Symbolic Verification of Deterministic Machines

Howard Barringer
Graham Gough
Alan Williams

HaVe Group
Department of Computer Science
University of Manchester

Brian Monahan, Harlequin, Ltd.

EuroForm Workshop, Ile des Embiez, France
15–16 September, 1994

The State Evolution Method for,

and Symbolic Bisimulation of,

Infinite,

but Bounded Diameter, State Machines

CONTENTS

Background

Our approach has been strongly guided by

> …embedding formal methods within the hardware designers normal environment …

Design capture/representation

ELLA:

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

Input/Output Automaton (IOA) compositional semantics for Core ELLA

State-base trace-equivalence/containment

BDD encoding

Process Algebraic Semantics

Higher-level algebraic representation of IOAs:

Process Algebra Framework (PAF):

- inspired by Milner + SCCS
- separate action algebra
- value-passing process calls

Instantiation: ELLA Process Algebra (EPA):

- ELLA action algebra: “rich” actions ($IOAct$)
- specialised composition operator

EPA

$$
\begin{array}{lcll}
P \in Proc & ::= & P_1 \;||\; P_2 & \text{Specialised Product} \\
& | & \Box xs \bullet P & \text{Process Choice} \\
& | & P \backslash nm & \text{Hiding} \\
& | & \alpha :: P & \text{Action Prefix} \\
& | & P(x_1, \ldots, x_k) & \text{Process Call}
\end{array}
$$

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

EPA DFA Form:

$$
\begin{array}{l}
P(v \;;\; i\colon I \;/\; o\colon O) \;\underline{\triangle} \\
\quad \Box x\colon I \bullet i(x) \;/\; o(r_P(x, v)) \\
\qquad :: P(e_P(x, v) \;;\; i \;/\; o)
\end{array}
$$

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

EPA (continued)

Example: Delayed Inverter:

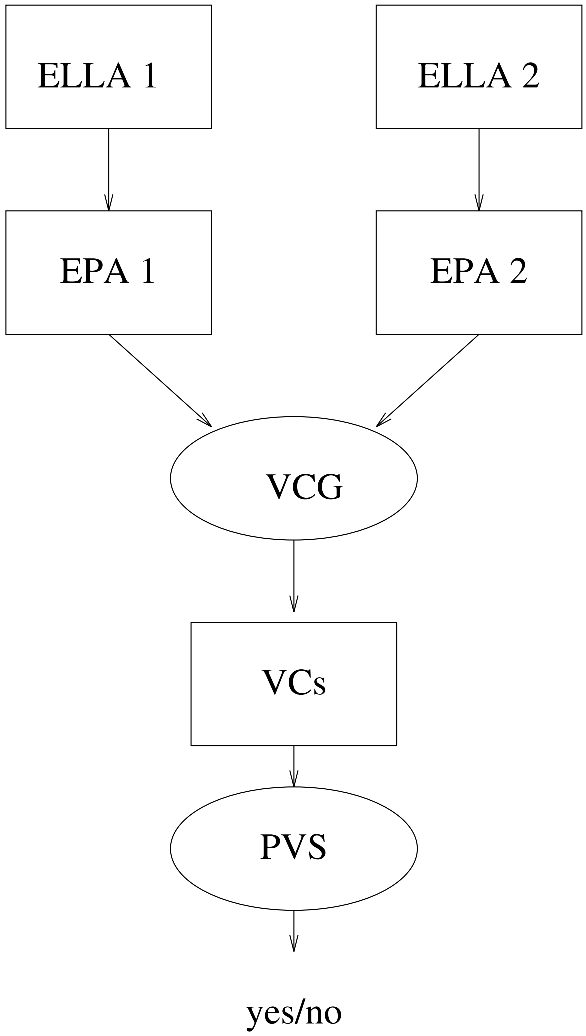
$$
\begin{aligned}
&P(v\ ;\ i\colon I\ /\ o\colon O)\ \underline{\triangle}\\
&\quad \Box x\colon I\ \bullet\ i(x)\ /\ o(r_P(x,v))\\
&\qquad :: P(e_P(x,v)\ ;\ i\ /\ o)
\end{aligned}
$$

Input/Output domains:	$I, O = \mathbb{B}$
Initial value:	$v = f$
output expression:	$r_P(x,v) = v$
state expression:	$e_P(x,v) =$ **case** x **in**
	$\quad t : f$
	else t
	end

Verification Approaches

- State-based:
  - automated (+ve)
  - combinatorial explosion ($-$ve)
- Theorem-proving:
  - high-level structured representation (+ve)
  - proof obligation extraction/formulation ($-$ve)
  - verification of VIPER: (e.g)
    case analysis + theorem-proving
- Combined approach:
  - “merge” approaches
  - use knowledge of application domain
  - extract (simple) proof obligations

Verification Approaches (continued)



Bisimulation

(Park and Milner)

The relation $S \subseteq \mathit{Proc} \times \mathit{Proc}$ is a (strong) bisimulation if $(P,\ Q) \in S$ implies, for all $\alpha \in \mathit{IOAct}$:

- 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.

Deterministic Machines

Represent process P as...

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

I, O	– input/output domains
St_P	– state space
$i_P \in St_P$	– initial state
$r_P\colon I \times St_P \to O$	– response function
$e_P\colon I \times St_P \to St_P$	– evolution function

note: bisimulation for deterministic machine is equivalent to trace equivalence

State Bisimulation

$SB \subseteq (St_P \times St_Q)$:

SB is a state bisimulation for $P,\ Q$ when:

1. $\langle i_P,\ i_Q \rangle \in SB$
2. If $\langle \sigma_P,\ \sigma_Q \rangle \in SB$ Then

   $\forall\, x \in I$,

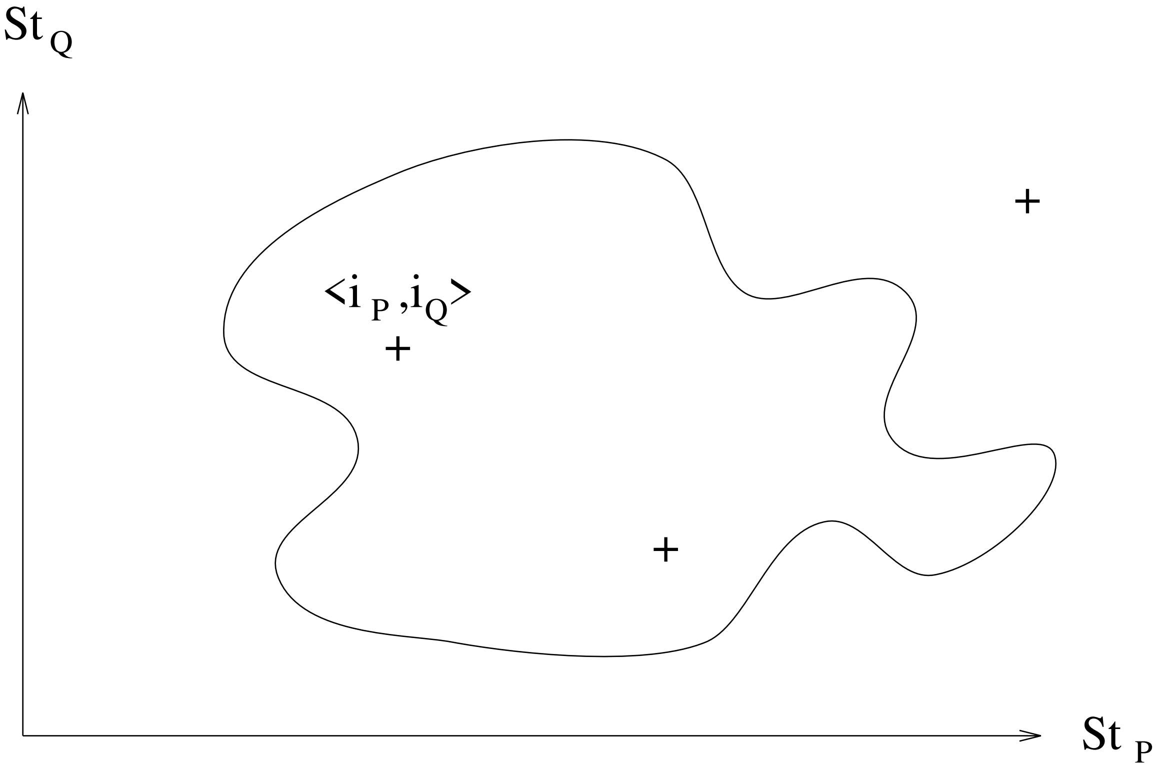
   (a) $r_P(x, \sigma_P) = r_Q(x, \sigma_Q)$, and

   (b) $\langle e_P(x, \sigma_P),\ e_Q(x, \sigma_Q) \rangle \in SB$

Correctness of state bisimulation

Let $P,\ Q$ be deterministic systems in EPA DFA form.

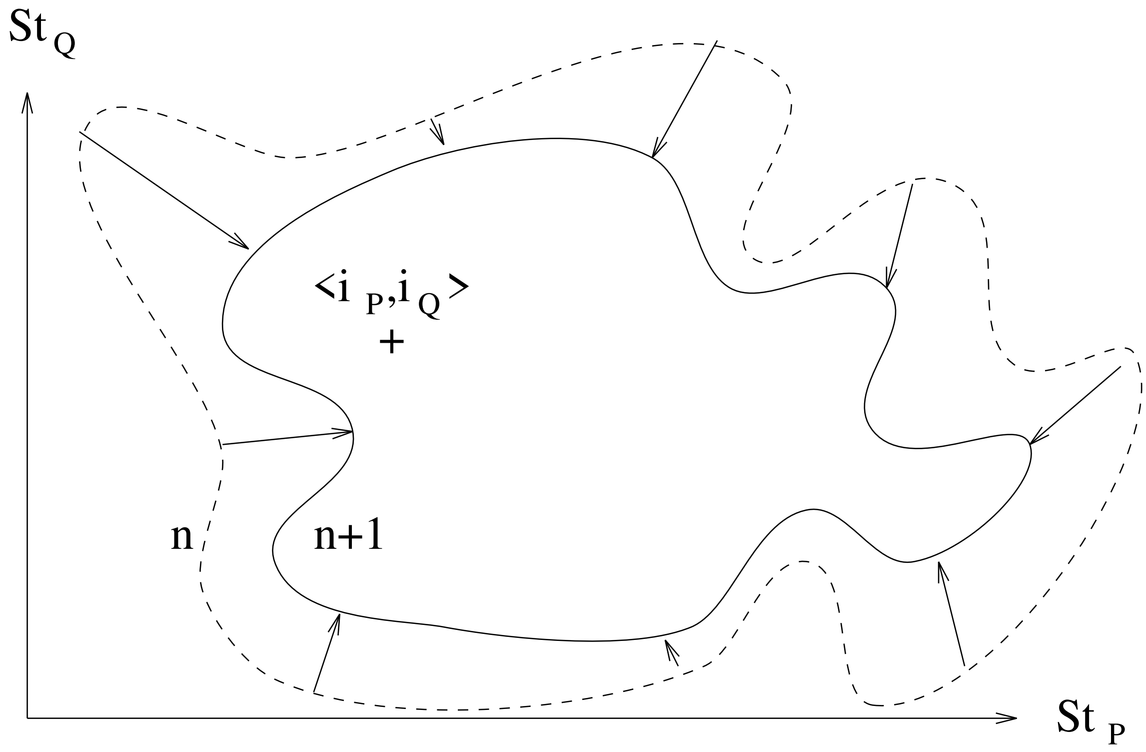
$$(P \sim Q) \;\Leftrightarrow\; \exists\, SB \subseteq (St_P \times St_Q) \cdot$$
$$SB \text{ is a state bisimulation for } P \text{ and } Q$$



State Bisimulation SB between P and Q

State Evolution

Calculation of state bisimulation



State Evolution (continued)

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

where

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

$$sp_1 = \sigma_P$$
$$sq_1 = \sigma_Q$$

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

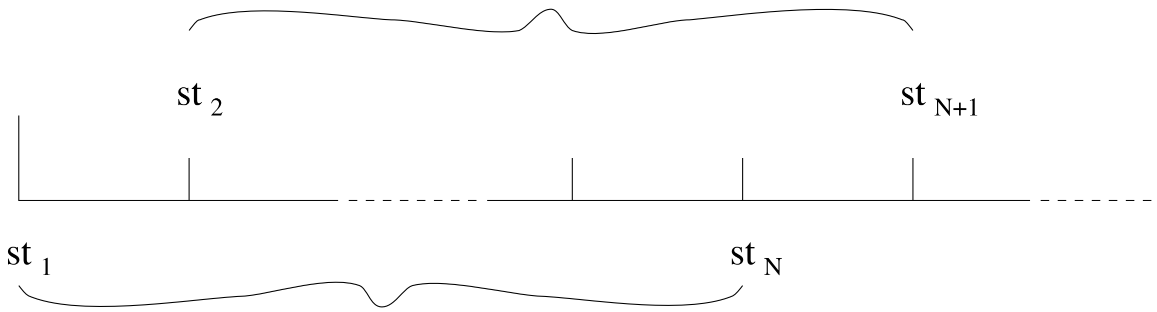
State Evolution (continued)

$$\frac{\begin{array}{l}\text{For some fixed } N \in \mathbb{N}: \\ \quad \text{(a)} \quad \Gamma \vdash AR_N(i_P, i_Q) \\ \quad \text{(b)} \quad \Gamma \vdash \forall\, \sigma_P,\ \sigma_Q \ \cdot \\ \qquad\qquad AR_N(\sigma_P, \sigma_Q) \ \Rightarrow\ R_{N+1}(\sigma_P, \sigma_Q)\end{array}}{\Gamma \vdash P \sim Q}\ \boxed{\text{St-Ev}}$$

How to find N?:
related to circuit latency

Can N always be found?:
termination conditions: e.g. bounded diameter

State Evolution (continued)



fixed N

symbolic representation:
universally-quantified state variables

but ... how to find:

$AR_N(i_P, i_Q)$

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

Verification Condition Generation

Functional Normal Form, and merge:

$$F(x) \triangleq \bigwedge_{i=0}^{k} (A_i[x] \Rightarrow t_i)$$

$$G(x) \triangleq \bigwedge_{j=0}^{m} (B_j[x] \Rightarrow u_j)$$

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

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

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

Verification Condition Generation (continued)

Example: Delayed Inverter ($N = 1$)

$$
\begin{array}{ll|ll}
 & v_1 = w_1 & & R_1(v_1, w_1) \\
\wedge & v_2 = \textbf{case } x_1 \textbf{ in} & \wedge & v_2 = e_P(x_1, v_1) \\
 & \quad\quad t : f & & \\
 & \quad \textbf{else } t & & \\
 & \quad \textbf{end} & & \\
\wedge & w_2 = \textbf{case } x_1 \textbf{ in} & \wedge & w_2 = e_Q(x_1, v_1) \\
 & \quad\quad t : f & & \\
 & \quad \textbf{else } t & & \\
 & \quad \textbf{end} & & \\
\Rightarrow & v_2 = w_2 & \Rightarrow & R_2(v_1, w_1)
\end{array}
$$

Verification Condition Generation (continued)

$v_2 =$ **case** x_1 **in**

$\quad t : f$

else t

end

$$
\begin{array}{lll}
\multicolumn{3}{l}{\forall\, v_2 : \mathbb{B} \cdot} \\
 & x_1 \in \{t\} & (1) \\
\wedge & v_2 = f & \\
 & \vee & \\
 & x_1 \in \mathbb{B} - \{t\} & (2) \\
\wedge & v_2 = t &
\end{array}
$$

VC Merge Example

$v_2, w_2, x_1 \in \mathbb{B}$:

$$\left(\begin{array}{ll} & x_1 \in \{t\} \\ \wedge & v_2 = f \end{array}\right) \quad \wedge \quad \left(\begin{array}{ll} & x_1 \in \{t\} \\ \wedge & w_2 = f \end{array}\right) \quad \Rightarrow \; v_2 = w_2$$

$$\left(\begin{array}{ll} & x_1 \in \mathbb{B} - \{t\} \\ \wedge & v_2 = t \end{array}\right) \quad \wedge \quad \left(\begin{array}{ll} & x_1 \in \{t\} \\ \wedge & w_2 = f \end{array}\right) \quad \Rightarrow \; v_2 = w_2$$

$$\left(\begin{array}{ll} & x_1 \in \{t\} \\ \wedge & v_2 = f \end{array}\right) \quad \wedge \quad \left(\begin{array}{ll} & x_1 \in \mathbb{B} - \{t\} \\ \wedge & w_2 = t \end{array}\right) \quad \Rightarrow \; v_2 = w_2$$

$$\left(\begin{array}{ll} & x_1 \in \mathbb{B} - \{t\} \\ \wedge & v_2 = t \end{array}\right) \quad \wedge \quad \left(\begin{array}{ll} & x_1 \in \mathbb{B} - \{t\} \\ \wedge & w_2 = t \end{array}\right) \quad \Rightarrow \; v_2 = w_2$$

Verification Conditions

- merge FNFs
- VC simplification:
  - set expressions (constraints)
  - tautologies (conclusion)
  - contradictions (antecedent)

Example

$$P(v\colon \mathbb{Z}\ ;\ i\colon \mathbb{Z}\ /\ o\colon \mathbb{Z})\ \underline{\triangle}$$
$$\quad \square x\colon \mathbb{Z}\ \bullet\ i(x)\ /\ o(v)$$
$$\qquad :: P(x\ ;\ i\ /\ o)$$

$$Q(\langle w,s\rangle\colon \mathbb{Z}\times\mathbb{B}\ ;\ i\colon \mathbb{Z}\ /\ o\colon \mathbb{Z})\ \underline{\triangle}$$
$$\quad \square y\colon \mathbb{Z}\ \bullet\ i(y)\ /\ o(r_Q(y,w,s))$$
$$\qquad :: Q(e_Q(y,w,s)\ ;\ i\ /\ o)$$

where

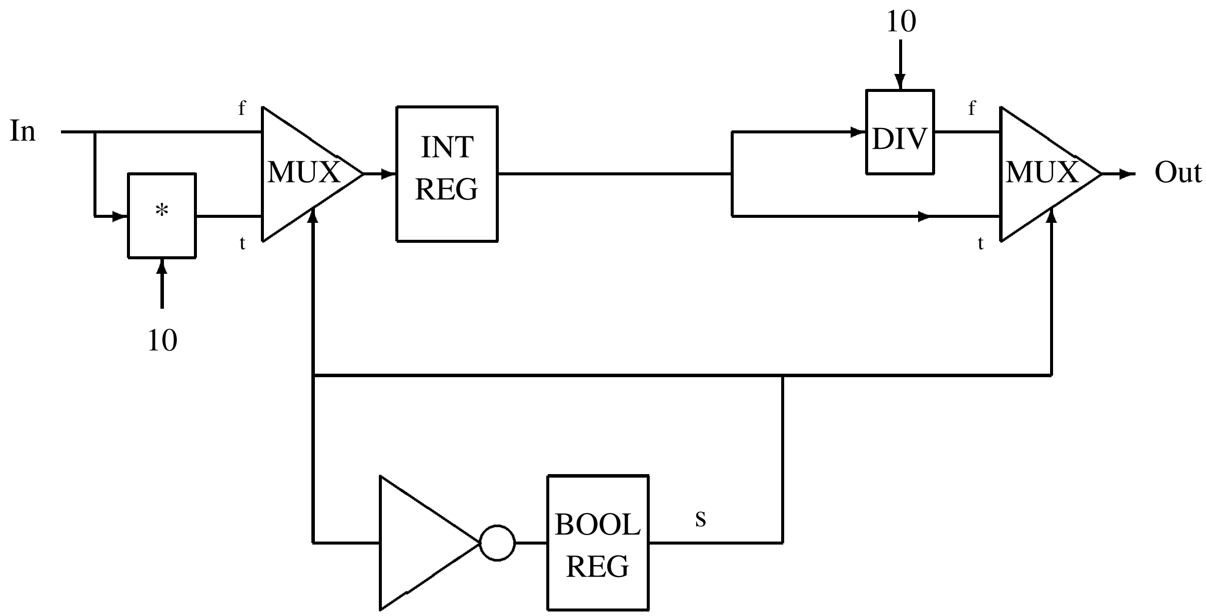
$r_Q(y,\langle w,s\rangle) =$ **case** s **in**
    $t : w$
  else $(w/10)$
  end

$e_Q(y,\langle w,s\rangle) =$ **case** s **in**
    $t : \langle (y*10,f)\rangle$
  else $\langle y,t\rangle$
  end

$i_P = 0$

$i_Q = \langle 0,t\rangle$

Example (continued)



Integer Latch Example: Process Q

Example (continued)

State Evolution (b):

$$
\begin{array}{rrcl}
 & r_P(x_1, sp_1) & = & r_Q(x_1, sq_1) \\
\wedge & sp_2 & = & e_P(x_1, sp_1) \\
\wedge & sq_2 & = & e_Q(x_1, sq_1) \\
\Rightarrow & r_P(x_2, sp_2) & = & r_Q(x_2, sq_2)
\end{array}
$$

Example (continued)

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

clause_3: THEOREM
 (
   (arith_var_94 = proj_2(inp_82)) AND
   (arith_var_93 = proj_2(stv_6))

 )
   IMPLIES

   (inp_82 = (d, div((10 * arith_var_94), 10)))


clause_4: THEOREM
 (
   (arith_var_94 = proj_2(inp_82)) AND
   (arith_var_93 = proj_2(stv_6))  AND

   (t = f)
 )
   IMPLIES

   (inp_82 = (d, (10 * arith_var_94)))
```

Theorem-Proving: PVS

VCs very similar to PVS format

- add function signatures
- represent sets as predicates
- decompose structured types
  e.g. tagged values
- add PVS axioms/declarations

(*PVS: from SRI International*)

Implementation

Design Environment (Harlequin Ltd)

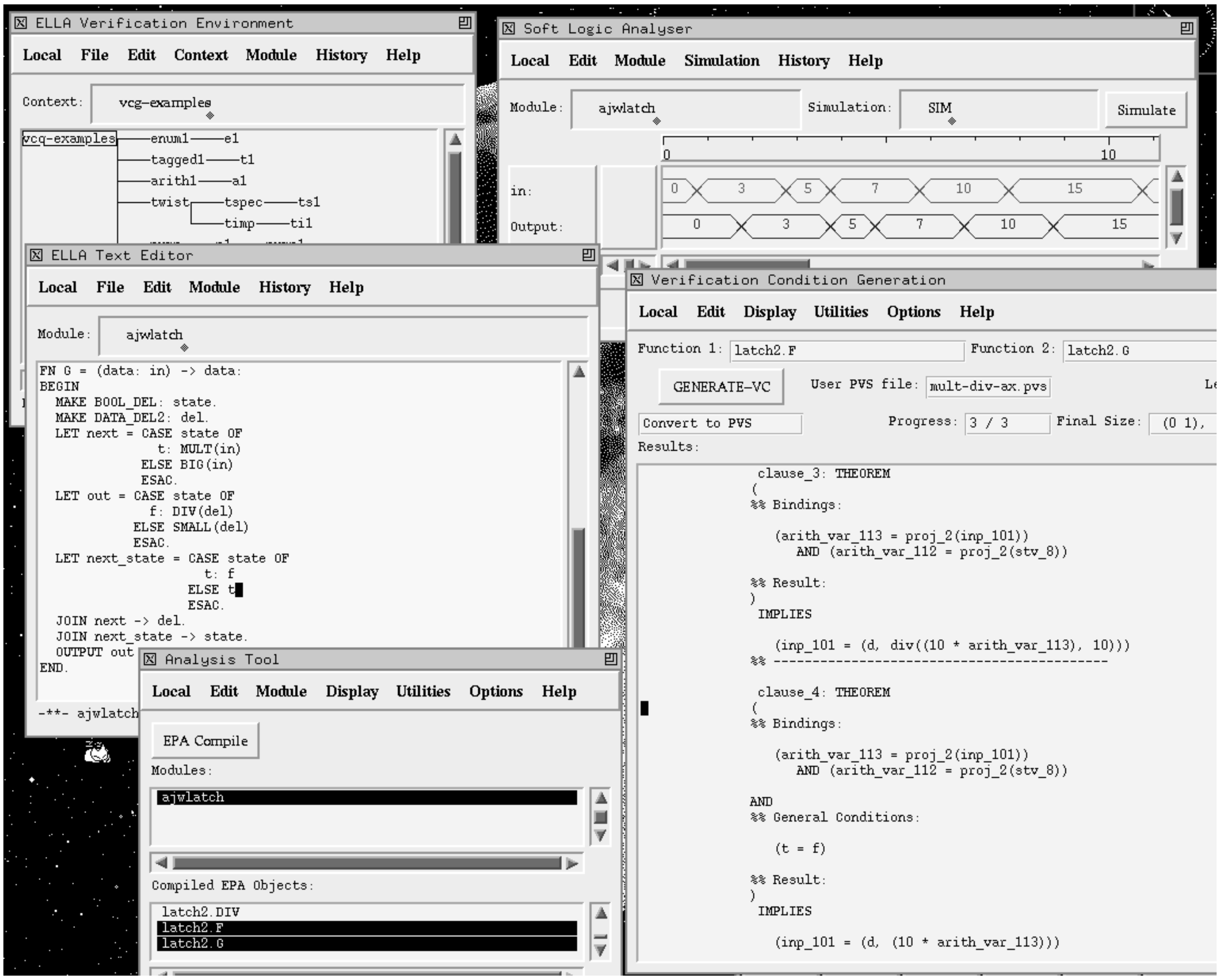
LispWorks (CLOS)

GT (Generic Term) (Manchester)

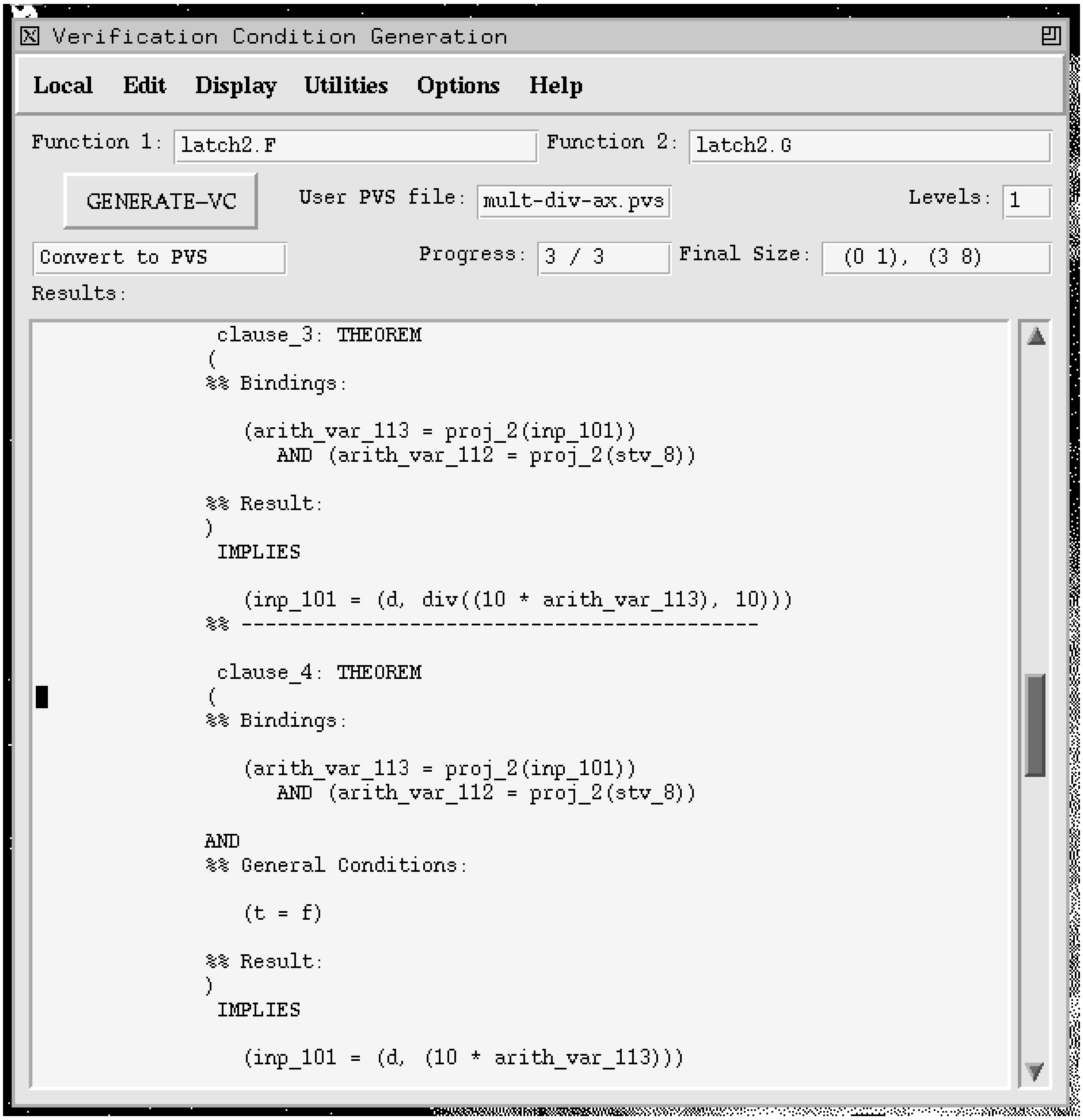
Embedded VCG, with GUI

Stand-alone PVS

Implementation (continued)



Implementation (continued)



Summary

Embedded verification tools

Semantics for commercial HDL

EPA normal-form: state-machine calculus

State Bisimulation

State Evolution

Verification Conditions

Theorem-Prover/Proof Assistant: PVS

Semi-automated approach:

- automatic theorem generation
- ‘simple’ conditions to prove

Data abstraction

Preserve design structure: useful for feedback

Designer information