Types, Event Structures and the π-Calculus

Daniele Varacca, Nobuko Yoshida

Imperial College London

Sussex, November 30th 2005

Historical perspective

An unfair and myopic view of the last 40 years

Historical perspective

An unfair and myopic view of the last 40 years

Petri ['60]

Petri nets

Historical perspective

An unfair and myopic view of the last 40 years

Petri ['60]

Scott and Strachey ['70]

Denotational semantics - Domain theory

Historical perspective

An unfair and myopic view of the last 40 years

$$\begin{tikzcd}
\text{Petri ['60]} \arrow[dr] & \text{Scott and Strachey ['70]} \arrow[d] \\
& \text{Nielsen, Plotkin and Winskel ['80]}
\end{tikzcd}$$

Event structures

Historical perspective

An unfair and myopic view of the last 40 years

Petri ['60]

Scott and Strachey ['70]

Park and Milner ['80]

Nielsen, Plotkin and Winskel ['80]

Transition systems and bisimulation

Historical perspective

An unfair and myopic view of the last 40 years

$$
\begin{tikzcd}
\text{Petri ['60]} \arrow[dr] & \text{Scott and Strachey ['70]} \arrow[d] \\
\text{Park and Milner ['80]} \arrow[d] & \text{Nielsen, Plotkin and Winskel ['80]} \\
\text{Berry and Boudol ['90]} &
\end{tikzcd}
$$

Reduction semantics

Historical perspective

An unfair and myopic view of the last 40 years

Petri ['60]

Scott and Strachey ['70]

Park and Milner ['80]

Nielsen, Plotkin and Winskel ['80]

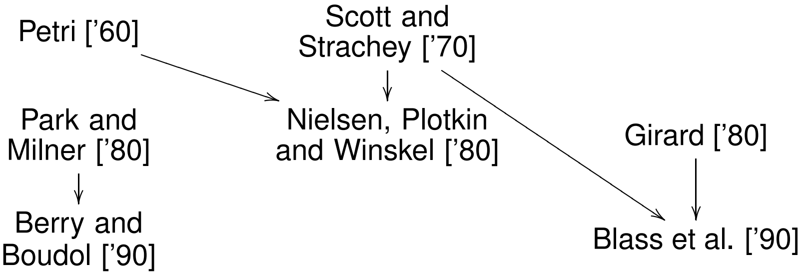
Girard ['80]

Berry and Boudol ['90]

Linear logic

Historical perspective

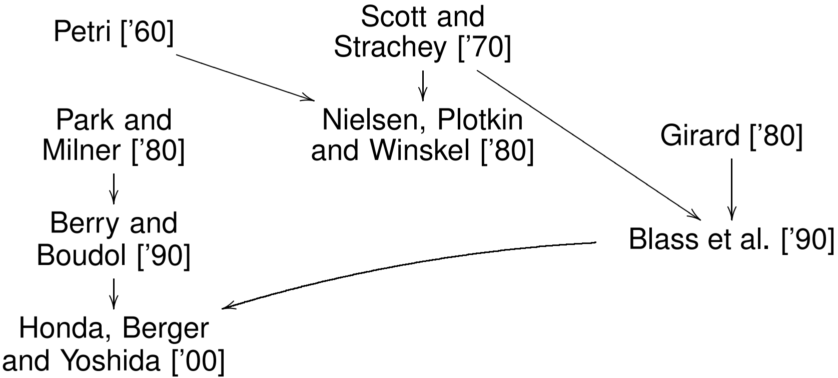
An unfair and myopic view of the last 40 years



Game semantics

Historical perspective

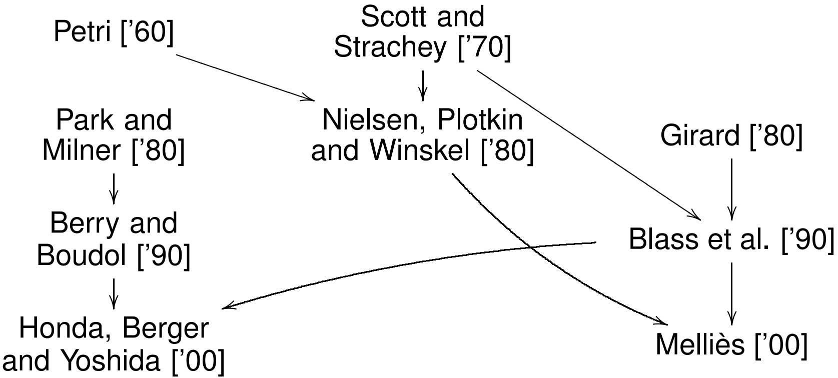
An unfair and myopic view of the last 40 years



Linearly typed π calculus

Historical perspective

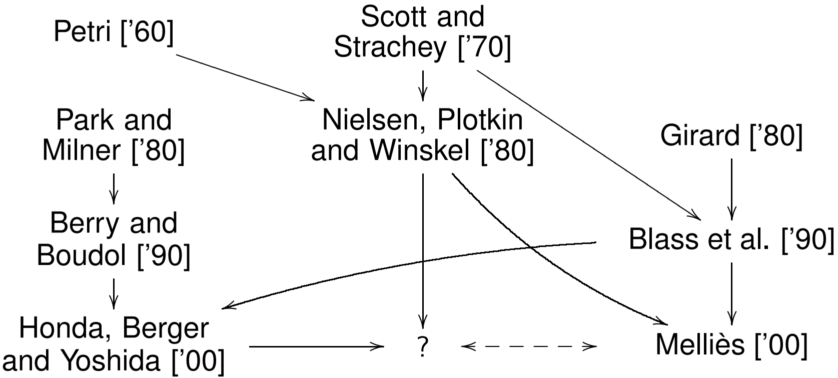
An unfair and myopic view of the last 40 years



True concurrent games

Historical perspective

An unfair and myopic view of the last 40 years



This talk

Typed π-calculus

We all know what the π-calculus is

$$x(\tilde{y}).P \mid \overline{x}\langle\tilde{z}\rangle.Q \;\longrightarrow\; P\{\tilde{z}/\tilde{y}\} \mid Q$$

Typed π-calculus

We all know what the π-calculus is

$$x(\tilde{y}).P \mid \overline{x}\langle\tilde{z}\rangle.Q \longrightarrow P\{\tilde{z}/\tilde{y}\} \mid Q$$

We consider a restricted version:
bound output only (“internal” mobility)

Typed π-calculus

We all know what the π-calculus is

$$x(\tilde{y}).P \mid \overline{x}(\tilde{y}).Q \;\longrightarrow\; (\nu\,\tilde{y})(P \mid Q)$$

We consider a restricted version:
bound output only ("internal" mobility)

Typed π-calculus

We all know what the π-calculus is

$$x(\tilde{y}).P \mid \overline{x}(\tilde{y}).Q \;\longrightarrow\; (\nu\,\tilde{y})(P \mid Q)$$

We consider a restricted version:
bound output only ("internal" mobility)

A linear type discipline:

(A) for each linear name there are a unique input and a unique output
(B) for each replicated name there is a unique stateless replicated input with zero or more dual outputs

Confluence

Linearly typed π is confluent

$$
\begin{tikzcd}
 & P \arrow[dl] \arrow[dr] & \\
P' \arrow[dr] & & P'' \arrow[dl] \\
 & Q &
\end{tikzcd}
$$

Road Map

1. Event Structures
   - Confusion Freeness
   - Conflict Freeness
2. Types
   - Syntax and Semantics
   - Typed Event Structures
3. Semantics of π
   - Syntax
   - Event Structure Semantics
   - Correspondence

Road Map

True concurrency

Standard “interleaving” semantics

- reduces parallelism to nondeterministic interleaving (“expansion law”)
- Labelled transition systems, reduction semantics

True concurrency

Standard “interleaving” semantics

- reduces parallelism to nondeterministic interleaving (“expansion law”)
- Labelled transition systems, reduction semantics

“True concurrent” models

- Represent explicitly causality, conflict, independence
- Petri nets, Mazurkiewicz traces, event structures

Event structures

An event structure is a partial order $\langle E, \leq \rangle$ together with a conflict relation $\smile$

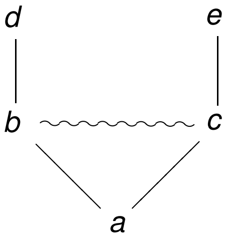
- order represents causal dependency
- conflict is irreflexive an symmetric
- conflict is "hereditary":

$$e_1 \smile e \text{ and } e_1 \leq e_2 \text{ implies } e_2 \smile e$$

A conflict is immediate if it is not inherited from another conflict

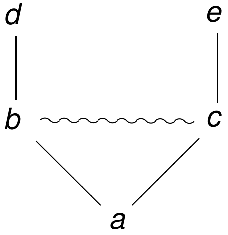
Event structures

Example



Event structures

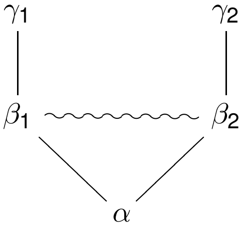
Example



Events can also be labelled: $\lambda : E \longrightarrow L$

Event structures

Example



Events can also be labelled: $\lambda : E \longrightarrow L$

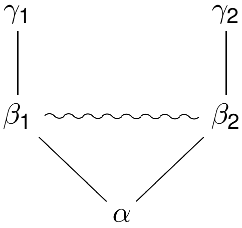
Operators on event structures

Prefixing $\alpha.\mathscr{E}$

$$\begin{tikzcd} \gamma_1 & & \gamma_2 \\ \beta_1 \arrow[u, dash] \arrow[rr, squiggly, dash] & & \beta_2 \arrow[u, dash] \end{tikzcd}$$

Operators on event structures

Prefixing $\alpha.\mathscr{E}$



Operators on event structures

Prefixed sum $\sum_{i \in I} \alpha_i.\mathscr{E}_i$

$$
\begin{tikzcd}
\gamma_1 & \gamma_2 \\
\beta_1 \arrow[u, dash] & \beta_2 \arrow[u, dash]
\end{tikzcd}
$$

Operators on event structures

Prefixed sum $\sum_{i \in I} \alpha_i.\mathscr{E}_i$

$$\begin{tikzcd}
\gamma_1 \arrow[d, dash] & & \gamma_2 \arrow[d, dash] \\
\beta_1 \arrow[d, dash] & & \beta_2 \arrow[d, dash] \\
\alpha_1 \arrow[rr, squiggly, dash] & & \alpha_2
\end{tikzcd}$$

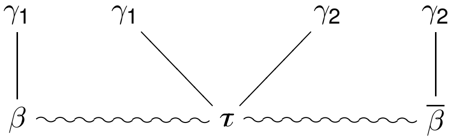
Operators on event structures

Parallel composition $\mathscr{E}_1 \| \mathscr{E}_2$

$$
\begin{tikzcd}
\gamma_1 \arrow[d, no head] & & \gamma_2 \arrow[d, no head] \\
\beta & & \overline{\beta}
\end{tikzcd}
$$

Operators on event structures

Parallel composition $\mathscr{E}_1 \| \mathscr{E}_2$



A complex construction involving synchronisation

Event structures and transition systems

Consider

- $\mathscr{E} = \langle E, \leq, \smile, \lambda \rangle$, a labelled event structure
- e, one of its minimal events

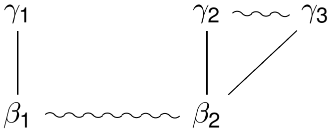
We define $\mathscr{E}\lfloor e$ as $\mathscr{E}$ minus event e, and minus all events that are in conflict with e

We can then generate a labelled transition system as follows: if $\lambda(e) = \beta$, then

$$\mathscr{E} \xrightarrow{\beta} \mathscr{E}\lfloor e$$

Event structures and transition systems

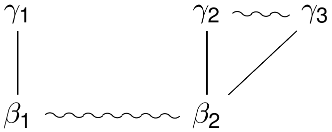
Example



An event structure $\mathscr{E}$

Event structures and transition systems

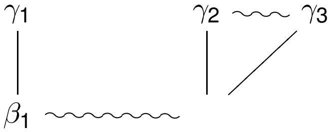
Example



Eliminate a minimal event *e* (labelled by β_2)

Event structures and transition systems

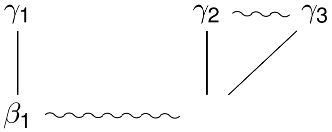
Example



Eliminate a minimal event *e* (labelled by β_2)

Event structures and transition systems

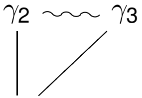
Example



And every event in conflict with it

Event structures and transition systems

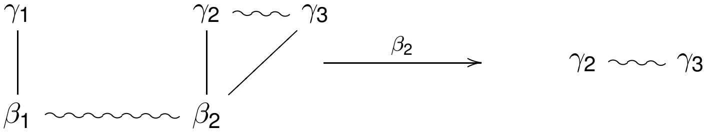
Example



And every event in conflict with it

Event structures and transition systems

Example



$$\mathcal{E} \xrightarrow{\beta_2} \mathcal{E}\lfloor e$$

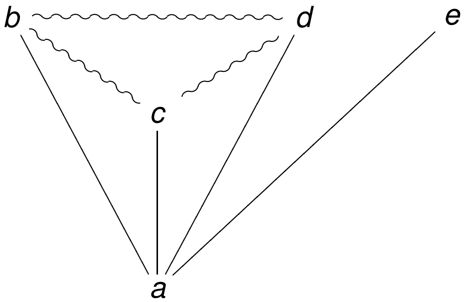
Confusion freeness

An event structure is confusion-free when

- "reflexive" immediate conflict is an equivalence
- any two events in immediate conflict have the same predecessors

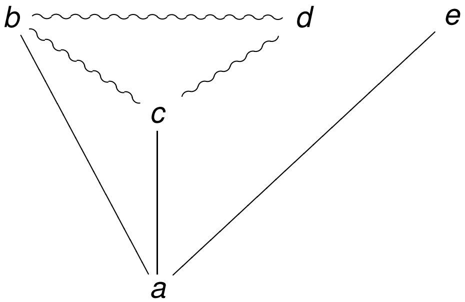
The equivalence classes are the cells
Cells represent local choices

Examples



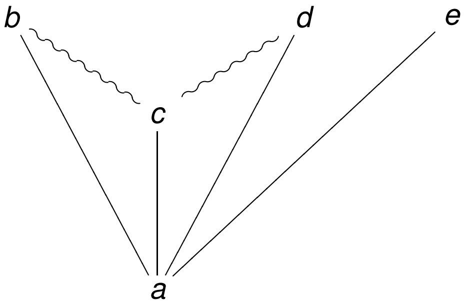
Confusion Free

Examples



Confusion!

Examples



Confusion!

Where confusion arises

Confusion arises from synchronisation

Consider $(\overline{a} \mid a)$

The event structure for this is

$$\begin{tikzcd} \overline{a} \arrow[dr, no head, squiggly] & & a \arrow[dl, no head, squiggly] \\ & \tau & \end{tikzcd}$$

Confusion - the choice is not local

Where confusion arises

Confusion arises from synchronisation
Consider $(\overline{a} \mid a)$
The event structure for this is

$$\begin{tikzcd} \overline{a} \arrow[dr, dash, squiggly] & & a \arrow[dl, dash, squiggly] \\ & \tau & \end{tikzcd}$$

Confusion - the choice is not local

Issue: how to perform synchronisation without introducing confusion

Conflict freeness

When the conflict relation is empty, the corresponding transition system is confluent
A special case of confusion freeness

Conflict freeness

When the conflict relation is empty, the corresponding transition system is confluent
A special case of confusion freeness

Idea: give a conflict free event structure semantics to the linear π-calculus
Issues:

- difficult to handle name generation
- hidden conflicts appear

The post office

Example:

- Stateless replicated resource: post office $!a.P$
- Clients: customers $\overline{a}.C$

Every customer wants to send a letter a

The post office

The process $\overline{a}.D \mid \overline{a}.N \mid !a.P$ is confluent

$$\begin{tikzcd}
& \overline{a}.D \mid \overline{a}.N \mid !a.P \arrow[dl] \arrow[dr] & \\
D \mid P \mid \overline{a}.N \mid !a.P \arrow[dr] & & N \mid P \mid \overline{a}.D \mid !a.P \arrow[dl] \\
& N \mid P \mid D \mid P \mid !a.P &
\end{tikzcd}$$

The post office

Situation 1: two customers, one till
A conflict to resolve: who goes first?
Eventually, it does not matter, but the two events are not independent

The post office

Situation 1: two customers, one till
A conflict to resolve: who goes first?
Eventually, it does not matter, but the two events are not independent

Situation 2: two customers, infinitely many identical tills
if the two customers want to go to the same till, there is a conflict

The post office

Situation 1: two customers, one till
A conflict to resolve: who goes first?
Eventually, it does not matter, but the two events are not independent

Situation 2: two customers, infinitely many identical tills
if the two customers want to go to the same till, there is a conflict

Situation 3: one customer, infinitely many identcal tills
the customer has to choose which till to go to

The post office

Solution: no conflict arises if every possible customer is assigned a spefic till in advance

Road Map

Types for event structures

$$
\begin{array}{lcll}
\Gamma, \Delta & ::= & y_1 : \sigma_1, \ldots, y_n : \sigma_n & \text{type environment} \\
\tau, \sigma & ::= & \&_{i \in I} \Gamma_i & \text{branching} \\
& \mid & \bigoplus_{i \in I} \Gamma_i & \text{selection} \\
& \mid & \bigotimes_{i \in I} \Gamma_i & \text{server} \\
& \mid & \invamp_{i \in I} \Gamma_i & \text{client} \\
& \mid & \updownarrow & \text{closed type}
\end{array}
$$

Linearity condition: no name appears more than once

Composing environments

Notion of matching of types

- A branching type & matches the dual selection types $\oplus$, and the residual type is $\updownarrow$

Composing environments

Notion of matching of types

- A branching type & matches the dual selection types $\oplus$, and the residual type is $\updownarrow$
- A server type $\otimes$ matches a client type $\invamp$ if all requests correspond to an available resource. The residual is again a server type $\otimes$ that records which resources are still available

Composing environments

Notion of matching of types

- A branching type & matches the dual selection types $\oplus$, and the residual type is $\updownarrow$
- A server type $\otimes$ matches a client type $⅋$ if all requests correspond to an available resource. The residual is again a server type $\otimes$ that records which resources are still available
- Two environments Γ_1, Γ_2 be composed if the types of the common names match

Composing environments

Notion of matching of types

- A branching type & matches the dual selection types $\oplus$, and the residual type is $\updownarrow$
- A server type $\otimes$ matches a client type $\parr$ if all requests correspond to an available resource. The residual is again a server type $\otimes$ that records which resources are still available
- Two environments Γ_1, Γ_2 be composed if the types of the common names match
- Such names are given the residual type by the resulting environment $\Gamma_1 \odot \Gamma_2$

Type environments

Example

- $\tau_1 = \&_{i\in\{1,2\}}(x_i : \bigoplus_{j\in J})$
- $\tau_2 = \bigoplus_{i\in\{1,2\}}(x_i : \&_{j\in J})$
- $\sigma_1 = \invamp_{i\in\{1\}}(y_i : \updownarrow)$
- $\sigma_2 = \bigotimes_{i\in\{1,2\}}(y_i : \updownarrow)$

Type environments

Example

- $\tau_1 = \&_{i\in\{1,2\}}(x_i : \bigoplus_{j\in J})$
- $\tau_2 = \bigoplus_{i\in\{1,2\}}(x_i : \&_{j\in J})$
- $\sigma_1 = \bigparr_{i\in\{1\}}(y_i : \updownarrow)$
- $\sigma_2 = \bigotimes_{i\in\{1,2\}}(y_i : \updownarrow)$

- τ_1 matches τ_2
- the residual type is $\updownarrow$

Type environments

Example

- $\tau_1 = \&_{i\in\{1,2\}}(x_i : \bigoplus_{j\in J})$
- $\tau_2 = \bigoplus_{i\in\{1,2\}}(x_i : \&_{j\in J})$
- $\sigma_1 = ⅋_{i\in\{1\}}(y_i : \updownarrow)$
- $\sigma_2 = \bigotimes_{i\in\{1,2\}}(y_i : \updownarrow)$

- σ_1 matches σ_2
- the residual type is $\bigotimes_{i\in\{2\}}(y_i : \updownarrow)$

Type environments

Example

- $\tau_1 = \&_{i\in\{1,2\}}(x_i : \bigoplus_{j\in J})$
- $\tau_2 = \bigoplus_{i\in\{1,2\}}(x_i : \&_{j\in J})$
- $\sigma_1 = \bigparr_{i\in\{1\}}(y_i : \updownarrow)$
- $\sigma_2 = \bigotimes_{i\in\{1,2\}}(y_i : \updownarrow)$

- $\Gamma_1 = a : \tau_1, b : \sigma_1,$
- $\Gamma_2 = a : \tau_2, b : \sigma_2$

Type environments

Example

- $\tau_1 = \&_{i \in \{1,2\}}(x_i : \bigoplus_{j \in J})$
- $\tau_2 = \bigoplus_{i \in \{1,2\}}(x_i : \&_{j \in J})$
- $\sigma_1 = \invamp_{i \in \{1\}}(y_i : \updownarrow)$
- $\sigma_2 = \bigotimes_{i \in \{1,2\}}(y_i : \updownarrow)$

- $\Gamma_1 = a : \tau_1, b : \sigma_1,$
- $\Gamma_2 = a : \tau_2, b : \sigma_2$
- $\Gamma_1 \odot \Gamma_2 = a : \updownarrow, b : \bigotimes_{i \in \{2\}}(y_i : \updownarrow)$

Labelled event structures

Labels:

$$
\begin{array}{rclll}
\alpha, \beta & ::= & x\mathtt{in}_i\langle\tilde{y}\rangle & \text{branching} \\
& \mid & \overline{x}\mathtt{in}_i\langle\tilde{y}\rangle & \text{selection} \\
& \mid & x\langle\tilde{y}\rangle & \text{server} \\
& \mid & \overline{x}\langle\tilde{y}\rangle & \text{client} \\
& \mid & \tau & \text{synchronisation}
\end{array}
\qquad
\begin{array}{rcl}
\tau & ::= & (x, \overline{x})\mathtt{in}_i\langle\tilde{y}\rangle \\
& \mid & (x, \overline{x})\langle\tilde{y}\rangle
\end{array}
$$

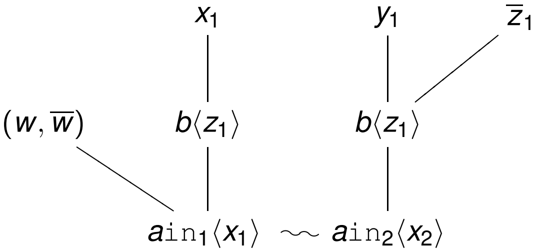
Typing via morphisms

An event structure $\mathscr{E}$ is well typed in Γ if

- $\mathscr{E}$ is confusion free
- cells are partitioned in branching, selection, client, server and synchronisation cells
- all the non-synchronisation events of are represented in Γ
- causality in $\mathscr{E}$ refines name causality of Γ

Technically: via morphisms of the category of event structures

A typed event structure



$$\mathscr{E} \quad \triangleright \quad a : \mathop{\Large\&}_{i\in\{1,2\}} (x_i : \mathop{\Large\&}_{k\in\{1\}})$$

$$b : \bigotimes_{j\in\{1\}} (z_j : \bigparr_{l\in\{1\}})$$

Typed event structures

Properties

- Typed event structures are confusion free (by definition)
- Prefixing, prefixed sum and parallel composition preserve typing

In particular parallel composition of typed event structures is confusion free

Typed event structures

Properties

- Typed event structures are confusion free (by definition)
- Prefixing, prefixed sum and parallel composition preserve typing

In particular parallel composition of typed event structures is confusion free

Theorem [Parallel composition]

If $\mathscr{E}_1 \triangleright \Gamma_1$ and $\mathscr{E}_2 \triangleright \Gamma_2$ and $\Gamma_1 \odot \Gamma_2$ is defined, then

$$(\mathscr{E}_1 \| \mathscr{E}_2) \setminus S \triangleright \Gamma_1 \odot \Gamma_2$$

Typed event structures

Properties

- Typed event structures are confusion free (by definition)
- Prefixing, prefixed sum and parallel composition preserve typing

In particular parallel composition of typed event structures is confusion free

Theorem [Parallel composition]

If $\mathcal{E}_1 \triangleright \Gamma_1$ and $\mathcal{E}_2 \triangleright \Gamma_2$ and $\Gamma_1 \odot \Gamma_2$ is defined, then

$$(\mathcal{E}_1 \| \mathcal{E}_2) \setminus S \triangleright \Gamma_1 \odot \Gamma_2$$

(S is the set of names not allowed by the new environment)

Typed event structures

When branching and selection types are trivial:

- Typed event structures are conflict free
- Prefixing, and parallel composition preserve typing

In particular parallel composition of typed event structures is conflict free

Road Map

The syntax

π processes

P	$::=$	$x \,\&_{i \in I}\, \mathtt{in}_i(\tilde{y}_i).P_i$	branching
	$\mid$	$\overline{x}\mathtt{in}_j(\tilde{y}).P$	selection
	$\mid$	$!x(\tilde{y}).P$	server
	$\mid$	$\overline{x}(\tilde{y}).P$	client
	$\mid$	$P \mid Q$	parallel
	$\mid$	$(\nu\, x)P$	restriction
	$\mid$	$\mathbf{0}$	inaction

The syntax

π processes

$$
\begin{array}{rcll}
P & ::= & x \binampersand_{i\in I} \mathtt{in}_i(\tilde{y}_i).P_i & \text{branching} \\
 & \mid & \overline{x} \bigoplus_{i\in I} \mathtt{in}_i(\tilde{y}_i).P_i & \text{selection} \\
 & \mid & !x(\tilde{y}).P & \text{server} \\
 & \mid & \overline{x}(\tilde{y}).P & \text{client} \\
 & \mid & P \mid Q & \text{parallel} \\
 & \mid & (\nu\, x)P & \text{restriction} \\
 & \mid & \mathbf{0} & \text{inaction}
\end{array}
$$

The types

π types

$$
\begin{array}{lcll}
\sigma & ::= & \&_{i \in I}\, (\tilde{\sigma}_i)^{\downarrow} & \text{branching} \\
 & | & \bigoplus_{i \in I}\, (\tilde{\sigma}_i)^{\uparrow} & \text{selection} \\
 & | & (\tilde{\sigma})^{\mathbf{!}} & \text{server} \\
 & | & (\tilde{\sigma})^{\mathbf{?}} & \text{client} \\
\tau & ::= & \sigma \mid \updownarrow &
\end{array}
$$

Environments compose in a similar way as event structure environments

Examples

$$\overline{a}.b \mid \overline{a}.c \mid a$$

This is not typable as *a* appears twice as output

Examples

$$b.\overline{a} \mid c.\overline{b} \mid a.(\overline{c} \mid \overline{e})$$

This is typable since each channel appears at most once as input and output

Examples

$$\overline{a}.(b \oplus c) \mid a.(\overline{d} \,\&\, \overline{e})$$

This process is typable, and contains nondeterminism:

$$Q_3 \longrightarrow (b \mid \overline{d})$$

$$Q_3 \longrightarrow (c \mid \overline{e})$$

Examples

$$!\, b.\overline{a} \mid !\, b.\overline{c}$$

This is not typable as there are two different servers associated with b

Examples

$$!\, b.\overline{a} \mid \overline{b} \mid !\, c.\overline{b}$$

This is typable: the two clients on *b* are associated to a unique server

Typed transition system

Operational semantics

- As usual $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma'$
- The transition must be allowed by the environment
- (The enviroment performs implicit restricitions)

Event structure semantics of π

The semantics has the form $[\![P \triangleright \Gamma]\!]^{\Delta}$, where Δ is an event structure environment

Event structure semantics of π

The semantics has the form $[\![P \triangleright \Gamma]\!]^{\Delta}$, where Δ is an event structure environment

Δ fixes a choice of the newly generated names

Event structure semantics of π

The semantics has the form $[\![P \triangleright \Gamma]\!]^{\Delta}$, where Δ is an event structure environment

Δ fixes a choice of the newly generated names
Δ assigns each client a specific instance of its server

Event structure semantics of π

$$\llbracket \overline{a} \bigoplus_{i \in I} \mathtt{in}_i(y_i).P_i \triangleright \Gamma, a : \bigoplus_{i \in I}(\tau_i) \rrbracket^{\Delta, a : \bigoplus_{i \in I} z_i : \hat{\tau}_i}$$

$$=$$

$$\sum_{i \in I} \overline{a}\mathtt{in}_i\langle z_i \rangle . \llbracket P_i[z_i/y_i] \triangleright \Gamma, z_i : \tau_i \rrbracket^{\Delta, z_i : \hat{\tau}_i}$$

Event structure semantics of π

$$[\![!a(y).P \triangleright \Gamma, a : (\tau)^{!}]\!]^{\Delta, a : \bigotimes_{k \in K}(y^k : \hat{\tau}^k)}$$

$$=$$

$$\|_{k \in K} a\langle y^k \rangle . [\![P[y^k/y] \triangleright \Gamma[y^k/y]]\!]^{\Delta^k, y^k : \hat{\tau}^k}$$

Event structure semantics of π

$$\llbracket P_1 \mid P_2 \triangleright \Gamma_1 \odot \Gamma_2 \rrbracket^{\Delta_1 \odot \Delta_2}$$

$$=$$

$$(\llbracket P_1 \triangleright \Gamma_1 \rrbracket^{\Delta_1} \| \llbracket P_2 \triangleright \Gamma_2 \rrbracket^{\Delta_2}) \setminus S$$

Event structure semantics of π

The interpretation functions are partial functions: for the wrong choice of Δ_1, Δ_2, the interpretation of the parallel composition could be undefined, because $\Delta_1 \odot \Delta_2$ may be undefined

Event structure semantics of π

The interpretation functions are partial functions: for the wrong choice of Δ_1, Δ_2, the interpretation of the parallel composition could be undefined, because $\Delta_1 \odot \Delta_2$ may be undefined

It is always possible to find suitable Δ_1, Δ_2
We perform α-conversion "at compile time"

Event structure semantics of π

The interpretation functions are partial functions: for the wrong choice of Δ_1, Δ_2, the interpretation of the parallel composition could be undefined, because $\Delta_1 \odot \Delta_2$ may be undefined

Theorem: [Event structure semantics]
For every judgement $P \triangleright \Gamma$ in the π-calculus, there exists an environment Δ such that $[\![P \triangleright \Gamma]\!]^{\Delta}$ is defined
Also: $[\![P \triangleright \Gamma]\!]^{\Delta} \triangleright \Delta$

Correspondence

Correspondence between transition system and event structure:

Theorem: [Operational correspondence]

If $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma'$, then $[\![P \triangleright \Gamma]\!]^{\Delta} \xrightarrow{\beta} \cong [\![P' \triangleright \Gamma']\!]^{\Delta'}$

Correspondence

Correspondence between transition system and event structure:

Theorem: [Operational correspondence]

If $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma'$, then $[\![P \triangleright \Gamma]\!]^{\Delta} \xrightarrow{\beta} \cong [\![P' \triangleright \Gamma']\!]^{\Delta'}$

If $[\![P \triangleright \Gamma]\!]^{\Delta} \xrightarrow{\beta} \mathscr{E}'$, then there exists P' such that $P \triangleright \Gamma \xrightarrow{\beta} P' \triangleright \Gamma'$ and $[\![P' \triangleright \Gamma']\!]^{\Delta'} \cong \mathscr{E}'$

Summary

What we have done

- First typing system for event structures
- Typing system for true concurrent behavioural properties
- First *explicit* event structure semantics of π

Summary

What we have done

- First typing system for event structures
- Typing system for true concurrent behavioural properties
- First *explicit* event structure semantics of π

What we will do

- Probabilistic event structures
- Connections with true concurrent games
- Connections with Beffara's thesis