Event Structure Semantics for Nominal Calculi

Roberto Bruni

Dipartimento di Informatica
Università di Pisa

CONCUR 2006
Bonn, August 27–30, 2006

A joint work with:

H. Melgratti and U. Montanari

Disclaim: Lightweight presentation of technical details, where intuition is deliberately advantaged over precision

Event Structure Semantics for Nominal Calculi

Roberto Bruni

Dipartimento di Informatica
Università di Pisa

CONCUR 2006
Bonn, August 27–30, 2006

A joint work with:

H. Melgratti and U. Montanari

Disclaim: Lightweight presentation of technical details, where intuition is deliberately advantaged over precision

Outline

True Concurrent Semantics

Models of concurrency

- *interleaving*: concurrency reduces to nondeterministic choices
- *conflict*: focus on choice points
- *causal*: causal dependencies are explicit
- *concurrent*: multiple actions are allowed

(Prime) Event Structures

Suitable domain of events where conflicts, causality and concurrency are accounted for and related altogether.

A Curious Fact of Life

In concurrency theory, interleaving semantics are far more frequently studied than true concurrent ones (even at CONCUR).

True Concurrent Semantics

Models of concurrency

- *interleaving*: concurrency reduces to nondeterministic choices
- *conflict*: focus on choice points
- *causal*: causal dependencies are explicit
- *concurrent*: multiple actions are allowed

(Prime) Event Structures

Suitable domain of events where conflicts, causality and concurrency are accounted for and related altogether.

A Curious Fact of Life

In concurrency theory, interleaving semantics are far more frequently studied than true concurrent ones (even at CONCUR).

True Concurrent Semantics

Models of concurrency

- *interleaving*: concurrency reduces to nondeterministic choices
- *conflict*: focus on choice points
- *causal*: causal dependencies are explicit
- *concurrent*: multiple actions are allowed

(Prime) Event Structures

Suitable domain of events where conflicts, causality and concurrency are accounted for and related altogether.

A Curious Fact of Life

In concurrency theory, interleaving semantics are far more frequently studied than true concurrent ones (even at CONCUR).

Event Structures and Nominal Calculi

Event Structures and π

Daniele Varacca and Nobuko Yoshida recently defined the (**direct**, typed, sound) event structure semantics of (a **linearly typed version** of) Sangiorgi's π**I-calculus** [MFPS'06]

- Glynn Winskel's original event structure semantics of CCS is extended to π-calculi for the *first* time
- the considered fragment is expressive enough to encode the typed λ-calculus (fully abstractly)
- compile-time α-conversion is allowed
- typed event structures guarantees confusion freeness (closed under parallel composition)

Event Structures and Nominal Calculi

Event Structures and π

Daniele Varacca and Nobuko Yoshida recently defined the (**direct**, typed, sound) event structure semantics of (a **linearly typed version** of) Sangiorgi's π**I-calculus** [MFPS'06]

- Glynn Winskel's original event structure semantics of CCS is extended to π-calculi for the *first* time
- the considered fragment is expressive enough to encode the typed λ-calculus (fully abstractly)
- compile-time α-conversion is allowed
- typed event structures guarantees confusion freeness (closed under parallel composition)

Indirect Event Structures Semantics?

Via Petri Nets Encoding

- Engelfriet [CONCUR'93]
- Busi and Gorrieri [CONCUR'95]
- Buscemi and Sassone (for join) [FOSSACS'01]
- Devillers, Klaudel, Koutny [FORTE'04]

Via Graph Rewriting Encoding

- Montanari and Pistore [MFPS'95]
- Gadducci and Montanari (for ambients) [MFPS'01]
- Gadducci [APLAS'03]

Devil is in the Detail!

Not as immediately applicable (and satisfactory) as it may seem!

Indirect Event Structures Semantics?

Via Petri Nets Encoding

- Engelfriet [CONCUR'93]
- Busi and Gorrieri [CONCUR'95]
- Buscemi and Sassone (for join) [FOSSACS'01]
- Devillers, Klaudel, Koutny [FORTE'04]

Via Graph Rewriting Encoding

- Montanari and Pistore [MFPS'95]
- Gadducci and Montanari (for ambients) [MFPS'01]
- Gadducci [APLAS'03]

Devil is in the Detail!

Not as immediately applicable (and satisfactory) as it may seem!

Indirect Event Structures Semantics?

Via Petri Nets Encoding

- Engelfriet [CONCUR'93]
- Busi and Gorrieri [CONCUR'95]
- Buscemi and Sassone (for join) [FOSSACS'01]
- Devillers, Klaudel, Koutny [FORTE'04]

Via Graph Rewriting Encoding

- Montanari and Pistore [MFPS'95]
- Gadducci and Montanari (for ambients) [MFPS'01]
- Gadducci [APLAS'03]

Devil is in the Detail!

Not as immediately applicable (and satisfactory) as it may seem!

A Nice Event Structures Semantics

Winskel's Construction

$$\textbf{Safe Nets} \underset{\mathcal{U}}{\overset{\perp}{\rightleftarrows}} \textbf{Occurrence Nets} \underset{\mathcal{E}}{\overset{\mathcal{N}}{\underset{\perp}{\leftrightarrows}}} \textbf{Prime Event Structures} \underset{\mathcal{L}}{\overset{\mathcal{P}}{\underset{\sim}{\leftrightarrows}}} \textbf{Domains}$$

- **Categories** instead of sets (objects related via morphisms)
- **Functorial** semantics (morphisms are preserved)
- **Backward** constructions (and they are still functors)
- Forward and backward constructions form **adjunctions** (universal property witnesses optimality, preservation of (co)limits)...
- ... more precisely, **coreflections** (particularly nice adjunctions, where a natural iso establishes an equivalence between the denotational domain and a full subcategory of computational models)
- Later generalized to (semi-weighted) P/T nets (Sassone's PhD Thesis)

A Nice Event Structures Semantics

Winskel's Construction

$$\begin{tikzcd}[column sep=large]
\textbf{Safe Nets} \arrow[r, shift right, "\mathcal{U}"'] \arrow[r, phantom, "\perp"] & \textbf{Occurrence Nets} \arrow[l, hook', shift right] \arrow[r, shift right, "\mathcal{E}"'] \arrow[r, phantom, "\perp"] & \textbf{Prime Event Structures} \arrow[l, shift right, "\mathcal{N}"'] \arrow[r, shift right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}"']
\end{tikzcd}$$

- **Categories** instead of sets (objects related via morphisms)
- **Functorial** semantics (morphisms are preserved)
- **Backward** constructions (and they are still functors)
- Forward and backward constructions form **adjunctions** (universal property witnesses optimality, preservation of (co)limits)...
- ... more precisely, **coreflections** (particularly nice adjunctions, where a natural iso establishes an equivalence between the denotational domain and a full subcategory of computational models)
- Later generalized to (semi-weighted) P/T nets (Sassone's PhD Thesis)

A Nice Event Structures Semantics

Winskel's Construction

$$\begin{tikzcd} \textbf{Safe Nets} \arrow[r, shift right, "\mathcal{U}"'] \arrow[r, phantom, "\perp"] & \textbf{Occurrence Nets} \arrow[l, hook', shift right] \arrow[r, shift right, "\mathcal{E}"'] \arrow[r, phantom, "\perp"] & \textbf{Prime Event Structures} \arrow[l, shift right, "\mathcal{N}"'] \arrow[r, shift right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}"'] \end{tikzcd}$$

- **Categories** instead of sets (objects related via morphisms)
- **Functorial** semantics (morphisms are preserved)
- **Backward** constructions (and they are still functors)
- Forward and backward constructions form **adjunctions** (universal property witnesses optimality, preservation of (co)limits)...
- ... more precisely, **coreflections** (particularly nice adjunctions, where a natural iso establishes an equivalence between the denotational domain and a full subcategory of computational models)
- Later generalized to (semi-weighted) P/T nets (Sassone's PhD Thesis)

A Nice Event Structures Semantics

Winskel's Construction

$$
\begin{tikzcd}
\textbf{Safe Nets} \arrow[r, shift right=1.5ex, "\mathcal{U}"'] \arrow[r, phantom, "\perp"] & \textbf{Occurrence Nets} \arrow[l, hook', shift right=1.5ex] \arrow[r, shift right=1.5ex, "\mathcal{E}"'] \arrow[r, phantom, "\perp"] & \textbf{Prime Event Structures} \arrow[l, shift right=1.5ex, "\mathcal{N}"'] \arrow[r, shift right=1.5ex, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, shift right=1.5ex, "\mathcal{P}"']
\end{tikzcd}
$$

- **Categories** instead of sets (objects related via morphisms)
- **Functorial** semantics (morphisms are preserved)
- **Backward** constructions (and they are still functors)
- Forward and backward constructions form **adjunctions** (universal property witnesses optimality, preservation of (co)limits)...
- ... more precisely, **coreflections** (particularly nice adjunctions, where a natural iso establishes an equivalence between the denotational domain and a full subcategory of computational models)
- Later generalized to (semi-weighted) P/T nets (Sassone's PhD Thesis)

A Nice Event Structures Semantics

Winskel's Construction

$$
\begin{tikzcd}
\textbf{Safe Nets} \arrow[r, shift right, "\mathcal{U}"'] \arrow[r, phantom, "\perp"] & \textbf{Occurrence Nets} \arrow[l, hook', shift right] \arrow[r, shift right, "\mathcal{E}"'] \arrow[r, phantom, "\perp"] & \textbf{Prime Event Structures} \arrow[l, shift right, "\mathcal{N}"'] \arrow[r, shift right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}"']
\end{tikzcd}
$$

- **Categories** instead of sets (objects related via morphisms)
- **Functorial** semantics (morphisms are preserved)
- **Backward** constructions (and they are still functors)
- Forward and backward constructions form **adjunctions** (universal property witnesses optimality, preservation of (co)limits)...
- ... more precisely, **coreflections** (particularly nice adjunctions, where a natural iso establishes an equivalence between the denotational domain and a full subcategory of computational models)
- Later generalized to (semi-weighted) P/T nets (Sassone's PhD Thesis)

A Nice Event Structures Semantics

Winskel's Construction

$$\begin{tikzcd}[column sep=large]
\textbf{Safe Nets} \arrow[r, bend right=0, shift right, "\mathcal{U}"'] \arrow[r, phantom, "\perp"] & \textbf{Occurrence Nets} \arrow[l, hook', shift right] \arrow[r, shift right, "\mathcal{E}"'] \arrow[r, phantom, "\perp"] & \textbf{Prime Event Structures} \arrow[l, shift right, "\mathcal{N}"'] \arrow[r, shift right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}"']
\end{tikzcd}$$

- **Categories** instead of sets (objects related via morphisms)
- **Functorial** semantics (morphisms are preserved)
- **Backward** constructions (and they are still functors)
- Forward and backward constructions form **adjunctions** (universal property witnesses optimality, preservation of (co)limits)...
- ... more precisely, **coreflections** (particularly nice adjunctions, where a natural iso establishes an equivalence between the denotational domain and a full subcategory of computational models)
- Later generalized to (semi-weighted) P/T nets (Sassone's PhD Thesis)

More Sophisticated Event Structures

Goguen's Categorical Manifesto [MSCS 1(1):49–67, 1991]

- To each species of mathematical structure, there corresponds a category whose objects have that structure, and whose morpshisms preserve it.
- To any construction from one species of structure to another, there corresponds a functor between the corresponding categories
- To any **canonical** construction from one species of structure to another corresponds an adjunction between the corresponding categories

What about read / inhibitor arcs, high-level / dynamic nets, graph grammars?

Additional Efforts

- Asymmetric Event Structures (AES) and Inhibitor Event Structures (IES) are needed (Baldan's PhD Thesis)
- Theory not available when encodings were first defined
- Constructions become much more complex, less elegant (categorically speaking)... and still not always applicable

More Sophisticated Event Structures

Goguen's Categorical Manifesto [MSCS 1(1):49–67, 1991]

- To each species of mathematical structure, there corresponds a category whose objects have that structure, and whose morpshisms preserve it.
- To any construction from one species of structure to another, there corresponds a functor between the corresponding categories
- To any **canonical** construction from one species of structure to another corresponds an adjunction between the corresponding categories

What about read / inhibitor arcs, high-level / dynamic nets, graph grammars?

Additional Efforts

- Asymmetric Event Structures (AES) and Inhibitor Event Structures (IES) are needed (Baldan's PhD Thesis)
- Theory not available when encodings were first defined
- Constructions become much more complex, less elegant (categorically speaking)... and still not always applicable

More Sophisticated Event Structures

Goguen's Categorical Manifesto [MSCS 1(1):49–67, 1991]

- To each species of mathematical structure, there corresponds a category whose objects have that structure, and whose morpshisms preserve it.
- To any construction from one species of structure to another, there corresponds a functor between the corresponding categories
- To any **canonical** construction from one species of structure to another corresponds an adjunction between the corresponding categories

What about read / inhibitor arcs, high-level / dynamic nets, graph grammars?

Additional Efforts

- Asymmetric Event Structures (AES) and Inhibitor Event Structures (IES) are needed (Baldan's PhD Thesis)
- Theory not available when encodings were first defined
- Constructions become much more complex, less elegant (categorically speaking)... and still not always applicable

Our Contribution

Computational Model Requirements

- Based on Graph Grammars (convenient for modeling many features)
- As simple as possible, but general enough to be widely applicable for encoding nominal calculi (finite representation of each process, avoid dealing with those GG features not really needed in encodings)
- PES semantics via chain of coreflections (like Winskel's approach)...
- ... possibly stable under SPO and DPO approaches

Our Proposal: Persistent Graph Grammars

$$\begin{tikzcd}
\textbf{Persistent GG} \arrow[r, shift right=1.5ex, "\mathcal{U}_p"'] \arrow[r, phantom, "\perp"] & \textbf{Persistent Occurrence GG} \arrow[l, hook', shift right=1.5ex] \arrow[r, shift right=1.5ex, "\mathcal{E}_p"'] \arrow[r, phantom, "\perp"] & \textbf{PES} \arrow[l, shift right=1.5ex, "\mathcal{N}_p"'] \arrow[r, shift right=1.5ex, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, shift right=1.5ex, "\mathcal{P}"']
\end{tikzcd}$$

Sample encodings: π-calculus and join calculus

Our Contribution

Computational Model Requirements

- Based on Graph Grammars (convenient for modeling many features)
- As simple as possible, but general enough to be widely applicable for encoding nominal calculi (finite representation of each process, avoid dealing with those GG features not really needed in encodings)
- PES semantics via chain of coreflections (like Winskel's approach)...
- ... possibly stable under SPO and DPO approaches

Our Proposal: Persistent Graph Grammars

$$\begin{tikzcd}
\textbf{Persistent GG} \arrow[r, shift right, "\mathcal{U}_p"'] \arrow[r, phantom, "\perp"] & \textbf{Persistent Occurrence GG} \arrow[l, hook', shift right] \arrow[r, shift right, "\mathcal{E}_p"'] \arrow[r, phantom, "\perp"] & \textbf{PES} \arrow[l, shift right, "\mathcal{N}_p"'] \arrow[r, shift right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}"']
\end{tikzcd}$$

Sample encodings: π-calculus and join calculus

Outline

DPO Graph Grammars at a Glance

Configurations are (Directed) Typed Graphs

$\tau_G : G \rightarrow T$ (ex: G bipartite if $T =$ $\begin{tikzcd} n_1 \arrow[r, bend right, "a_1"'] & n_2 \arrow[l, bend right, "a_2"'] \end{tikzcd}$, G any if $T =$ $\begin{tikzcd} n \arrow[loop right, "a"] \end{tikzcd}$)

Rewrite Rules are Spans

$p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ (T-typed:

$$\begin{tikzcd} L \arrow[dr, "\tau_L"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "\tau_K"] & R \arrow[dl, "\tau_R"] \\ & T & \end{tikzcd}$$

)

Double Pushout Rewriting (from G to H via p)

$$p : \begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(2)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(3)"] & R \arrow[d, "f"] \\ G & D \arrow[l] \arrow[r] & H \end{tikzcd}$$

(1) find a (valid) match m
(2) $D =$ remove $m(L - l(K))$ from G
(3) $H =$ paste a fresh copy of $R - r(K)$
T is preserved everywhere

DPO Graph Grammars at a Glance

Configurations are (Directed) Typed Graphs

$\tau_G : G \to T$ (ex: G bipartite if $T = n_1 \underset{a_1}{\overset{a_2}{\rightleftarrows}} n_2$, G any if $T = n \circlearrowleft a$)

Rewrite Rules are Spans

$p : (L \xleftarrow{l} K \xrightarrow{r} R)$ (T-typed:

$$\begin{tikzcd} L \arrow[dr, "\tau_L"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "\tau_K"] & R \arrow[dl, "\tau_R"] \\ & T & \end{tikzcd}$$

)

Double Pushout Rewriting (from G to H via p)

$$p: \quad \begin{tikzcd} L \arrow[d, "m"'] \arrow[d, phantom, shift right=6ex, "(1)"] \arrow[dr, phantom, "(2)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(3)"] & R \arrow[d, "f"] \\ G & D \arrow[l] \arrow[r] & H \end{tikzcd}$$

(1) find a (valid) match m

(2) $D =$ remove $m(L - l(K))$ from G

(3) $H =$ paste a fresh copy of $R - r(K)$

T is preserved everywhere

DPO Graph Grammars at a Glance

Configurations are (Directed) Typed Graphs

$\tau_G : G \to T$ (ex: G bipartite if $T =$ $\begin{tikzcd} n_1 \arrow[r, bend right, "a_1"'] & n_2 \arrow[l, bend right, "a_2"'] \end{tikzcd}$, G any if $T =$ $\begin{tikzcd} n \arrow[loop right, "a"] \end{tikzcd}$)

Rewrite Rules are Spans

$p : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ (T-typed:

$$\begin{tikzcd} L \arrow[dr, "\tau_L"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "\tau_K"] & R \arrow[dl, "\tau_R"] \\ & T & \end{tikzcd}$$

)

Double Pushout Rewriting (from G to H via p)

p :

$$\begin{tikzcd} L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] & R \arrow[d, "f"] \\ G & D \arrow[l] \arrow[r] & H \end{tikzcd}$$

(1) (2) (3)

(1) find a (valid) match m

(2) $D =$ remove $m(L - l(K))$ from G

(3) $H =$ paste a fresh copy of $R - r(K)$

T is preserved everywhere

Example: Place/Transition Petri Nets as Graph Grammars

From N to GG_N

- Consider just discrete graphs (no arcs)
- Type graph T = set of places of N
- Typed graphs = markings
- Nodes of a typed graph = (named) tokens
- Productions = transitions of N
  - L_t is the preset of t
  - K_t is empty
  - R_t is the postset of t

A Tiny Example

preset of $t = a \oplus b$

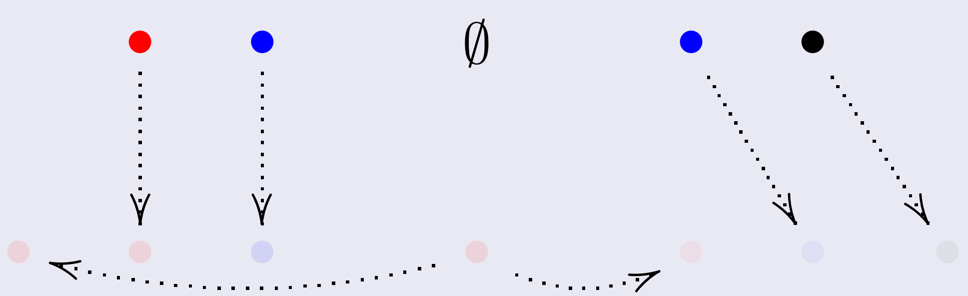
postset of $t = b \oplus c$

Example: Place/Transition Petri Nets as Graph Grammars

From N to GG_N

- Consider just discrete graphs (no arcs)
- Type graph T = set of places of N
- Typed graphs = markings
- Nodes of a typed graph = (named) tokens
- Productions = transitions of N
    - L_t is the preset of t
    - K_t is empty
    - R_t is the postset of t

A Tiny Example



preset of $t = a \oplus b$

postset of $t = b \oplus c$

initial marking $= 2a \oplus b$

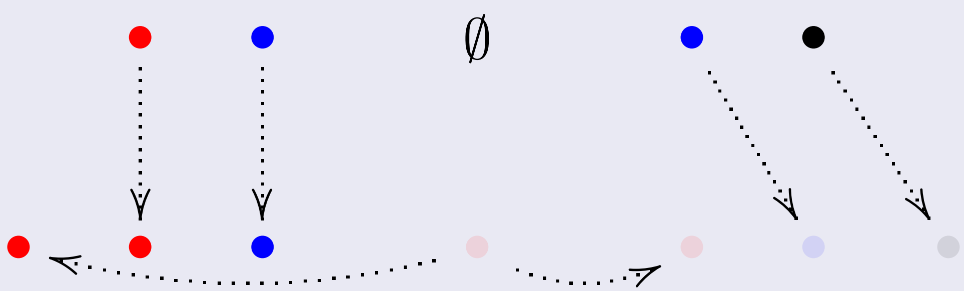
firing $= 2a \oplus b\,[t\rangle\, a \oplus b \oplus c$

Example: Place/Transition Petri Nets as Graph Grammars

From N to GG_N

- Consider just discrete graphs (no arcs)
- Type graph T = set of places of N
- Typed graphs = markings
- Nodes of a typed graph = (named) tokens
- Productions = transitions of N
  - L_t is the preset of t
  - K_t is empty
  - R_t is the postset of t

A Tiny Example



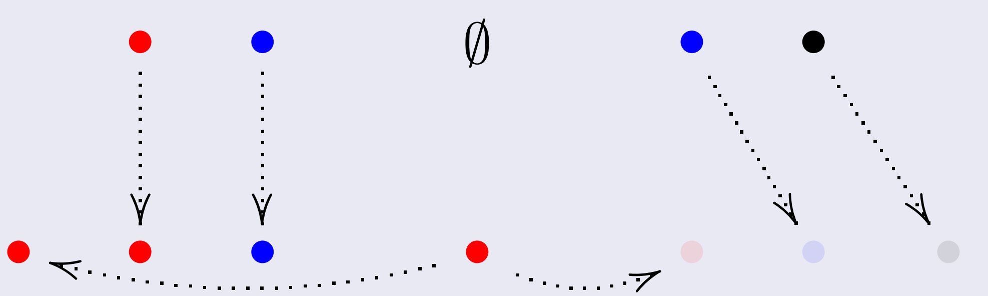
preset of $t = a \oplus b$
postset of $t = b \oplus c$
initial marking $= 2a \oplus b$
firing $= 2a \oplus b\,[t\rangle\, a \oplus b \oplus c$

Example: Place/Transition Petri Nets as Graph Grammars

From N to GG_N

- Consider just discrete graphs (no arcs)
- Type graph T = set of places of N
- Typed graphs = markings
- Nodes of a typed graph = (named) tokens
- Productions = transitions of N
    - L_t is the preset of t
    - K_t is empty
    - R_t is the postset of t

A Tiny Example



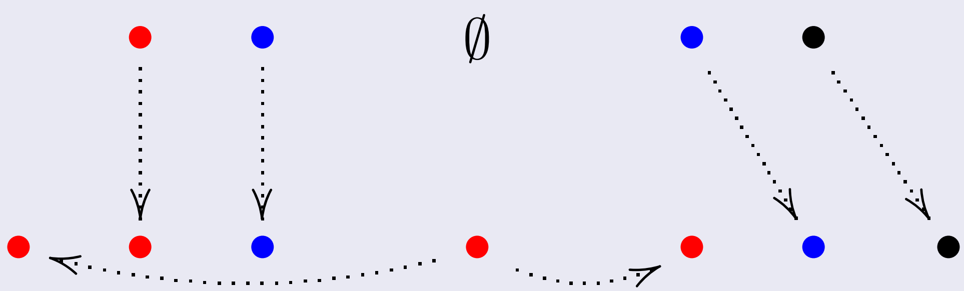
preset of $t = a \oplus b$
postset of $t = b \oplus c$
initial marking $= 2a \oplus b$
firing $= 2a \oplus b\,[t\rangle\, a \oplus b \oplus c$

Example: Place/Transition Petri Nets as Graph Grammars

From N to GG_N

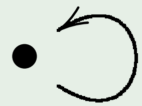
- Consider just discrete graphs (no arcs)
- Type graph T = set of places of N
- Typed graphs = markings
- Nodes of a typed graph = (named) tokens
- Productions = transitions of N
  - L_t is the preset of t
  - K_t is empty
  - R_t is the postset of t

A Tiny Example



preset of $t = a \oplus b$
postset of $t = b \oplus c$
initial marking $= 2a \oplus b$
firing $= 2a \oplus b\,[t\rangle\, a \oplus b \oplus c$

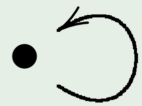
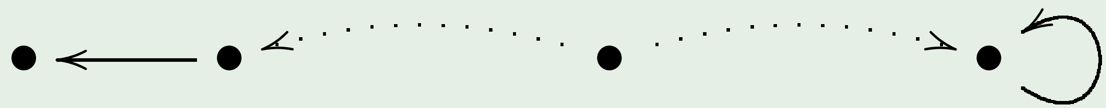
DPO vs SPO (Informally)

- type graph: 
- production:
- initial configuration:
- matching:

Final configuration?

- DPO not applicable: dangling arc in
- SPO is applicable: dangling arcs are removed

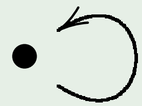
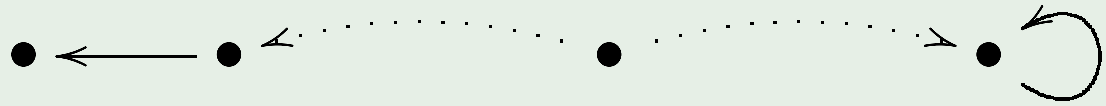
DPO vs SPO (Informally)

- type graph: 
- production: 
- initial configuration:
- matching:

Final configuration?

- DPO not applicable: dangling arc in

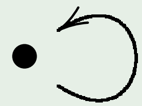
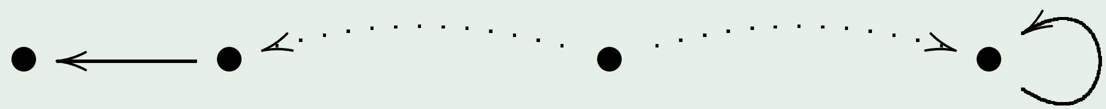
DPO vs SPO (Informally)

- type graph: 
- production: 
- initial configuration: 
- matching:

Final configuration?

- DPO not applicable: dangling arc in
- SPO is applicable: dangling arcs are removed

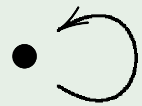
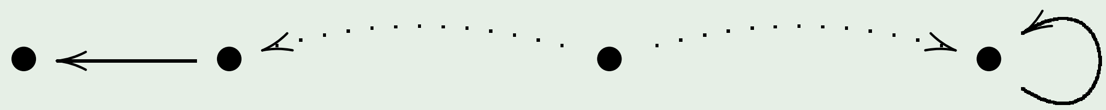
DPO vs SPO (Informally)

- type graph: 
- production: 
- initial configuration: 
- matching: 

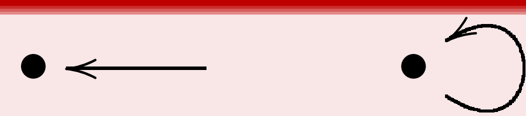
Final configuration?

- DPO not applicable: dangling arc in
- SPO is applicable: dangling arcs are removed

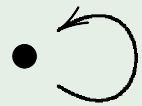
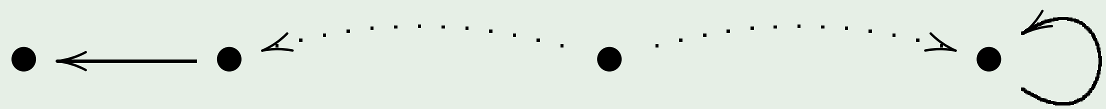
DPO vs SPO (Informally)

- type graph: 
- production: 
- initial configuration: 
- matching: 

Final configuration?

- DPO not applicable: dangling arc in 
- SPO is applicable: dangling arcs are removed

DPO vs SPO (Informally)

- type graph: 
- production: 
- initial configuration: 
- matching: 

Final configuration?

- DPO not applicable: dangling arc in 
- SPO is applicable: dangling arcs are removed 

DPO vs SPO (Semantically)

DPO: Baldan's PhD Thesis

$$\begin{tikzcd} \textbf{S-W DPO GG} \arrow[r, shift right, "\mathcal{U}_g"'] \arrow[r, phantom, "\perp"] & \textbf{S-W Occurrence DPO GG} \arrow[l, hook, shift right] \arrow[r, "\mathcal{E}_g"'] & \textbf{IES} \arrow[r, shift right, "\mathcal{L}_i"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}_i"'] \end{tikzcd}$$

SPO: A Recent Result (Baldan, Corradini, Montanari, Ribeiro)

$$\begin{tikzcd} \textbf{S-W SPO GG} \arrow[r, shift right, "\mathcal{U}_s"'] \arrow[r, phantom, "\perp"] & \textbf{S-W Occurrence SPO GG} \arrow[l, hook, shift right] \arrow[r, shift right, "\mathcal{E}_s"'] \arrow[r, phantom, "\perp"] & \textbf{AES} \arrow[l, shift right, "\mathcal{N}_s"'] \arrow[r, shift right, "\mathcal{L}_a"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}_a"'] \end{tikzcd}$$

DPO vs SPO (Semantically)

DPO: Baldan's PhD Thesis

$$\begin{tikzcd} \textbf{S-W DPO GG} \arrow[r, shift right, "\mathcal{U}_g"'] \arrow[r, phantom, "\perp"] & \textbf{S-W Occurrence DPO GG} \arrow[l, hook', shift right] \arrow[r, "\mathcal{E}_g"'] & \textbf{IES} \arrow[r, shift right, "\mathcal{L}_i"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}_i"'] \end{tikzcd}$$

SPO: A Recent Result (Baldan, Corradini, Montanari, Ribeiro)

$$\begin{tikzcd} \textbf{S-W SPO GG} \arrow[r, shift right, "\mathcal{U}_s"'] \arrow[r, phantom, "\perp"] & \textbf{S-W Occurrence SPO GG} \arrow[l, hook', shift right] \arrow[r, shift right, "\mathcal{E}_s"'] \arrow[r, phantom, "\perp"] & \textbf{AES} \arrow[l, shift right, "\mathcal{N}_s"'] \arrow[r, shift right, "\mathcal{L}_a"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}_a"'] \end{tikzcd}$$

Graph Grammars Instead of Petri Nets

Graph Grammars Benefits

- Computational power
- Multiple accesses in reading is built-in (via K)
- More sophisticated typing mechanism
- Structural congruence as graph isomorphism
- Name sharing as node sharing
- Creation of fresh items is built-in (right-hand square of DPO)
- Can encode dynamic productions (Bruni and Melgratti [ICGT'06])

Applicability vs Categorical Adequacy of the Semantics

- Which constructions?
- Which approach (DPO /SPO)?
- Too much general for the purpose of encoding nominal calculi?

Graph Grammars Instead of Petri Nets

Graph Grammars Benefits

- Computational power
- Multiple accesses in reading is built-in (via K)
- More sophisticated typing mechanism
- Structural congruence as graph isomorphism
- Name sharing as node sharing
- Creation of fresh items is built-in (right-hand square of DPO)
- Can encode dynamic productions (Bruni and Melgratti [ICGT'06])

Applicability vs Categorical Adequacy of the Semantics

- Which constructions?
- Which approach (DPO /SPO)?
- Too much general for the purpose of encoding nominal calculi?

Outline

PGG

Some Restrictions

- *Consuming Productions*: to avoid infinite auto-concurrency
- *Node Persistence*: to reconcile SPO and DPO
- **Typeable Persistent Arcs**: to construct PES instead of AES
- *Semi-Weightedness*: to avoid ambiguity and redundancy in the unfolding and get a chain of coreflections

Main Result

$$
\begin{tikzcd}
\textbf{Persistent GG} \arrow[r, bend right, "\mathcal{U}_p"'] \arrow[r, phantom, "\perp"] & \textbf{Persistent Occurrence GG} \arrow[l, hook', bend right] \arrow[r, bend right, "\mathcal{E}_p"'] \arrow[r, phantom, "\perp"] & \textbf{PES} \arrow[l, bend right, "\mathcal{N}_p"'] \arrow[r, bend right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \textbf{Domains} \arrow[l, bend right, "\mathcal{P}"']
\end{tikzcd}
$$

PGG

Some Restrictions

- *Consuming Productions*: to avoid infinite auto-concurrency
- *Node Persistence*: to reconcile SPO and DPO
- **Typeable Persistent Arcs**: to construct PES instead of AES
- *Semi-Weightedness*: to avoid ambiguity and redundancy in the unfolding and get a chain of coreflections

Main Result

$$\textbf{Persistent GG} \;\underset{\mathcal{U}_p}{\overset{\perp}{\rightleftarrows}}\; \begin{array}{c}\textbf{Persistent}\\ \textbf{Occurrence GG}\end{array} \;\underset{\mathcal{E}_p}{\overset{\mathcal{N}_p}{\rightleftarrows}}\; \textbf{PES} \;\underset{\mathcal{L}}{\overset{\mathcal{P}}{\rightleftarrows}}\; \textbf{Domains}$$

Discussion

Some Restrictions

- **Consuming Productions**: corresponds to the common constraint on Petri nets about *non-emptiness of presets*. This constraint has e.g. some consequences in the design of productions for modeling replication and join definitions.
- **Node Persistence**: solves the matter of *dangling arcs*. No relevant consequences for the event structure semantics, because in our modeling all computational entities are represented as arcs, not as nodes.
- **Typeable Persistent Arcs**: solves the matter of *asymmetric conflicts* (it is not possible to fetch resources that others can access in reading). It has no particular consequences in the encodings that we have considered.
- **Semi-Weightedness**: bans the presence of *multiple, indistinguishable resources*. It has some (non-dramatic) consequences on the encodable agents and it is likely the most restrictive requirement on PGG.

Note that "red items" apply to general GG, not just PGG

Outline

Why π-calculus and join calculus

Why π-calculus

- Popularity
- Refine Montanari and Pistore's [MFPS'95] encoding
- Not just the finite fragment, still finite representation of each process
- Asynchronous case shown in the paper (see Uwe Nestmann's tutorial), but straightforward extension to the synchronous case

Why join calculus

- Carefully designed with an eye to implementation issues
- Interesting as reflexive extension of Petri nets
- Join definitions introduce challenging synchronization patterns for the reuse of other techniques (like direct encoding)
- First event structure semantics for the join calculus (that we are aware of)

Why π-calculus and join calculus

Why π-calculus

- Popularity
- Refine Montanari and Pistore's [MFPS'95] encoding
- Not just the finite fragment, still finite representation of each process
- Asynchronous case shown in the paper (see Uwe Nestmann's tutorial), but straightforward extension to the synchronous case

Why join calculus

- Carefully designed with an eye to implementation issues
- Interesting as reflexive extension of Petri nets
- Join definitions introduce challenging synchronization patterns for the reuse of other techniques (like direct encoding)
- First event structure semantics for the join calculus (that we are aware of)

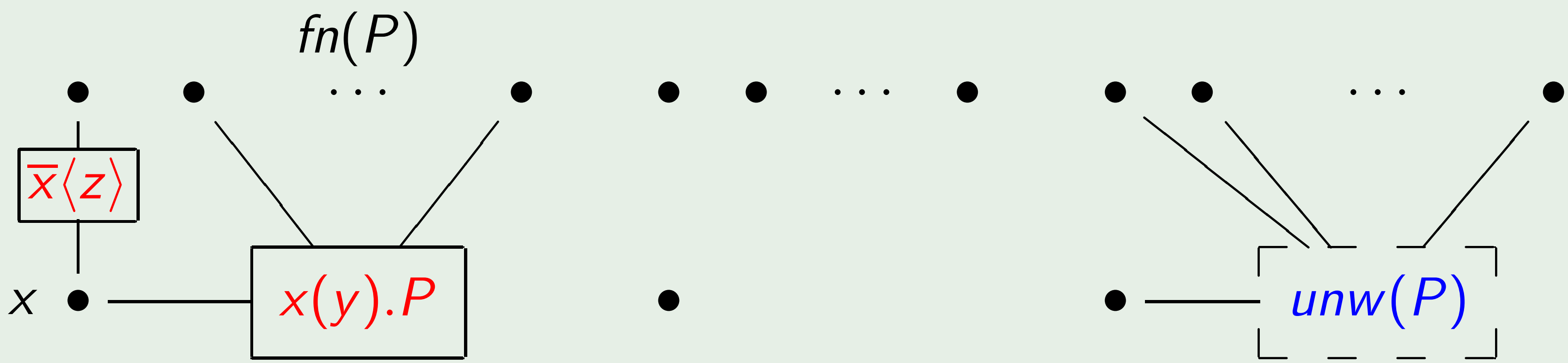
The General Idea

Common Guidelines

- Canonical representation of processes
- Names are represented as nodes in configurations
- Sequential computational entities are represented as (hyper)arcs in configurations
- Distinct agents can execute concurrently
- Node sharing is the key to establish connectivity and communication
- Finitely many types, statically determined by subterms of canonical processes
- Types determine the applicable productions and hence the behaviour of computational entities
- Within productions, π-replication and join-definitions are modeled as persistent resources, so they can produce multiple concurrent events without introducing unnecessary serialization

Sketched Rules

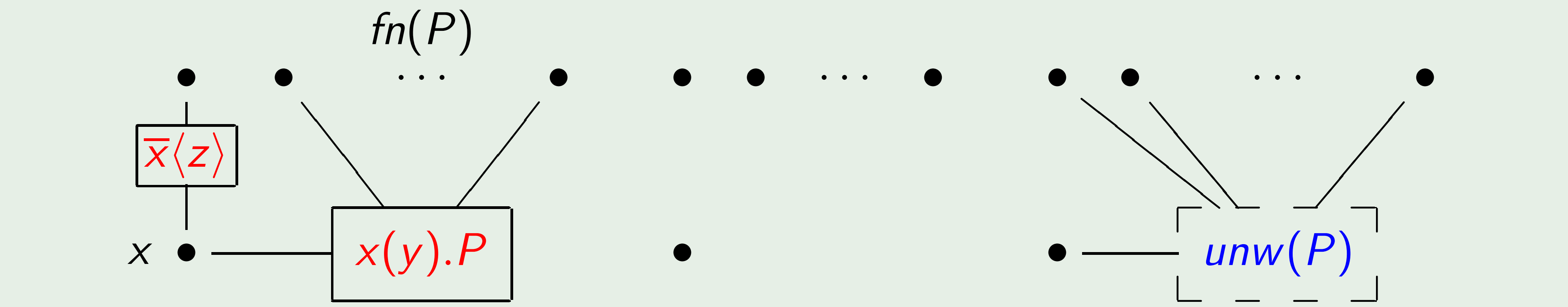
Input in π



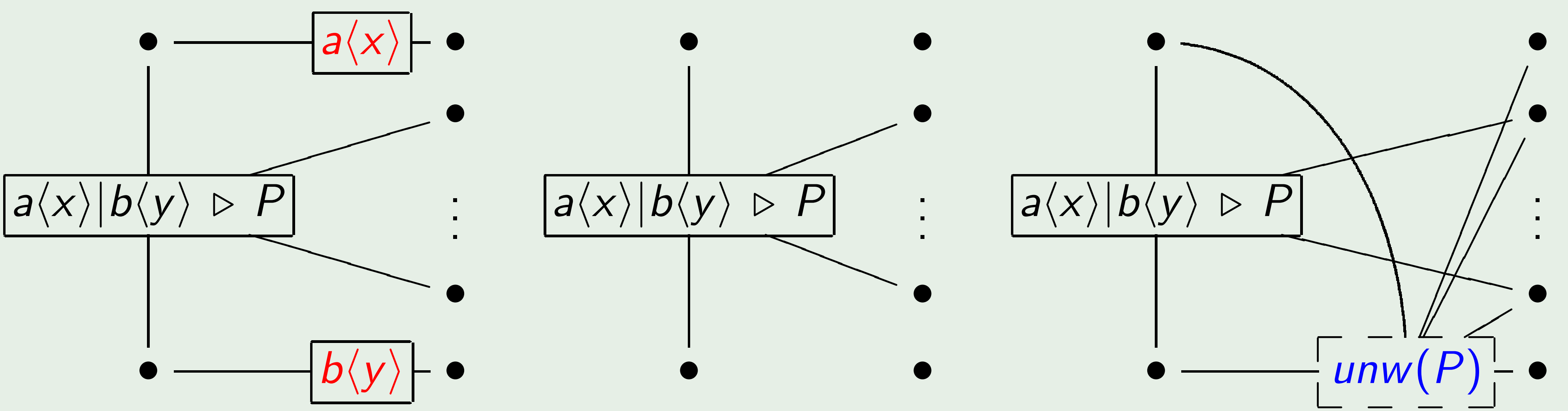
Reaction in Core Join

Sketched Rules

Input in π



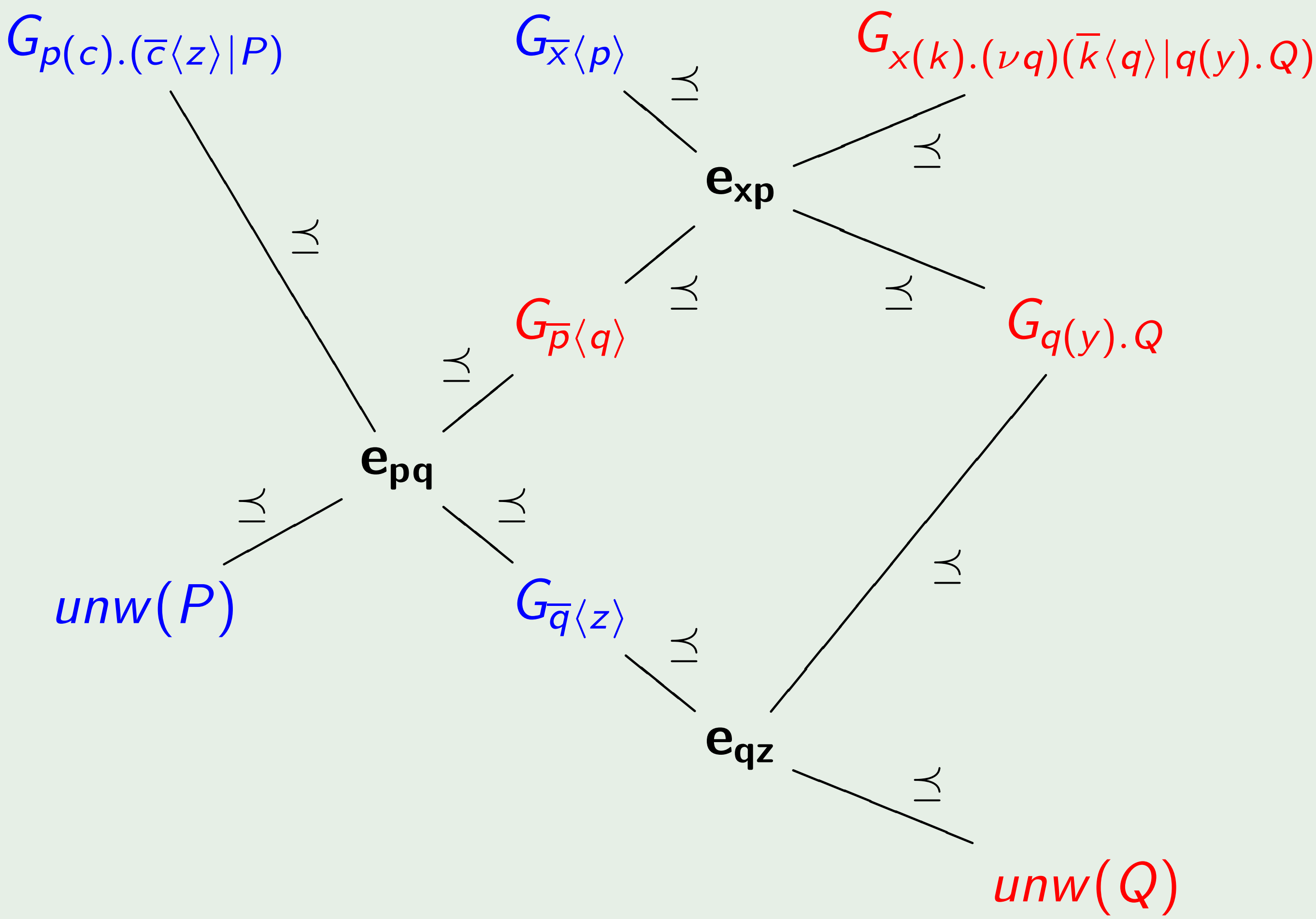
Reaction in Core Join



Short Example

From Synchronous to Asynchronous

$$S = \quad \ldots \quad \overline{x}\langle z\rangle.P \quad \ldots \quad x(y).Q \quad \ldots$$
$$A = \quad \ldots \quad (\nu p)(\overline{x}\langle p\rangle | p(c).(\overline{c}\langle z\rangle | P)) \quad \ldots \quad x(k).(\nu q)(\overline{k}\langle q\rangle | q(y).Q) \quad \ldots$$



Outline

Conclusions

Recap

- Persistent Graph Grammars (PGGs) offer a suitable setting for equipping nominal calculi with (indirect) truly concurrent semantics collecting the best of the two worlds:
  - PES via a chain of coreflections
  - disciplined encoding of nominal calculi
- Constructions are stable under DPO and SPO
- First PES semantics for join
  - applicable to coloured and reconfigurable nets by exploiting the result in Buscemi and Sassone [FOSSACS'01]
- We exploit finite type graph (and configuration) for each process

Related Issues

Shifting the Ground

- Reduction semantics vs open semantics: problems of non-consuming productions
- Parallel extrusion: it can be allowed or disallowed as already shown by Montanari and Pistore [MFPS'01]
- Structure vs link dependencies: not clear how to distinguish between the two
- Isolated (persistent) nodes: always garbage collectable
- Hierarchical encoding of sequential processes vs flat encoding: the latter may require node fusion (see Baldan, Gadducci and Montanari's paper)
- Non semi-weighted processes: serialize the release of resources with the same type

Future Work

Open issues

- Relax semi-weightedness? (adjunctions instead of coreflections?)
- Similarities between Varacca and Yoshida's linearity constraint and semi-weightedness criterion? (can their type systems be transferred to graph grammar productions?)
- Application of the technique to other mobile calculi, like ambients?
- Comparison with previous non-interleaving semantics of the π-calculus?

The End

T H A N K S !