A Graph Syntax for Processes and Services

Alberto Lluch Lafuente
(joint-work with Roberto Bruni and Fabio Gadducci)

Department of Computer Science, University of Pisa
Software Engineering for Service-Oriented Overlay Computers

6th Int'l Workshop on Web Services and Formal Methods
Bologna, September 4-5, 2009

Goal statement

The spirit of our research is

"to conciliate algebraic and graph-based specifications"

Goal statement

The spirit of our research is

"to conciliate algebraic and graph-based specifications"

In this work we propose a graph syntax to

"Equip algebraic specifications with a graphical representation that is

- *Intuitive*
- *Easy to define*
- *Easy to prove correct*

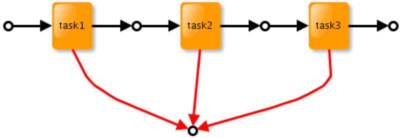
Running Example: Sagas

We shall consider a simple language for transactions with

- sequential composition;
- parallel (split-join) composition;
- compensations;
- saga scoping.

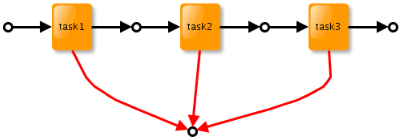
This example is inspired by the *Nested Sagas* of [BMM05].

Modelling Sagas with Graphs (sketch)

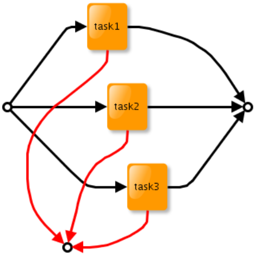


sequential composition

Modelling Sagas with Graphs (sketch)

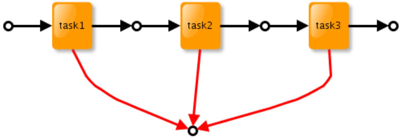


sequential composition

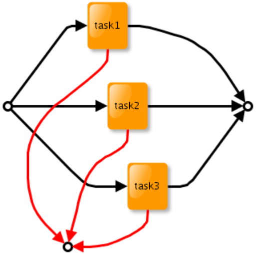


parallel composition

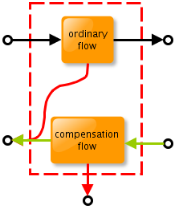
Modelling Sagas with Graphs (sketch)



sequential composition

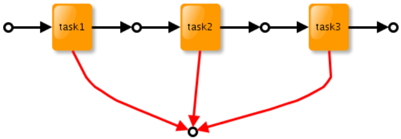


parallel composition

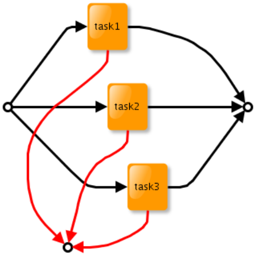


compensation

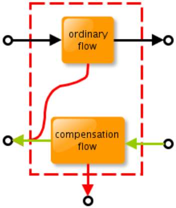
Modelling Sagas with Graphs (sketch)



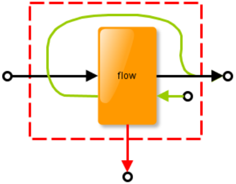
sequential composition



parallel composition

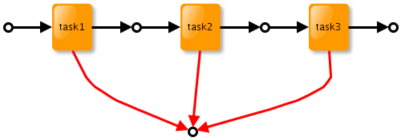


compensation



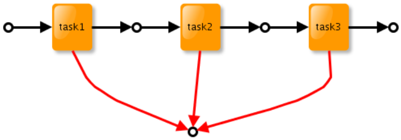
saga

Modelling Sagas with a Process Calculus (sketch)

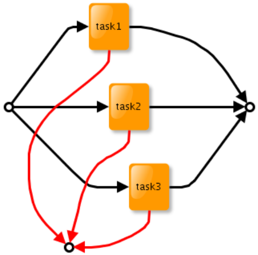



```
task1 ; task2 ; task3
```

Modelling Sagas with a Process Calculus (sketch)

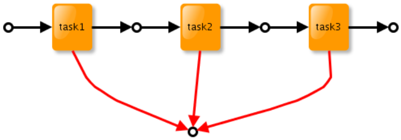


task1 ; task2 ; task3

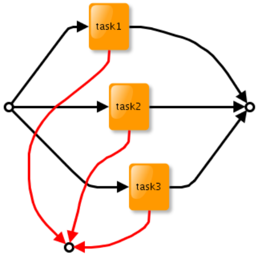


task1 | task2 | task3

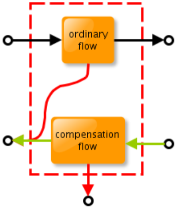
Modelling Sagas with a Process Calculus (sketch)



`task1 ; task2 ; task3`

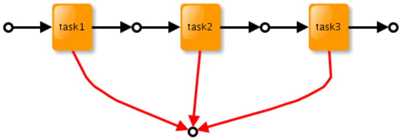


`task1 | task2 | task3`

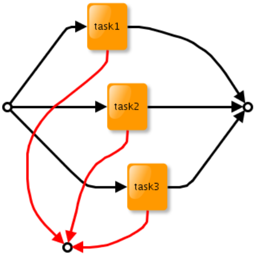



```
ordinary flow
%compensation flow
```

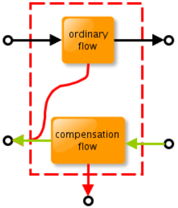
Modelling Sagas with a Process Calculus (sketch)



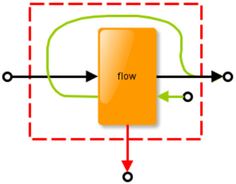
`task1 ; task2 ; task3`



`task1 | task2 | task3`




```
ordinary flow
%compensation flow
```



`[flow]`

Calculi vs Graphs

Algebraic

- Terms
  a | b

elements

Graph-based

- Graphs (diagrams)
  flat, hierarchical, etc.

Calculi vs Graphs

Algebraic

- Terms
  $\mathtt{a} \mid \mathtt{b}$
- Operations
  $\cdot|\cdot : \mathtt{W} \times \mathtt{W} \rightarrow \mathtt{W}$

elements

vocabulary

Graph-based

- Graphs (diagrams)
  flat, hierarchical, etc.
- Graph compositions
  Union, tensor, etc.

Calculi vs Graphs

Algebraic

- Terms
  $\mathtt{a} \mid \mathtt{b}$
- Operations
  $\cdot|\cdot : \mathtt{W} \times \mathtt{W} \rightarrow \mathtt{W}$
- Axioms
  $\mathtt{x} \mid \mathtt{y} \equiv \mathtt{y} \mid \mathtt{x}$

elements

vocabulary

equivalence

Graph-based

- Graphs (diagrams)
  flat, hierarchical, etc.
- Graph compositions
  Union, tensor, etc.
- Homomorphisms
  isomorphism, etc.

Calculi vs Graphs

Algebraic

- Terms
  $\mathtt{a} \mid \mathtt{b}$
- Operations
  $\cdot|\cdot : \mathtt{W} \times \mathtt{W} \rightarrow \mathtt{W}$
- Axioms
  $\mathtt{x} \mid \mathtt{y} \equiv \mathtt{y} \mid \mathtt{x}$
- Rewrite rules
  $\mathtt{a} \longrightarrow \mathtt{b}$

elements

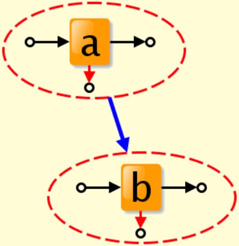
vocabulary

equivalence

dynamics

Graph-based

- Graphs (diagrams)
  flat, hierarchical, etc.
- Graph compositions
  Union, tensor, etc.
- Homomorphisms
  isomorphism, etc.
- Transformation rules

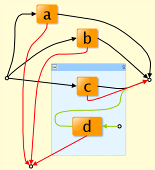


Main technical goal: mapping coherent wrt. equivalence

flow1

```
a
| b
| [ c % d]
```

graph1

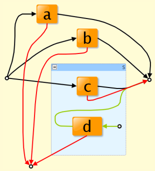


Main technical goal: mapping coherent wrt. equivalence

flow1

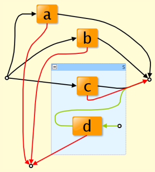
```
a
| b
| [ c % d]
```

graph1



Main technical goal: mapping coherent wrt. equivalence

graph1



flow1

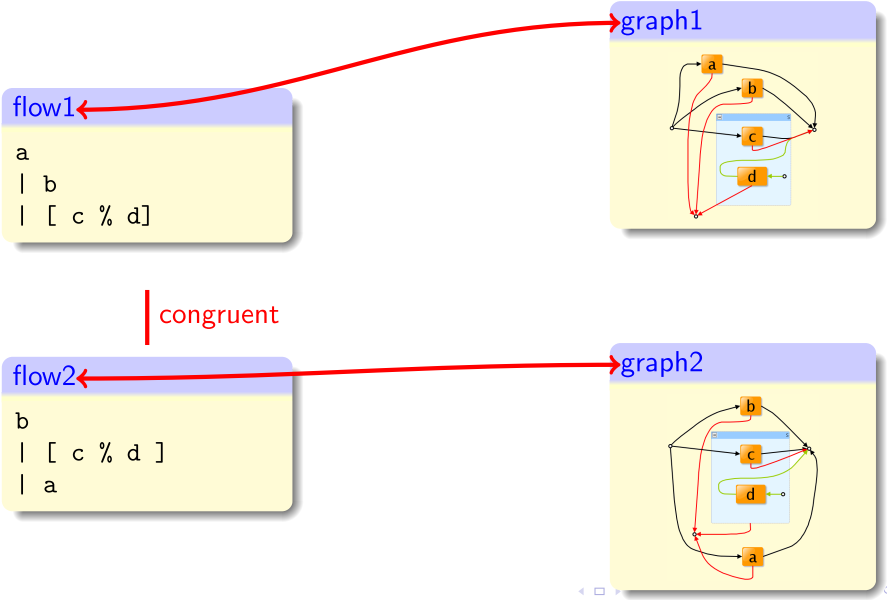
```
a
| b
| [ c % d]
```

congruent

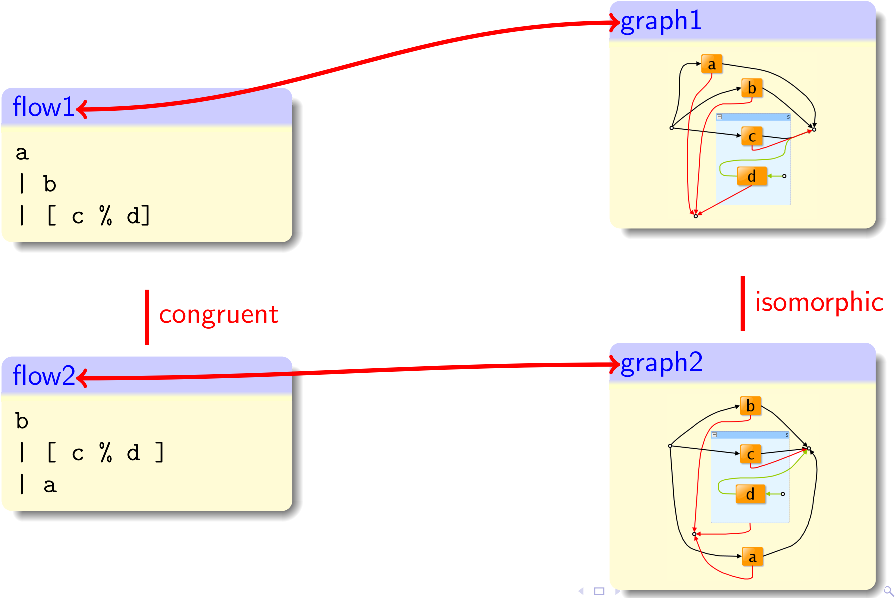
flow2

```
b
| [ c % d ]
| a
```

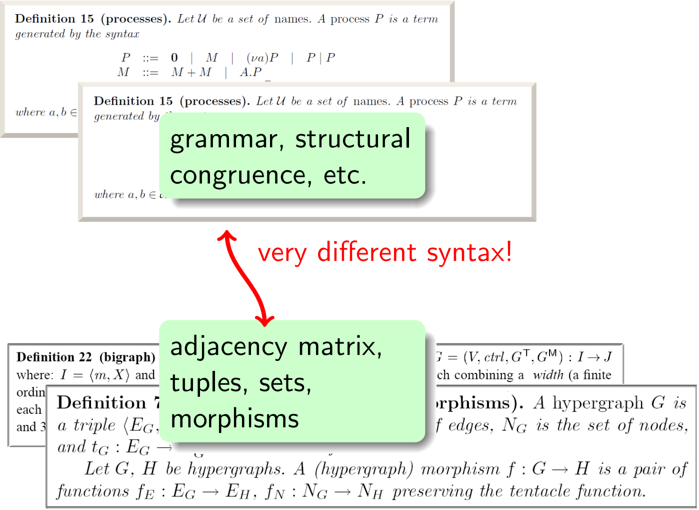
Main technical goal: mapping coherent wrt. equivalence



Main technical goal: mapping coherent wrt. equivalence



Main technical problem: representation distance



Main technical problem: representation distance

Definition 15 (processes). *Let* $\mathcal{U}$ *be a set of* names. *A* process P *is a term generated by the syntax*

$$P ::= \mathbf{0} \mid M \mid (\nu a)P \mid P \,|\, P$$
$$M ::= M + M \mid A.P$$

where $a, b \in$

Definition 1

generated by

where $a, b \in \mathcal{U}$

similar syntax

solution: graph algebras

$$[\![(\nu a)P]\!]_\Gamma = \begin{cases} [\![P]\!]_\Gamma & \text{if } a \notin \mathtt{fn}(P) \\ (id_p \otimes \nu_c \otimes id_\Gamma) \circ [\![P\{^c/_a\}]\!]_{\{c\}\uplus\Gamma} & \text{otherwise} \end{cases}$$

$$[\![P \mid Q]\!]_\Gamma = [\![P]\!]_\Gamma \otimes [\![Q]\!]_\Gamma \qquad [\![a(b).P]\!]_\Gamma = (in_{a,c} \otimes id_\Gamma) \circ [\![P\{^c/_b\}]\!]_{\{c\}\uplus\Gamma}$$

$$[\![0]\!]_\Gamma = 0_p \otimes 0_\Gamma \qquad [\![\bar{a}b.P]\!]_\Gamma = (out_{a,b} \otimes id_\Gamma) \circ [\![P]\!]_\Gamma$$

$$[\![\mathbf{0}]\!]_X = 1 \curlywedge X \qquad [\![P|Q]\!]_X = [\![P]\!]_X \curlywedge [\![Q]\!]_X \qquad [\![(x)P]\!]_X = \blacktriangle_x \circ [\![P]\!]_{X\uplus\{x\}}$$

$$[\![zx.P]\!]_X = \mathsf{get}^{x,z} \circ [\![P]\!]_X \qquad [\![\bar{z}x.P]\!]_X = \mathsf{send}^{x,z} \circ [\![P]\!]_X \qquad \text{where } x, z \in X$$

$$[(\nu a)P]_n^p = \mathtt{hide}_n([P\{^{n+1}/_a\}]_{n+1}^p) \qquad [\bar{i}j.P]_n^s = \mathtt{out}_{i,j,n}([P]_n^p)$$

$$[P \mid Q]_n^p = \mathtt{par}_n([P]_n^p, [Q]_n^p) \qquad [i(y).P]_n^s = \mathtt{in}_{i,n}([P\{^{n+1}/_y\}]_{n+1}^p)$$

$$[0]_n^s = \mathtt{nil}_n \qquad [M + N]_n^s = \mathtt{choice}_n([M]_n^s, [N]_n^s)$$

Definition 22 (bigraph)

where: $I = \langle m, X\rangle$ and J

ordin

each

and 3

Definition 7 (

a triple $\langle E_G, N_G, t_G\rangle$ *such that* E_G *is the set of edges,* N_G *is the set of nodes, and* $t_G : E_G \to N_G^*$ *is the tentacle function.*

Let G*,* H *be hypergraphs. A (hypergraph) morphism* $f : G \to H$ *is a pair of functions* $f_E : E_G \to E_H$*,* $f_N : N_G \to N_H$ *preserving the tentacle function.*

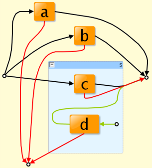
similar syntax

Main application: encodings are facilitated

flow1

```
a
| b
| [ c % d]
```

graph1



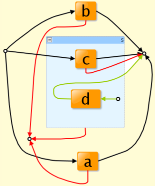
congruent

isomorphic

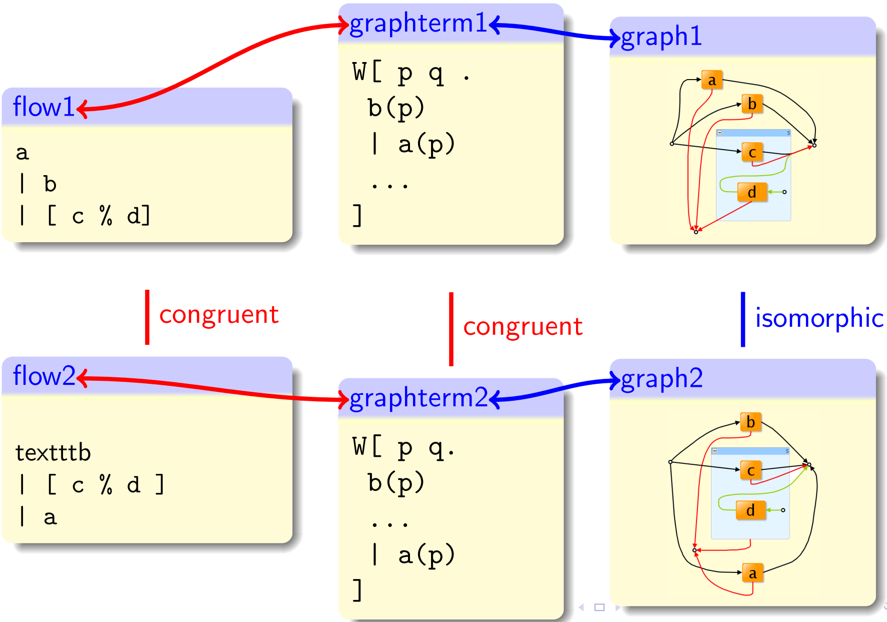
flow2

```
texttttb
| [ c % d ]
| a
```

graph2



Main application: encodings are facilitated



The syntax of the graph algebra

$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0}$$

the empty graph

The syntax of the graph algebra

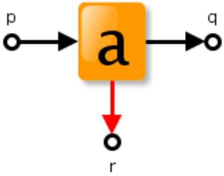
$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0} \mid x$$

a node called x

The syntax of the graph algebra

$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0} \mid x \mid t(\overline{x})$$

an (hyper)edge labelled with t attached to $\overline{x}$

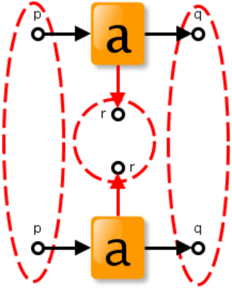


for instance, `a(p,q,r)`

The syntax of the graph algebra

$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0} \mid x \mid t(\overline{x}) \mid \mathbb{G}|\mathbb{H}$$

parallel composition: disjoint union up to common nodes

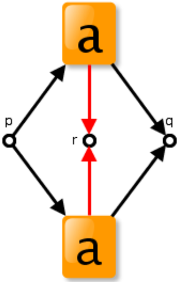


for instance, a(p,q,r) | a(p,q,r)

The syntax of the graph algebra

$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0} \;\mid x \;\mid t(\overline{x}) \;\mid \mathbb{G} | \mathbb{H}$$

parallel composition: disjoint union up to common nodes

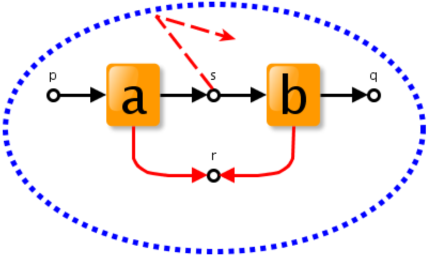


for instance, `a(p,q,r) | a(p,q,r)`

The syntax of the graph algebra

$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0} \mid x \mid t(\overline{x}) \mid \mathbb{G}|\mathbb{H} \mid (\nu x)\mathbb{G}$$

declaration of a new node x

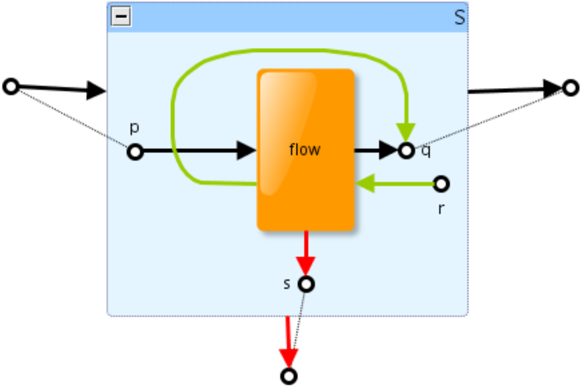


for instance, `(νs) (a(p,s,r) | b(s,q,r))`

The syntax of the graph algebra

$$\mathbb{D} \quad ::= \quad T_{\overline{x}}[\mathbb{G}]$$
$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0} \mid x \mid t(\overline{x}) \mid \mathbb{G}|\mathbb{H} \mid (\nu x)\mathbb{G}$$

graph G with interface of type T exposing $\overline{x}$

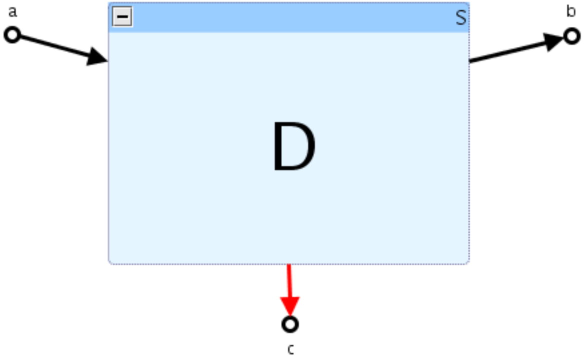


for instance, $\mathtt{S}_{\mathtt{p},\mathtt{q},\mathtt{s}}[(\nu\mathtt{r})\mathtt{flow}(\mathtt{p},\mathtt{q},\mathtt{r},\mathtt{q},\mathtt{s})]$

The syntax of the graph algebra

$$\mathbb{D} \quad ::= \quad T_{\overline{x}}[\mathbb{G}]$$
$$\mathbb{G}, \mathbb{H} \quad ::= \quad \mathbf{0} \;\mid x \;\mid t(\overline{x}) \;\mid \mathbb{G}|\mathbb{H} \;\mid (\nu x)\mathbb{G} \;\mid \mathbb{D}\langle\overline{y}\rangle$$

a nested graph attached to $\overline{y}$



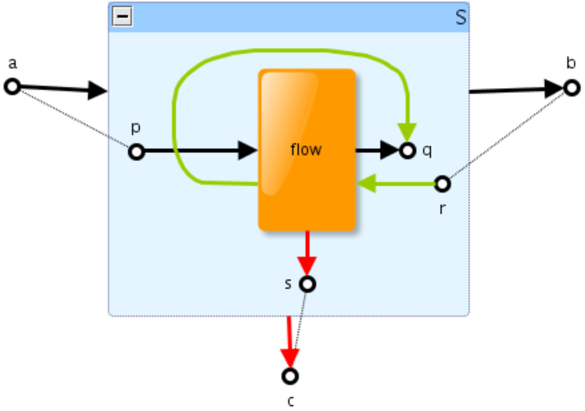
for instance, D⟨a,b,c⟩

The syntax of the graph algebra

$$\mathbb{D} ::= T_{\overline{x}}[\mathbb{G}]$$

$$\mathbb{G}, \mathbb{H} ::= \mathbf{0} \mid x \mid t(\overline{x}) \mid \mathbb{G}|\mathbb{H} \mid (\nu x)\mathbb{G} \mid \mathbb{D}\langle\overline{y}\rangle$$

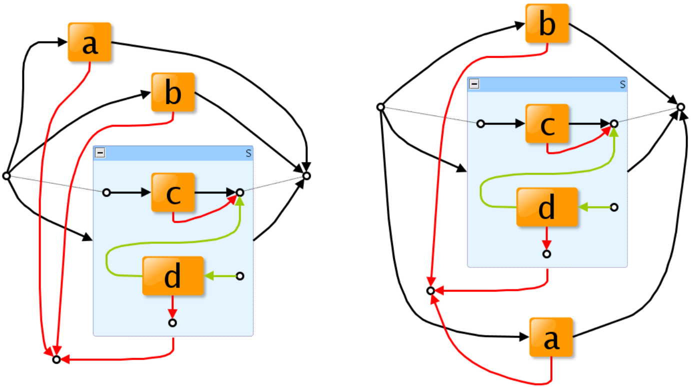
a nested graph attached to $\overline{y}$



for instance, $\mathtt{D}\langle\mathtt{a,b,c}\rangle$, with $\mathtt{D}{=}\mathtt{S}_{\mathtt{p,q,s}}[(\nu\mathtt{r})\mathtt{flow}(\mathtt{p,q,r,q,s})]$

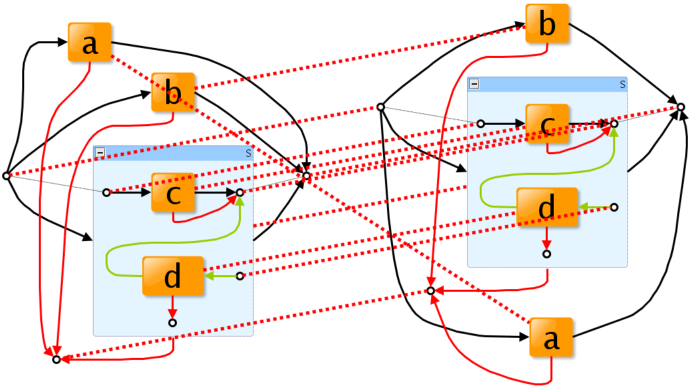
Identifying equivalent graphs

The actual model of hierarchical graphs has some notion of hierarchical isomorphism.



Identifying equivalent graphs

The actual model of hierarchical graphs has some notion of hierarchical isomorphism.



Identifying equivalent graphs

Isomorphism is elegantly captured by structural axioms.

$$\mathbb{G} \mid\mid \mathbb{H} \equiv \mathbb{H} \mid\mid \mathbb{G} \tag{PARALLEL1}$$

$$\mathbb{G} \mid\mid (\mathbb{H} \mid\mid \mathbb{I}) \equiv (\mathbb{G} \mid\mid \mathbb{H}) \mid\mid \mathbb{I} \tag{PARALLEL2}$$

is equivalent to

Identifying equivalent graphs

Isomorphism is elegantly captured by structural axioms.

$$
\begin{array}{rcll}
\mathbb{G} \mid\mid \mathbb{H} & \equiv & \mathbb{H} \mid\mid \mathbb{G} & \text{(PARALLEL1)} \\
\mathbb{G} \mid\mid (\mathbb{H} \mid\mid \mathbb{I}) & \equiv & (\mathbb{G} \mid\mid \mathbb{H}) \mid\mid \mathbb{I} & \text{(PARALLEL2)}
\end{array}
$$

$$
\begin{array}{rclll}
\mathbb{G} \mid\mid \mathbf{0} & \equiv & \mathbb{G} & & \text{(NODES1)} \\
(\nu x)(\nu y)\mathbb{G} & \equiv & (\nu y)(\nu x)\mathbb{G} & & \text{(NODES2)} \\
(\nu x)\mathbf{0} & \equiv & \mathbf{0} & & \text{(NODES5)} \\
(\nu x)\mathbb{G} & \equiv & (\nu y)\mathbb{G}\{^{y}/_{x}\} & \text{if } y \notin fn(\mathbb{G}) & \text{(NODES3)} \\
L_{\overline{x}}[\mathbb{G}] & \equiv & L_{\overline{y}}[\mathbb{G}\{^{\overline{y}}/_{\overline{x}}\}] & \text{if } |\overline{y}| \cap fn(\mathbb{G}) = \emptyset & \text{(NODES4)} \\
\mathbb{G} \mid\mid (\nu x)\mathbb{H} & \equiv & (\nu x)(\mathbb{G} \mid\mid \mathbb{H}) & \text{if } x \notin fn(\mathbb{G}) & \text{(NODES5)} \\
L_{\overline{x}}[(\nu y)\mathbb{G}](\overline{z}) & \equiv & (\nu y)L_{\overline{x}}[\mathbb{G}](\overline{z}) & \text{if } y \notin |\overline{x}| \cup |\overline{z}| & \text{(NODES6)} \\
x \mid\mid \mathbb{G} & \equiv & \mathbb{G} & \text{if } x \in fn(\mathbb{G}) & \text{(NODES7)}
\end{array}
$$

These axioms are rather *standard* and thus *intuitive* to those familiar with algebraic specifications.

Sagas encoding: sagas as calculus

Let us assume the following syntax for our sagas language

$$\begin{array}{lllllllllll} \mathtt{S} & ::= & \mathtt{a} & \mid & \mathtt{S};\mathtt{S} & \mid & \mathtt{S}|\mathtt{S} & \mid & [\mathtt{P}] & & \text{(sagas)} \\ \mathtt{P} & ::= & \mathtt{S}\%\mathtt{S} & \mid & \mathtt{P};\mathtt{P} & \mid & \mathtt{P}|\mathtt{P} & & & & \text{(processes)} \end{array}$$

with the usual following axioms holding

- associativity for sequential composition;
- associativity and commutativity for parallel composition.

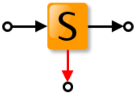
Sagas encoding: key ideas I

1. Algebraic reading of the calculus
   - Syntactical categories as *Sorts*
   - Productions as *Operators*

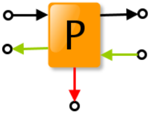
   for instance

   `S ::= S ; S` ====> $\_ \, ; \, \_ : \mathtt{S} \times \mathtt{S} \longrightarrow \mathtt{S}$

2. Each sort becomes a design label



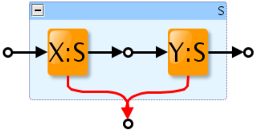
Sagas sort S



Process sort P

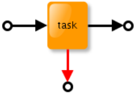
Sagas encoding: key ideas II

3. Each production becomes a derived operator



$$\mathtt{X} \; ; \; \mathtt{Y} \stackrel{\text{def}}{=} \mathtt{S}_{\mathtt{p,q,r}}[(\nu \mathtt{s})(\mathtt{X}\langle \mathtt{p}, \mathtt{s}, \mathtt{r}\rangle \mid \mathtt{Y}\langle \mathtt{s}, \mathtt{q}, \mathtt{r}\rangle)]$$

4. Some symbols should be material, i.e. represented by graph items like edges

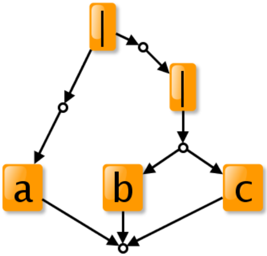


for instance, an activity

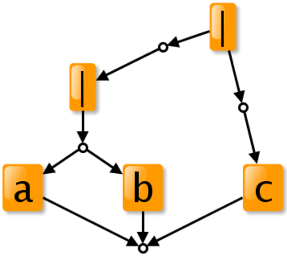
Sagas encoding: key ideas III

5. Some symbols should be immaterial. For instance, a material parallel operator yields non isomorphic graphs

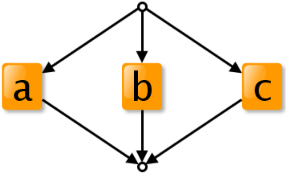
`( a | b ) | c`



`a | ( b | c )`



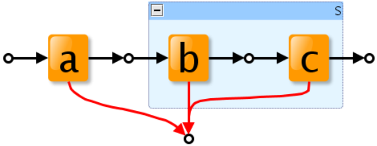
To capture associativity with iso we need something like



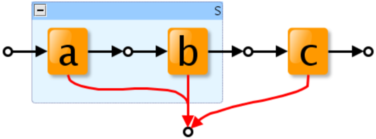
Sagas encoding: key ideas IV

6. Flattening dissolves composition frames.
   For instance, without flattening associativity is not captured by isomorphism

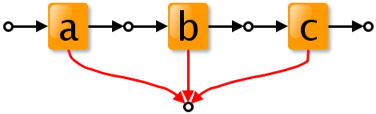
(a ; b) ; c



a ; (b ; c)

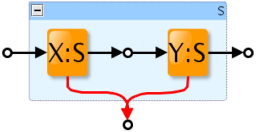


With flattening of sagas we get



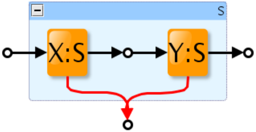
in both cases.

Sagas encoding: main productions

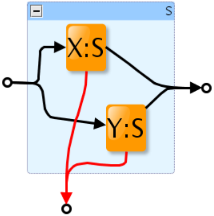


$$\mathtt{X} \;;\; \mathtt{Y} \stackrel{\textbf{def}}{=} \mathtt{S_{p,q,r}}[(\nu \mathtt{s})(\mathtt{X}\langle \mathtt{p}, \mathtt{s}, \mathtt{r}\rangle \mid \mathtt{Y}\langle \mathtt{s}, \mathtt{q}, \mathtt{r}\rangle)]$$

Sagas encoding: main productions

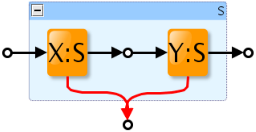


$$X \; ; \; Y \stackrel{\text{def}}{=} S_{p,q,r}[(\nu s)(X\langle p, s, r\rangle \mid Y\langle s, q, r\rangle)]$$

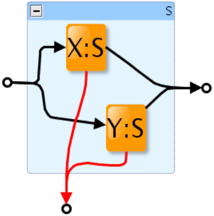


$$X \mid Y \stackrel{\text{def}}{=} S_{p,q,r}[X\langle p, q, r\rangle \mid Y\langle p, q, r\rangle]$$

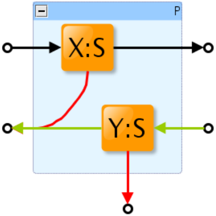
Sagas encoding: main productions



$$\mathtt{X\ ;\ Y} \stackrel{\text{def}}{=} \mathrm{S}_{p,q,r}[(\nu s)(\mathtt{X}\langle p,s,r\rangle \mid \mathtt{Y}\langle s,q,r\rangle)]$$

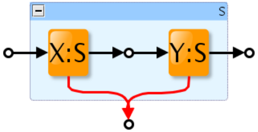


$$\mathtt{X} \mid \mathtt{Y} \stackrel{\text{def}}{=} \mathrm{S}_{p,q,r}[\mathtt{X}\langle p,q,r\rangle \mid \mathtt{Y}\langle p,q,r\rangle]$$

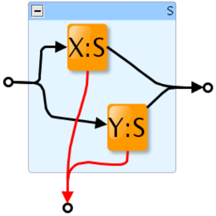


$$\mathtt{X\ \%\ Y} \stackrel{\text{def}}{=} \mathrm{P}_{p,q,r,s,t}[\mathtt{X}\langle p,q,s\rangle \mid \mathtt{Y}\langle r,s,t\rangle]$$

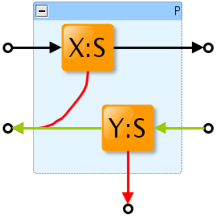
Sagas encoding: main productions



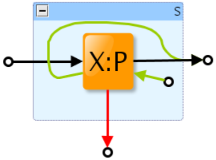
$$\mathtt{X}\ ;\ \mathtt{Y} \stackrel{\text{def}}{=} \mathtt{S}_{\mathtt{p,q,r}}[(\nu \mathtt{s})(\mathtt{X}\langle \mathtt{p}, \mathtt{s}, \mathtt{r}\rangle \mid \mathtt{Y}\langle \mathtt{s}, \mathtt{q}, \mathtt{r}\rangle)]$$



$$\mathtt{X} \mid \mathtt{Y} \stackrel{\text{def}}{=} \mathtt{S}_{\mathtt{p,q,r}}[\mathtt{X}\langle \mathtt{p}, \mathtt{q}, \mathtt{r}\rangle \mid \mathtt{Y}\langle \mathtt{p}, \mathtt{q}, \mathtt{r}\rangle]$$



$$\mathtt{X}\ \%\ \mathtt{Y} \stackrel{\text{def}}{=} \mathtt{P}_{\mathtt{p,q,r,s,t}}[\mathtt{X}\langle \mathtt{p}, \mathtt{q}, \mathtt{s}\rangle \mid \mathtt{Y}\langle \mathtt{r}, \mathtt{s}, \mathtt{t}\rangle]$$



$$[\mathtt{X}] \stackrel{\text{def}}{=} \mathtt{S}_{\mathtt{p,q,r}}[(\nu \mathtt{s})\mathtt{X}\langle \mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{q}, \mathtt{r}\rangle]$$

Sagas encoding: coherence proof

At the end we point at a result like

Theorem

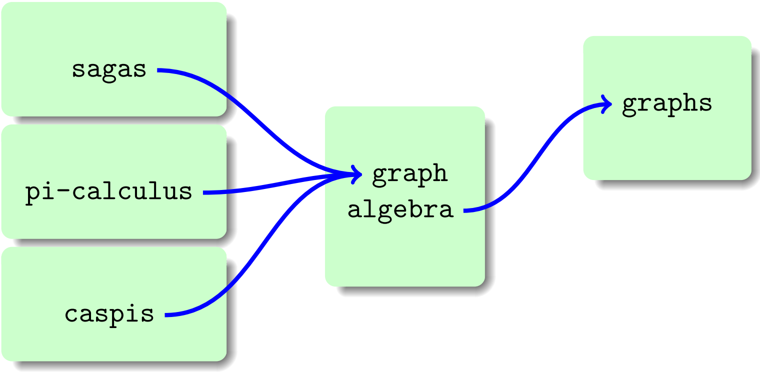
Two sagas S and R are congruent exactly when they are isomorphic.

- The proof of soundness is reduced to show that in each axiom of the structural congruence the lhs and rhs are isomorphic, which is facilitated by the similarity of the axioms.

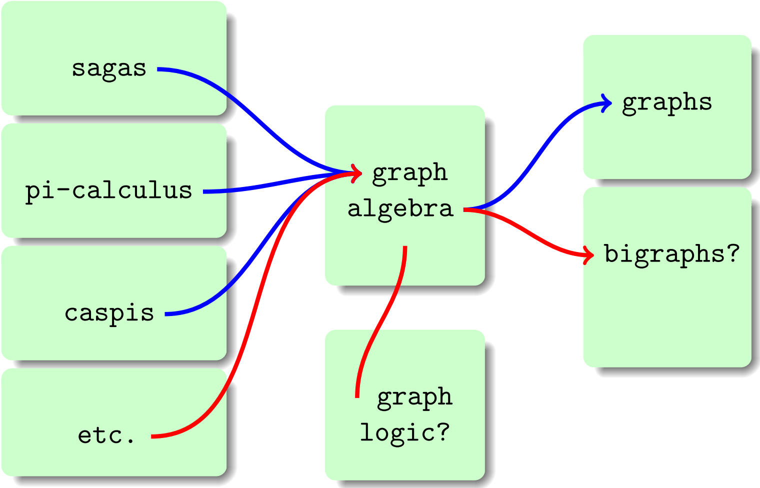
  For instance,
  $$\begin{aligned} \mathtt{X} \mid \mathtt{Y} &\stackrel{\text{def}}{=} \mathtt{S}_{\mathtt{p},\mathtt{q},\mathtt{r}}[\mathtt{X}\langle \mathtt{p},\mathtt{q},\mathtt{r}\rangle \mid \mathtt{Y}\langle \mathtt{p},\mathtt{q},\mathtt{r}\rangle] \\ &\stackrel{\text{par1}}{=} \mathtt{S}_{\mathtt{p},\mathtt{q},\mathtt{r}}[\mathtt{Y}\langle \mathtt{p},\mathtt{q},\mathtt{r}\rangle \mid \mathtt{X}\langle \mathtt{p},\mathtt{q},\mathtt{r}\rangle] \\ &\stackrel{\text{def}}{=} \mathtt{Y} \mid \mathtt{X} \end{aligned}$$
- The proof of completeness is done as usual by structural induction on the normal form of sagas terms. Still not easy, but at least we deal with similar notations.

Outline

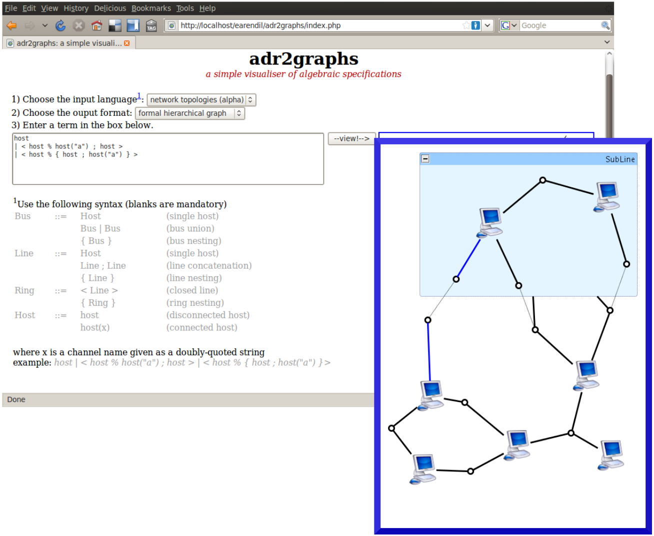
Possible scenario where the graph syntax could live



Possible scenario where the graph syntax could live



Implementation snapshot (a simple visualiser)

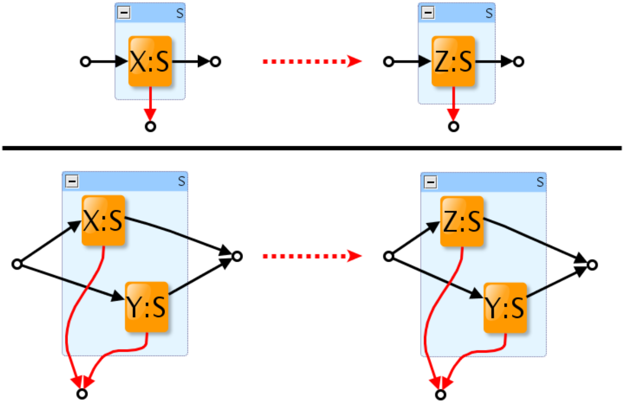


▶ Available at `www.albertolluch.com/adr2graphs`

One further goal

Our hope is to find a notion of graph rewriting such that graph transformations are directly inferred from

- the original semantic rules of the calculus
- the graphical encoding of terms.



Concluding remarks

The graphical syntax . . .

- Grounds on widely-accepted models;
- Simplifies the graphical representation of process calculi;
- Hides the complexity of hierarchical graphs;
- Enables proofs by structural induction;
- Has been evaluated on various calculi;
- Nesting and sharing features suitable for modelling soc features such as transactions or sessions.
- Natural implementation in RL/Maude (support for theorem proving, model checking, simulation, etc.)
- Offers a technique for complementing textual and visual notations in formal tools;

Credits and references I

[BGL09] Roberto Bruni, Fabio Gadducci, and Alberto Lluch Lafuente.
A graph syntax for processes and services.
In *WS-FM'09*, 2009.
To appear.

[BL09] Roberto Bruni and Alberto Lluch Lafuente.
Ten virtues of structured graphs.
In *Invited paper at the 8th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'09)*, Electronic Communications of the EASST, 2009.
To appear.

[BLM08] Roberto Bruni, Alberto Lluch Lafuente, and Ugo Montanari.
Hierarchical Design Rewriting with Maude.
In *Proceedings of the 7th International Workshop on Rewriting Logic and its Applications (WRLA'08)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2008.
To appear.

[BLME07] Roberto Bruni, Alberto Lluch Lafuente, Ugo Montanari, and Emilio Tuosto.
Service Oriented Architectural Design.
In *Proceedings of the 3rd International Symposium on Trustworthy Global Computing (TGC'07)*, volume 4912 of *Lecture Notes in Computer Science*, pages 186–203. Springer, 2007.

[BMM05] Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari.
Theoretical foundations for compensations in flow composition languages.
In Jens Palsberg and Martín Abadi, editors, *POPL*, pages 209–220. ACM, 2005.

[CG99] Andrea Corradini and Fabio Gadducci.
An algebraic presentation of term graphs, via gs-monoidal categories. applied categorical structures.
Applied Categorical Structures, 7:7–299, 1999.

[CMR94] Andrea Corradini, Ugo Montanari, and Francesca Rossi.
An abstract machine for concurrent modular systems: CHARM.
Theoretical Compututer Science, 122(1&2):165–200, 1994.

Credits and references II

[DHP02] Frank Drewes, Berthold Hoffmann, and Detlef Plump.
Hierarchical graph transformation.
Journal on Computer and System Sciences, 64(2):249–283, 2002.

[Gad03] Fabio Gadducci.
Term graph rewriting for the pi-calculus.
In Atsushi Ohori, editor, *Proceedings of the 1st Asian Symposium on Programming Languages and Systems*, volume 2895 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2003.

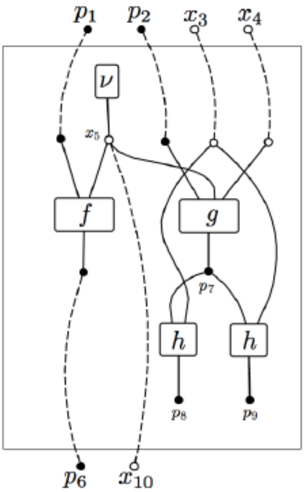
[JM03] O. H. Jensen and R. Milner.
Bigraphs and mobile processes.
Technical Report 570, Computer Laboratory, University of Cambridge, 2003.

Note: Some figures have been borrowed from the referred papers.

Related work

GS-Graphs [CG99]

- syntactical structure, algebraic presentation
- flat (hierarchy-as-tree)



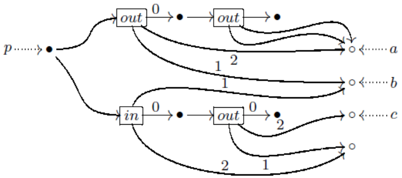
Related work

GS-Graphs [CG99]

- syntactical structure, algebraic presentation
- flat (hierarchy-as-tree)

Ranked Graphs [Gad03]

- node sharing, calculi encoding
- no composition interface, flat



Related work

GS-Graphs [CG99]

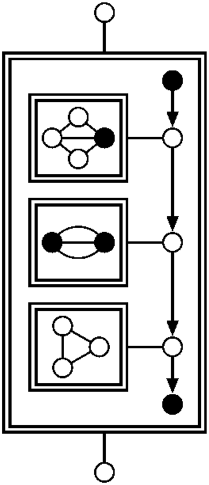
- syntactical structure, algebraic presentation
- flat (hierarchy-as-tree)

Ranked Graphs [Gad03]

- node sharing, calculi encoding
- no composition interface, flat

Hierarchical Graphs [DHP02]

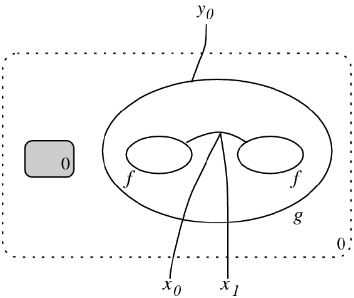
- basic model, composition interface
- no node sharing, no algebraic syntax



Related Work

Bigraphs [JM03]

- nesting + linking
- 2 overlapping structures, complex syntax, no composition interface, flat



Related Work

Bigraphs [JM03]

- nesting + linking
- 2 overlapping structures, complex syntax, no composition interface, flat

Graph Algebra, SHR [CMR94]

- basic algebra
- flat, no composition interface