EuroForm Workshop, Dresden, Germany

A Process Algebraic Approach to Synchronous Hardware Verification

Howard Barringer

Graham Gough

Brian Monahan*

Alan Williams

HaVe Group

Department of Computer Science

University of Manchester

funded by the UK SERC

3 – 5 February 1995

*Now with Harlequin Ltd., Alderley Edge.

CONTENTS

Previously... (Ile des Embiez)

Derive symbolic state machine:

- Process Algebraic Semantics: EPA
- EPA DFA Form for ELLA

Symbolic verification:

- (Strong) Bisimulation vs. State Bisimulation
- State Evolution
- Verification Condition Generation + Theorem-proving

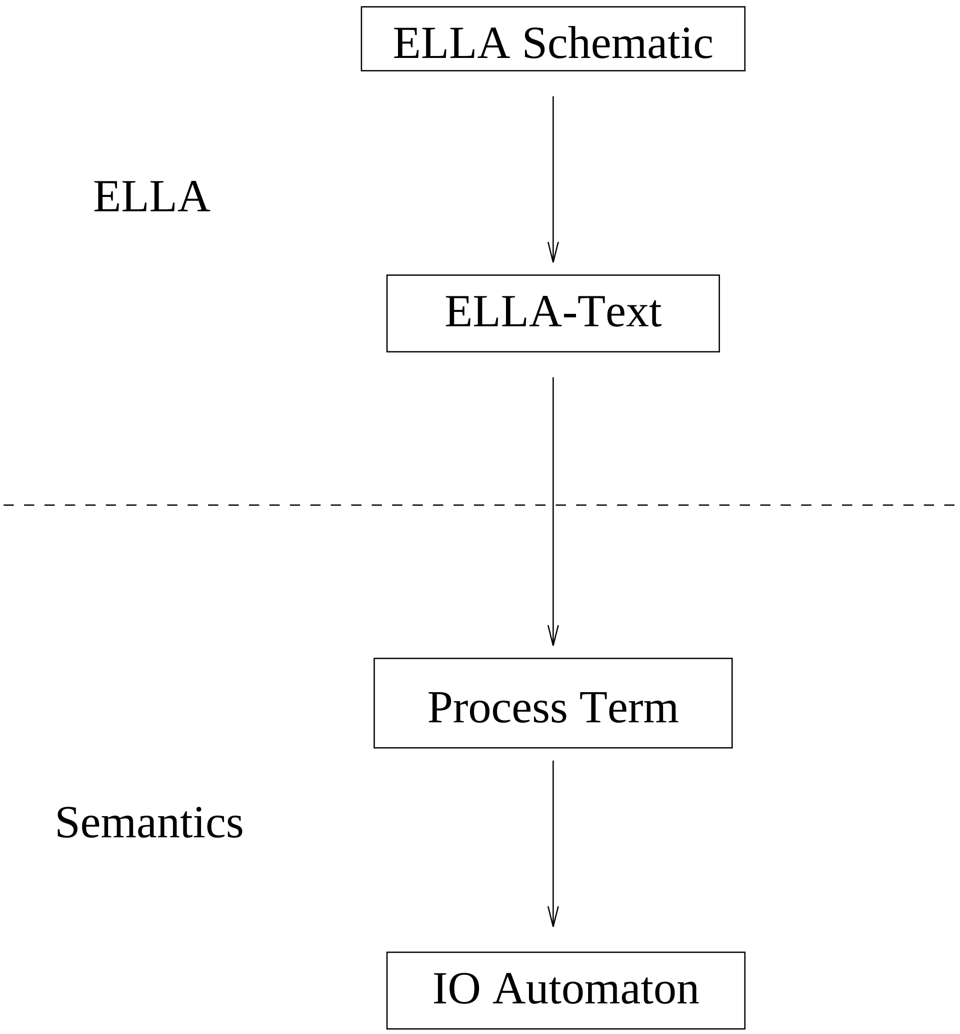
Background

IO Automaton semantics for ELLA HDL:

- compositional
- base semantic objects: finite state automata (transitions labelled by I/O pairs)
- input (output): map from signal names to values
- richly structured data

Need to reason at a higher level of abstraction.

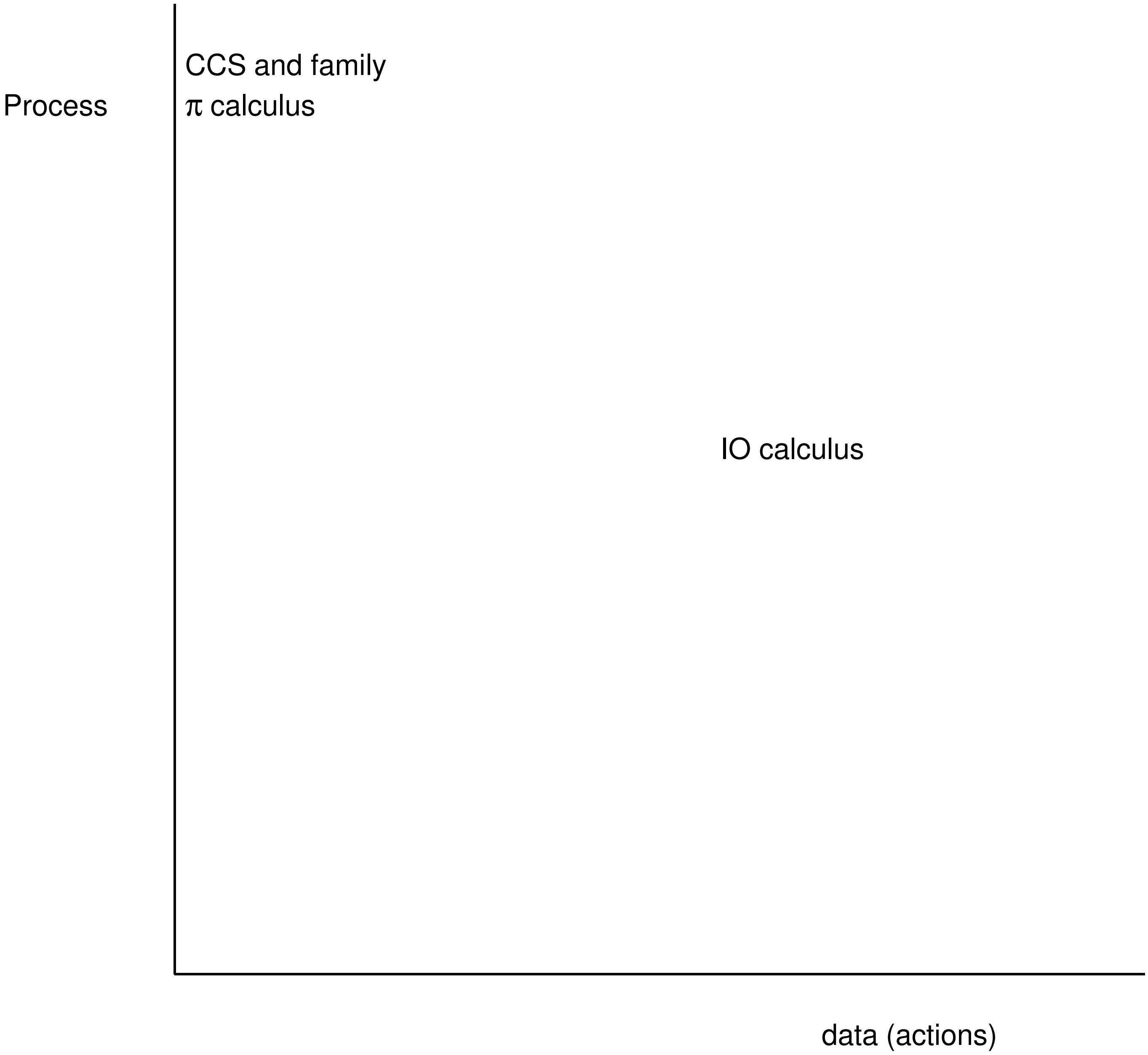
Design Translation



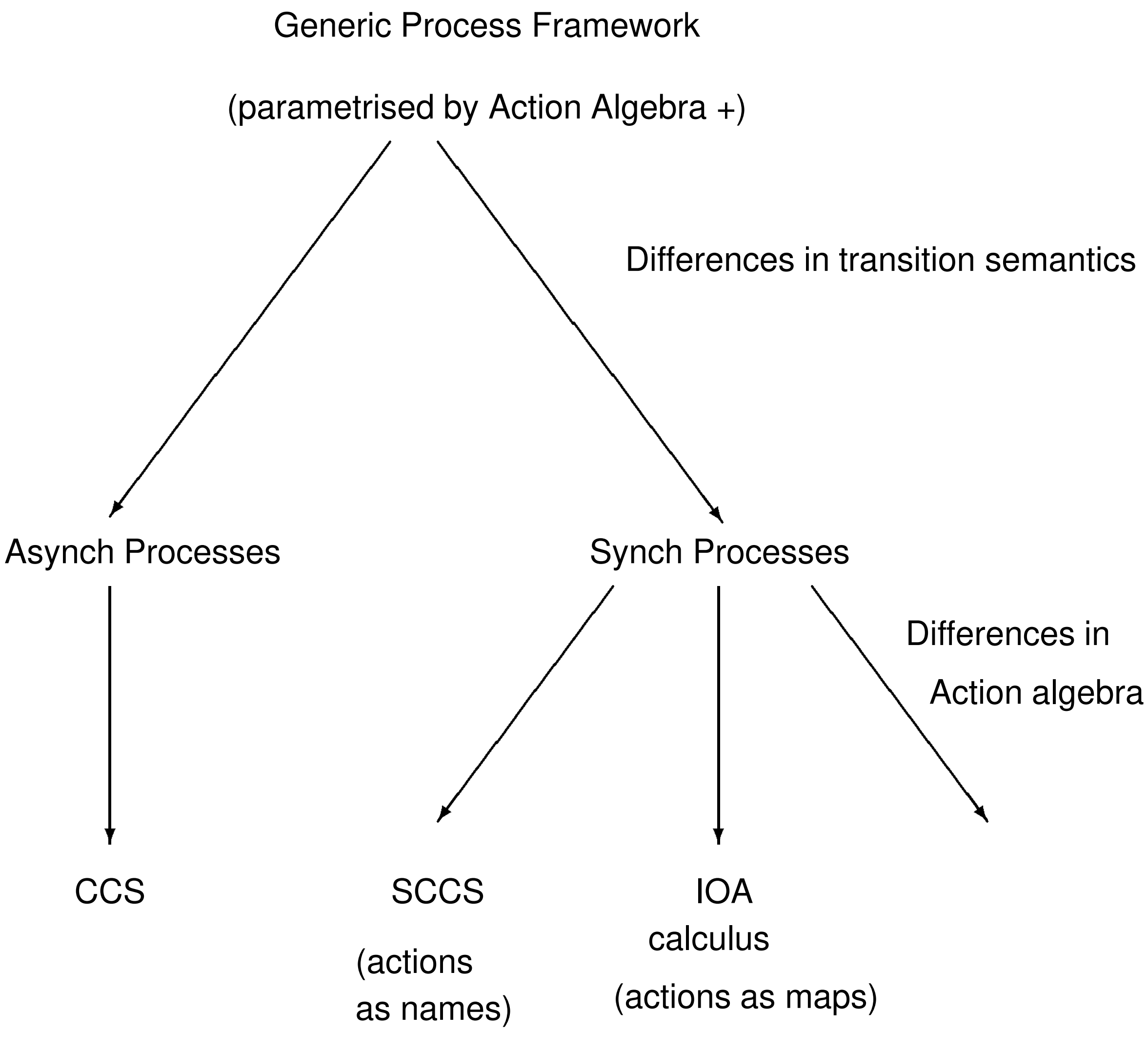
Why a Calculus?

- Preserves structure:
  - Automata are ‘flat’
  - BDDs merely encode flat enumerated versions
- Allows meta-level reasoning about process terms
- Provides a foundation for rewriting:
  - Low level : Semantic level
  - High level : ELLA design transforms
- Can build on existing work in process algebra to tackle other problems e.g., Graf/Steffen for modular verification

Process vs Data



General framework



Framework Definition

$$
\begin{array}{rcll}
\mathit{Context} & ::= & \vec{D} & \\
D \in \mathit{BodyDefn} & ::= & p(\vec{xs}) \;\underline{\triangle}\; B & \\
B \in \mathit{Body} & ::= & a :: B_1 & \mathit{Prefixing} \\
& | & g \rightarrow B_1 & \mathit{Guarding} \\
& | & \sum_{i \in I} B_i & \mathit{Meta\ Summation} \\
& | & p(\vec{t}) & \mathit{Continuation\ Call} \\
& | & B_1 \setminus_{\phi} t & \mathit{Generic\ Mutation} \\
& | & B_1 \,|_{\theta}\, B_2 & \mathit{Generic\ Composition} \\
xs \in \mathit{FormalParam} & ::= & x : s &
\end{array}
$$

$a \in Tm(\Sigma_{\mathsf{IND}} : Act), \quad g \in Tm(\Sigma_{\mathsf{IND}} : \mathbb{B})$

$s \in \mathcal{S}, \quad x \in Var$

$p \in BodyName$

$\theta \in CompOpr \qquad \theta : \; (Act^2 \rightarrow Act)$

$\phi \in MuRel \qquad \phi : \; (Act \times s \times Act \rightarrow \mathbb{B}),$

for some sort $s \in \mathcal{S}$

Semantic requirements on actions

Require that each action algebra Act contain the following:

$\iota : Act$	'Skip' action
$\boxtimes : Act$	'Improper' action

and that the following properties hold:

Composition:

associativity	$(a_1 \theta a_2) \theta a_3$	$= a_1 \theta (a_2 \theta a_3)$
commutativity	$a_1 \theta a_2$	$= a_2 \theta a_1$
zero	$\boxtimes \theta a$	$= \boxtimes$

Mutation:

zero $\phi(\boxtimes, x, a) \Rightarrow a = \boxtimes$

Semantic requirements on actions (continued)

So, we have

- Action sort, Act
- Interpretations for the constants $\iota, \boxtimes : Act$.
- *Composition* operators, from $CompOpr$ (satisfying above properties).
- *Mutation* relations, from $MuRel$, acting upon transitions (according to above requirement).

Transition rules : Closed body semantics

$$\boxed{\text{Vacuum}}\ \frac{}{C \xrightarrow{\boxtimes}_C C'} \qquad \boxed{\text{Skip}}\ \frac{}{C \xrightarrow{\iota}_C C}$$

$$\boxed{\text{Prefix}}\ \frac{}{a :: C \xrightarrow{a}_C C} \qquad \boxed{\text{Guard}}\ \frac{\vdash g, \quad C \xrightarrow{a}_C C'}{(g \to C) \xrightarrow{a}_C C'}$$

$$\boxed{\text{Meta Summation}}\ \frac{\text{for some } j \in I,\ C_j \xrightarrow{a}_C C'}{\sum_{i \in I} C_i \xrightarrow{a}_C C'}$$

Transition rules : Closed body semantics (continued)

$$\boxed{\text{Eq}}\ \frac{C_1(\vec{x}s)\ \underline{\triangle}\ C_2, \quad C_2[\vec{v}/\vec{x}] \xrightarrow{a}_C C'}{C_1(\vec{v}) \xrightarrow{a}_C C'}$$

$$\boxed{\text{Mutation}}\ \frac{C \xrightarrow{a}_C C', \quad \vdash \phi(a,t,b)}{C \setminus_\phi t \xrightarrow{b}_C C' \setminus_\phi t}$$

$$\boxed{\text{Product}}\ \frac{C_1 \xrightarrow{a_1}_C C_1', \quad C_2 \xrightarrow{a_2}_C C_2'}{C_1 \mid_\theta C_2 \xrightarrow{a_1\ \theta\ a_2}_C C_1' \mid_\theta C_2'}$$

Example (1): SCCS

Actions (for set L of Primitive actions):

$$
\begin{array}{lcll}
a \in Act & ::= & \boxtimes & \text{Inconsistent} \\
 & | & \iota & \text{Skip} \\
 & | & 1 & \text{Silent} \\
 & | & \alpha_1 \oplus \alpha_2 & \text{Action Sum} \\
 & | & b & \text{Primitive Action} \in L
\end{array}
$$

Let:

$$Act' \triangleq Act - \{\boxtimes, \iota\}$$

$$\langle Act', (\_ \star \_), \_^{-1}, 1 \rangle \text{ is an Abelian group}$$

Action Product on improper actions:

$$a \star \boxtimes = \boxtimes$$

$$a \star \iota = \boxtimes$$

Example (1): SCCS (continued)

Hiding:

$$hide_{SCCS}(\_,\_,\_) : Act \times SG_{Act'} \times Act \to \mathbb{B}$$

$$hide_{SCCS}(a, S, b) \underline{\triangle}$$

$$\textbf{if } a \in S \textbf{ then } b = a \textbf{ else } b = \boxtimes$$

note: $SG_{Act'}$ contains *subgroups* of $\langle Act', (\_ \star \_), \_^{-1}, 1\rangle$.

Operator families:

$$CompOpr_{SCCS} = \{\star\}$$
$$MuRel_{SCCS} = \{hide_{SCCS}\}$$

Example (2): CCS

Distinct elements $\{\tau, \boxtimes, \iota\} \subset Act$

$Act' \ \underline{\triangle}\ Act$—$\{\tau, \boxtimes, \iota\}$:

$$\overline{(\_)} : Act' \to Act'$$

$$\overline{a} \neq a$$

$$\overline{\overline{a}} = a$$

Let $x,\ y,\ z \in Act,\ a,\ b \in Act'$:

$$\begin{aligned}
x \star y &= y \star x \\
(x \star y) \star z &= x \star (y \star z) \\
x \star \iota &= x \\
x \star \boxtimes &= \boxtimes
\end{aligned}$$

$$\begin{aligned}
a \star \overline{a} &= \tau \\
a \star \tau &= \boxtimes \\
a \star b &= \boxtimes \text{ (where } b \neq \overline{a}\text{)}
\end{aligned}$$

Example (2): CCS (continued)

Mutation relation:

$$hide_{CCS}(\_,\_,\_) : Act \times Act' \times Act \rightarrow \mathbb{B}$$

$$hide_{CCS}(x,a,y) \triangleq$$
$$\textbf{if } x \in \{a,\ \overline{a}\} \textbf{ then } y = \boxtimes \textbf{ else } y = x$$

$$CompOpr_{CCS} = \{\star\}$$
$$MuRel_{CCS} = \{hide_{CCS}\}$$

$$\boxed{\text{Product}}\ \frac{C_1 \xrightarrow{a_1}_C C_1',\quad C_2 \xrightarrow{a_2}_C C_2'}{C_1 \mid C_2 \xrightarrow{a_1 \star a_2}_C C_1' \mid C_2'}$$

Example (3): IOA Calculus

- Algebraic abstraction of IO automaton models
  - Generalises *Mealy* machines from automata theory
- General SCCS framework as starting point:
  - Novel specialisation of action algebra
  - Not expressible in ‘standard’ SCCS without excessive modelling

Primitive IOA Calculus Syntax

$$
\begin{array}{llll}
P \in Proc & ::= & \gamma :: P & \text{Action Prefix} \\
& \mid & b \rightarrow P_1 & \textit{Guarding} \\
& \mid & \sum_{\vec{xs}} P_1 & \textit{Summation} \\
& \mid & P_1 + P_2 & \text{Choice}^* \\
& \mid & 0 & \text{Inaction}^* \\
& \mid & p(x_1, \ldots, x_k) & \text{Process Call} \\
& \mid & P \backslash nm & \text{Hiding} \\
& \mid & P_1 \mid P_2 & \text{Causal Product} \\
& \mid & P[nm_1 \mapsto nm_2] & \text{Renaming}^*
\end{array}
$$

$b : \mathbb{B}$

*derived operators

Action Product

Action product:

$$(\_ \star \_) : Act^2 \rightarrow Act$$

$$(\alpha_1/\beta_1) \star (\alpha_2/\beta_2) =$$
$$((\alpha_1 \oplus \alpha_2) \ominus (\beta_1 \oplus \beta_2))/(\beta_1 \oplus \beta_2))$$

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

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

with the following axioms on Actions:

$$(\alpha/\beta) = (\alpha \ominus \beta/\beta)$$
$$(\boxtimes/\beta) = \boxtimes = (\beta/\boxtimes)$$

Action Terms & IO Actions

Action terms:

$$
\begin{array}{lcll}
\gamma \in \mathit{Act} & ::= & \boxtimes & \text{Inconsistent} \\
 & | & (\alpha \,/\, \beta) & \text{IO Pairing}
\end{array}
$$

IO Actions:

$$
\begin{array}{lcll}
\alpha,\ \beta \in \mathit{IOAct} & ::= & \boxtimes & \text{Inconsistent} \\
 & | & \iota & \text{Skip Action} \\
 & | & \varepsilon & \text{Empty Action} \\
 & | & \alpha_1 \oplus \alpha_2 & \text{Action Sum} \\
 & | & \mathit{pa} & \text{Primitive Action } \in \mathit{Pa}
\end{array}
$$

Operations on IO Actions

- $\langle IOAct,\ \varepsilon,\ (\_ \oplus \_)\rangle$ is:
  - commutative, absorptive monoid

- $\boxtimes$ is an annihilator element

So, we have:

$$
\begin{aligned}
(\alpha_1 \oplus \alpha_2) \oplus \alpha_3 &= \alpha_1 \oplus (\alpha_2 \oplus \alpha_3) \\
\alpha_1 \oplus \alpha_2 &= \alpha_2 \oplus \alpha_1 \\
\varepsilon \oplus \alpha &= \alpha \\
\alpha \oplus \alpha &= \alpha \\
\boxtimes \oplus \alpha &= \boxtimes
\end{aligned}
$$

Maps

Need a more refined structure to capture maps

Add two relations:

$$
\begin{array}{rcll}
(\_ \preccurlyeq \_) & \subseteq & Act^2 & \text{Action Ordering} \\
(\_ \; \mathsf{CF} \; \_) & \subseteq & Act^2 & \text{Conflict Freedom}
\end{array}
$$

Derived relations:

$$
\begin{array}{ll}
(\_ \sqsubseteq \_) & \text{Action ordering, but taking conflict into account} \\
(\_ \simeq \_) & \text{Action congruence}
\end{array}
$$

Map Example

General algebraic structure captures maps:

- Action ordering on actions:
  $\{nm_1 : ty_1 \to v_1\} \preccurlyeq \{nm_2 : ty_2 \to v_2\} \;\Leftrightarrow$
  $\quad (v_2 \in ty_2) \;\Rightarrow$
  $\qquad ((nm_1 = nm_2) \;\wedge\; (v_1 = v_2) \;\wedge\; (ty_2 \subseteq ty_1))$

- Conflict relation on actions:
  $\{nm_1 : ty_1 \to v_1\}$ CF $\{nm_2 : ty_2 \to v_2\} \;\Leftrightarrow$
  $\quad (nm_1 \neq nm_2)$

- ε is the *empty map*, $\{\}$

- *action sum* is *map union*, $m_1 \cup m_2$

- ⊠ is represented by an *absence of consistent binding*

Elimination operator

Need notion of *elimination* operator $(\_ \ominus \_)$ between actions ...

$$(\_ \ominus \_) : Act^2 \rightarrow Act$$

(cf. *map subtraction*) satisfying various properties, such as:

$$\begin{aligned}
\alpha \ominus \alpha &= \varepsilon, \text{ for } (\alpha \neq \boxtimes) \\
(\alpha_1 \oplus \alpha_2) \ominus \beta &= (\alpha_1 \ominus \beta) \oplus (\alpha_2 \ominus \beta) \\
(\alpha \ominus \beta_1) \ominus \beta_2 &= \alpha \ominus (\beta_1 \oplus \beta_2)
\end{aligned}$$

Mutation

Mutation Relation:

$$
\begin{aligned}
&\mathit{hide\text{-}act}_{IO}\colon \mathit{Act} \times \mathcal{F}(\mathit{Name}) \to \mathit{Act}\\
&\mathit{hide\text{-}act}_{IO}(\boxtimes, \mathit{nms}) = \boxtimes\\
&\mathit{hide\text{-}act}_{IO}(\iota, \mathit{nms}) = \iota\\
&\mathit{hide\text{-}act}_{IO}(\alpha/\beta, \mathit{nms}) =\\
&\qquad \mathit{hide\text{-}io\text{-}act}(\alpha, \mathit{nms})/\mathit{hide\text{-}io\text{-}act}(\beta, \mathit{nms})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{hide\text{-}prim\text{-}act} : \mathit{pIOAct} \times \mathcal{F}(\mathit{Name}) \to (\mathit{pIOAct} \cup \{\varepsilon\})\\
&\mathit{hide\text{-}prim\text{-}act}([\mathit{nm}\colon \mathit{ty} \mapsto v], \mathit{nms}) =\\
&\quad \textbf{if } \mathit{nm} \in \mathit{nms} \textbf{ then } \varepsilon \textbf{ else } [\mathit{nm}\colon \mathit{ty} \mapsto v]
\end{aligned}
$$

Operator families:

$$
\begin{aligned}
&\mathit{CompOpr}_{IO} = \{\star\}\\
&\mathit{MulRel}_{IO} = \{\mathit{hide\text{-}act}_{IO}\}
\end{aligned}
$$

Conclusions

- Have developed a general framework which can be instantiated to different process calculi
- Emphasis on greater separation of process and data
- Used the framework to develop a calculus of IOAs
- Used this calculus to describe semantics of core ELLA
- Extend Framework theory
- Other Instantiations
- Related Work

World-Wide Web:

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