding to extend capability

Verification Approaches

State-based:

automated ($+$ve)

combinatorial explosion ($-$ve)

Theorem-proving:

high-level structured representation ($+$ve)

proof obligation extraction/formulation ($-$ve)

proofs can require high-level of theorem proving expertise.

e.g. verification of VIPER:

- case analysis + theorem-proving

Combined approach:

“merge” approaches

use knowledge of application domain

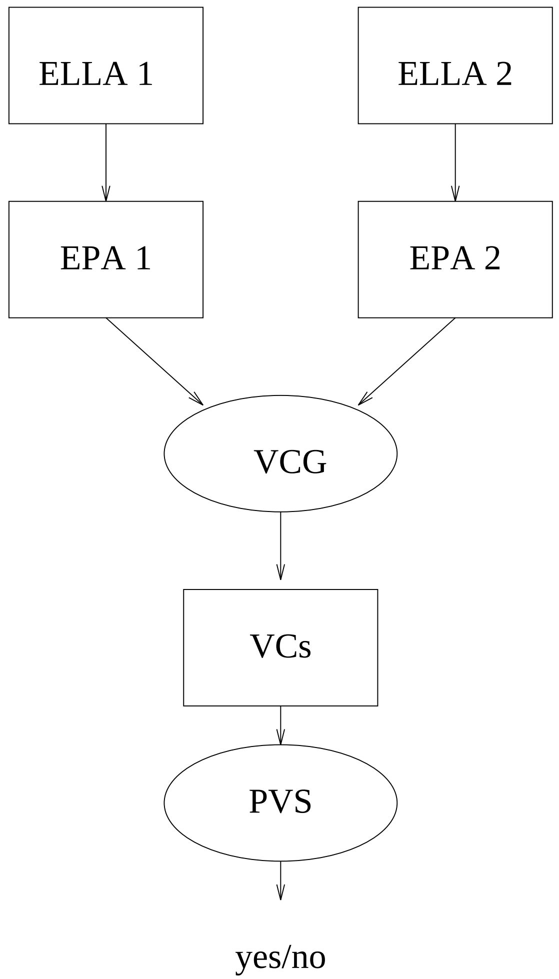
extract (simple) proof obligations

Symbolic Verification — problems to be solved:

- state-space exploration via symbolic simulation: build state-space relation and check for closure
- termination conditions: covering of state-space
- equivalence of responses

Approach:

- employ (independent) theorem-proving for equivalence-checking of response expressions
- derive logical expressions to characterise closure of state-space relation


ELLA 1
ELLA 2
EPA 1
EPA 2
VCG
VCs
PVS
yes/no

Deterministic Machines

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

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`

Example:

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

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

$$reset \rightarrow 0$$

$\textbf{else}\ in$

$\textbf{end}$

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

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

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

State Evolution

or... how to check for trace equivalence...

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

where

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

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

$$\text{(a)} \quad \Gamma \vdash \bigwedge_{j=1}^{N} R_j(i_P, i_Q)$$

$$\text{(b)} \quad \Gamma \vdash \forall\, \sigma_P,\ \sigma_Q \ \cdot$$

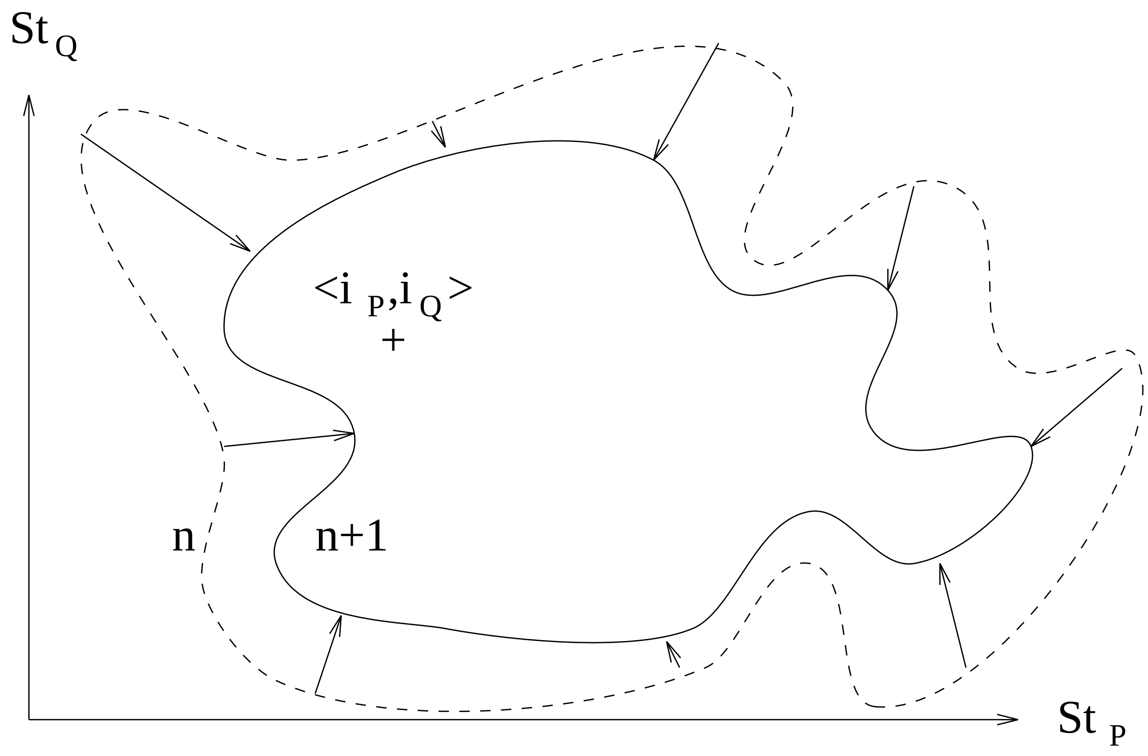
$$\bigwedge_{j=1}^{N} R_j(\sigma_P, \sigma_Q) \ \Rightarrow\ R_{N+1}(\sigma_P, \sigma_Q)$$

$$\boxed{\text{St-Ev}} \quad \overline{\qquad\qquad\qquad\qquad\qquad\qquad}$$

$$\Gamma \vdash P \sim Q$$

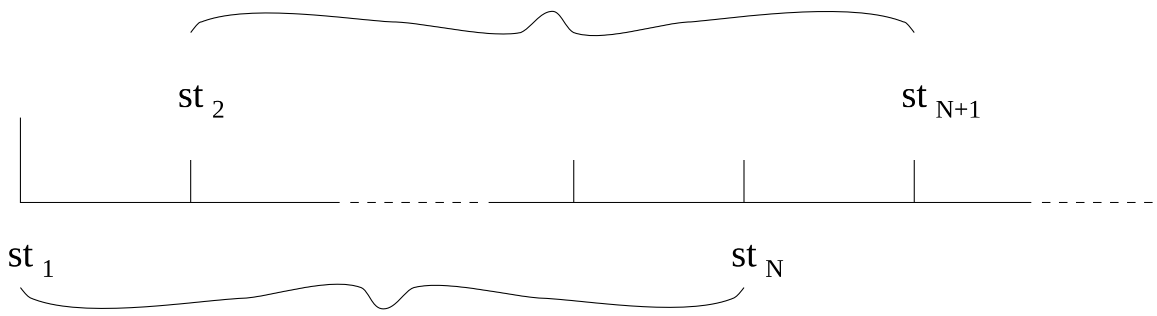
How to find N?

Can N always be found?



St_Q
$<i_P, i_Q>$
+
n
n+1
St_P

State Evolution (Continued)



Example

We now introduce a process Q which is (designed to be) equivalent to P.

Stores contents of buffer in two components (high and low parts).

Illustrate method by showing that P and Q are equivalent.

	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** $cntrl$: $reset \rightarrow 0$ **else** in **end**	$e_Q(\langle cntrl, in\rangle, \langle low, high\rangle) =$ **case** $cntrl$: $reset \rightarrow \langle 0, 0\rangle$ **else** $\langle in$ **mod** A, in **div** $A\rangle$ **end**

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

For $N = 1$, premise (b) above becomes

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

$$\begin{aligned} \text{iff} \quad & \forall\, in_2, cntrl_2, in_1, cntrl_1 \cdot \\ & \quad r_P(\langle cntrl_1, in_1\rangle, m_1) = r_Q(\langle cntrl_1, in_1\rangle, \langle low_1, high_1\rangle) \\ & \qquad \Rightarrow\; r_P(\langle cntrl_2, in_2\rangle, m_2) = r_Q(\langle cntrl_2, in_2\rangle, \langle low_2, high_2\rangle) \end{aligned}$$

$$\text{iff} \quad ((m_1 = low_1 + A * high_1) \quad \Rightarrow \; (m_2 = low_2 + A * high_2))$$

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

$(m_1 = low_1 + A * high_1)$

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

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

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

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

$$\vee$$

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

$$\langle low_2, high_2 \rangle = \textbf{case}\ cntrl_1 : \quad reset \rightarrow \langle 0, 0 \rangle \quad \textbf{else}\ \langle in_1\ \textbf{mod}\ A, in_1\ \textbf{div}\ A \rangle \quad \textbf{end}$$

$$\left(\begin{array}{ll} & cntrl_1 \in \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \end{array} \right) \qquad (1)$$

$$\vee$$

$$\left(\begin{array}{ll} & cntrl_1 \in control - \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle in_1\ \textbf{mod}\ A, in_1\ \textbf{div}\ A \rangle \end{array} \right) \qquad (2)$$

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

$$\left(\begin{array}{ll} & cntrl_1 \in \{reset\} \\ \wedge & m_2 = 0 \end{array}\right) \wedge \left(\begin{array}{ll} & cntrl_1 \in \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \end{array}\right) \Rightarrow m_2 = low_2 + A * high_2$$

$$\left(\begin{array}{ll} & cntrl_1 \in control - \{reset\} \\ \wedge & m_2 = in_1 \end{array}\right) \wedge \left(\begin{array}{ll} & cntrl_1 \in \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \end{array}\right) \Rightarrow m_2 = low_2 + A * high_2$$

$$\left(\begin{array}{ll} & cntrl_1 \in \{reset\} \\ \wedge & m_2 = 0 \end{array}\right) \wedge \left(\begin{array}{ll} & cntrl_1 \in control - \{reset\} \\ \wedge & \langle low_2, high_2 \rangle = \langle in_1 \ \textbf{mod} \ A, in_1 \ \textbf{div} \ A \rangle \end{array}\right) \Rightarrow m_2 = low_2 + A * high_2$$

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

Theorem-Proving

$$
\begin{array}{ll}
& cntrl_1 \in \{reset\} \\
\wedge & m_1 = low_1 + A * high_1 \\
\wedge & m_2 = 0 \\
\wedge & \langle low_2, high_2 \rangle = \langle 0, 0 \rangle \\
\Rightarrow & m_2 = low_2 + A * high_2
\end{array}
$$

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

- 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)[a]
- VCG implementation embedded in Design Environment
  - LispWorks/CLOS (Harlequin), GT (Manchester)

[a] from SRI International

Summary

Embed verification tools in Design Environment for ELLA HDL (see C-9.2)

Bisimulation Equvalence via

- State Evolution
- Verification Conditions

Semi-automated approach:

- automatic theorem generation
- ‘simple’ conditions to prove

Data abstraction

Preserve design structure: useful for feedback

Utilise designer information

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