Coalgebras for Named Sets

Ugo Montanari
Dipartimento di Informatica
Università di Pisa

Work in collaboration with
Vincenzo Ciancia
Universidad Complutense Madrid

Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

Nominal calculi

- At the end of the eighties, it was realised that novel features of programming languages, such as mobility of network components, required new programming paradigms and new models
- The π-calculus is **the** prototypical example: exploiting a *name generation* construct, reflected by a *scope extrusion* mechanism in the semantics, it achieves mobility
- Nowadays, name generation and name passing have proved fundamental for new trends in computing: e.g. *sessions* in service oriented computing, *objects* in object-oriented programming, *keys* and *nonces* in security protocols ...

The quest for a good model

The switch of programming paradigm required to change the underlying models, as set theory proved not to be at the right level of abstraction (no notion of binding in algebras, or fresh name generation in coalgebras).
Set theory needs to be enriched with names.

We want to be able to reason about the semantics:

- Proof systems: prove properties using inference rules
- Model checking: prove properties of finite-state systems in a fully automated fashion (e.g. deadlock-freeness)
- Equivalence checking: prove that two different programs behave in the same way
- Minimisation: find the system that has less states up-to behavioral equivalence

Three models

- **Nominal sets / permutation algebras**: names are implicit as the action of injective substitutions. Syntax [Gabbay, Pitts LICS 1999,...] semantics [Montanari, Pistore MFCS 2000,...] of name passing.

- **Presheaf models** [Fiore, Moggi, Sangiorgi LICS 1996,...], [Cattani, Sewell, Winskel CSL 1997,...]: names are indexes of *stages* of a staged set construction: $\mathbf{Set}^{\mathbf{I}}$ where $\mathbf{I}$ is the category of **finite sets** and **injections**. Actually, the pullback-preserving full subcategory of $\mathbf{Set}^{\mathbf{I}}$: the **Schanuel Topos**

- **Named sets** [Montanari, Pistore, Yankelevich ESOP 1997], named sets with symmetries [Pistore (Thesis) 1999,...]: names are finite sets, with an associated symmetry, attached to elements of a set

[Gadducci, Miculan, Montanari 2006], [Fiore, Staton IC 2006]: the three models are categorically equivalent.

Implementing nominal computation

- Presheaf models and nominal sets: good abstract models, **a simple specification formalism** for nominal calculi. **Infinite** in very simple cases.
- How do we implement the semantics? How do we find the minimal system, and define model-checking algorithms?
- [Ferrari, Montanari, Tuosto - TCS 2005]: we can minimise the **finite control π-calculus agents** and mechanically check bisimulation.
- Presence of various "tricks" that proved fundamental to correctly deal with name passing and name generation.
  The minimisation tool finds the **optimal symmetry reduction** up-to bisimulation.

New results

- provide a **set of operations** that can be used for arbitrary calculi: each operation corresponds to a **functor,** is proved **correct** by the means of category theory
- provide a close correspondence with functors, algebras and coalgebras in nominal sets, so translation from specification to implementation is easy and well-known.
- provide a final coalgebra theorem: **abstract semantics and minimisation**
- **generalise** the "tricks", or techniques, used for the π-calculus, that can now be applied to a wide range of cases, and recover **history-dependent bisimulation** for place-transition Petri Nets in the standard coalgebraic framework.
- Possibly high impact on applications. Example: **remove redundant variables** and find **optimal symmetry reduction** of a programming language, by just giving the operational rules in the formalism of nominal sets, that allows names and binding.

Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

Global vs. local names

- Presheaves and permutation algebras have **global names**, that have an "absolute" meaning

$$P(x,y) = \bar{x}y.0 \parallel \bar{y}x.0 \quad \neq \quad P(z,t) = \bar{z}t.0 \parallel \bar{t}z.0$$

- Up-to injective relabelling, the behavior of $P(x,y)$ and of $P(z,t)$ is the same so keeping them distinct is somewhat artificial.

- Named sets handle names as **local** resources, or placeholders.
  $P(x,y)$ and $P(z,t)$ (and any other injective substitution) are identified.

- Many consequences. In a sentence: categorical constructions over named sets (name abstraction, product, power set, non-deterministic choice) **do** reflect **common practice** when dealing with name generation and name passing, e.g. modelling garbage collection, or wiring up systems when composing them.

Named Sets

- Basic idea: model states of a system as a set whose elements have a set of "local names", or placeholders, attached
- Morphisms trace the *history* of names using an associated injection from names of the destination to names of the source

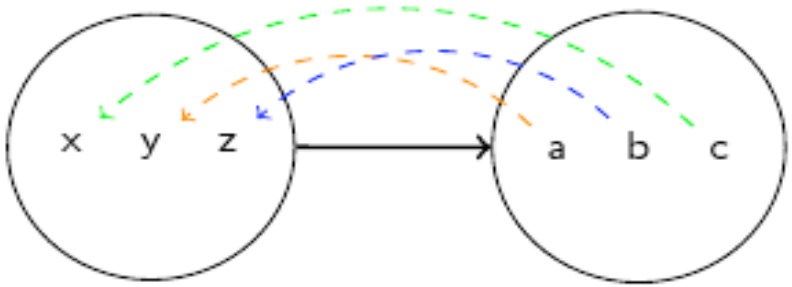
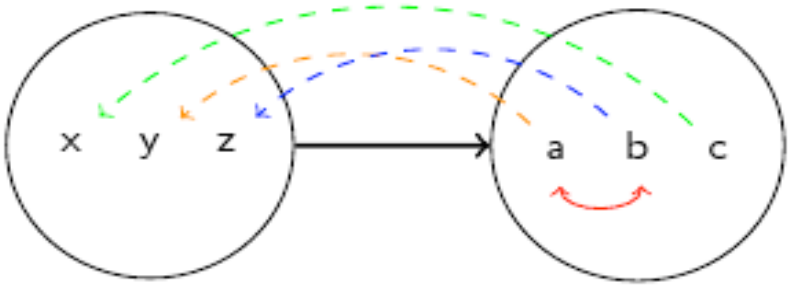


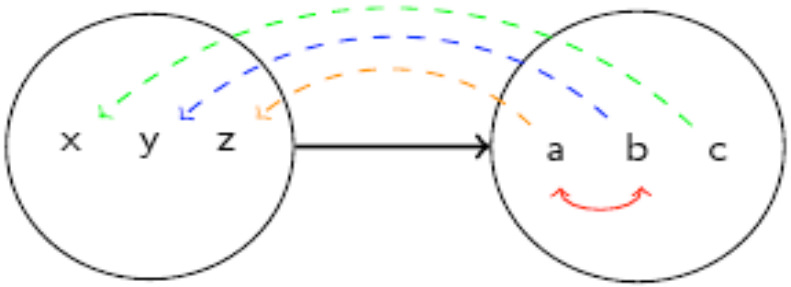
Figure: The action of a morphism on **a single** element of a named set

Avoid Observing too Much: states with name symmetries

If *a* and *b* are related by the symmetry, then morphisms should not distinguish them:



≡



Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

Nominal Sets as a category of algebras

Nominal Sets

give me a set, and tell me how injective substitution acts.
This is **all** that is needed to properly define *binding*

- *Autf* :finite-kernel permutations over the set of natural numbers ω, with axioms for identity and composition
  Names are identified with any countable set

- Algebras for the permutation signature are **permutation algebras**

$$\langle A, \{\pi_{\mathcal{A}} : A \to A \mid \pi \in Autf\}\rangle$$

A is the carrier set, $\pi_{\mathcal{A}}$ is the interpretation of the operation π

Axioms: $\pi\,(\pi'(x) = \pi \circ \pi'(x) \quad id(x) = x$

"Group-theoretical" properties

Nominal Sets

finitely supported permutation algebras $\mathbf{FSAlg}^{\pi}$
=
nominal sets by Gabbay and Pitts

- **Symmetry** (stabiliser, isotropy group) of $a \in A$:

$$\mathcal{G}_{\mathcal{A}}(a) = \{\pi \in Autf \mid \pi_{\mathcal{A}}(a) = a\}$$

- S **supports** a if $fix(S) \subseteq \mathcal{G}_{\mathcal{A}}(a)$
- S **is the support** $supp_{\mathcal{A}}(a)$ if it is finite and least supporting set
- The **orbit** of a is the set $\{\pi_{\mathcal{A}}(a) \mid \pi \in Autf\}$

"Nominal" transition systems

- A *labelled transition system* is a function $f : A \longrightarrow \mathcal{P}_{cnt}(L \times A)$ for a set of states A.
- If we require A and L to be permutation algebras, and f to respect the permutation action, we have obtained an *enriched transition system* featuring names in states, and along transitions.
- A is not a set! f is no longer called "transition system" but "coalgebra". **The theory of coalgebras in the category $\mathbf{FSAlg}^{\pi}$ is the theory of transition systems enriched with names**.
- Bisimulation is a standard notion. The final system is equipped with a permutation action (it is a **bialgebra**).

The symmetry grows, the support shrinks

Respecting the permutation action has two consequences:

- The symmetry **can never shrink** along morphisms.
- Consequently, the support **can never grow** along morphisms.
- Thus, we cannot generate new names. Actually we can, by defining a "name abstraction functor" as we shall see.

Consequence: the final system

- The *final* system (final coalgebra, minimal system) contains the minimum number of names **up-to bisimulation**
- The final system has the **greatest possible symmetry group** on its names
- A finite representation of the final system provides a representation of the greatest symmetry group over names of a system, up-to bisimulation. **History-dependent automata provide such a finite representation**.

Name Abstraction

- Using finitely supported permutation algebras (a.k.a. nominal sets) we have many different isomorphic versions of the **name abstraction functor** $\delta(\mathcal{A})$.
- The definition by Gabbay and Pitts [LICS 1999] (extended later to a functor, e.g. [Klin, MFPS07] $\sim$ quotient systems by α-conversion.
- Many other possible choices. In [Ciancia, Montanari - CMCS08] we present a pair of adjoint functors derived from a simple theory morphism. One adjoint is abstraction, the other one is concretion.
  1. It gives rise to abstract syntax using De-Bruijn indexes
  2. It is isomorphic to the notion by Gabbay and Pitts
  3. It is equivalent to the δ functor on $\mathbf{Set}^{\mathbb{I}}$ (from the literature on presheaves)
  4. It recovers the fully abstract semantics of the π-calculus using permutation algebras that was used by Montanari and Pistore

$$\delta((A,\{\pi_A\})) = (a,\{\pi_A{}^{+1}\}) \qquad \delta(f) = f$$

Infinite models

Two relevant sources of infinity

- Name allocation generates infinite systems:

$$P(x) = (\nu y)\bar{x}y.P(y)$$

- Input transitions (name passing) generate infinite systems, too:

$$a(x).P \xrightarrow{az} P\,[^{z}/_{x}] \qquad \textbf{for all } z$$

Question: how do we minimise a system featuring name passing and name allocation? How do we model check it?

Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

Named Sets and HD-Automata

- Named sets are a category for modelling name passing that is capable of garbage collection
- Coalgebras over named sets are called *history-dependent automata*
- Main feature: names are handled as bindable, local resources, rather than global constants
- Local names = relabellings along morphisms $\Longrightarrow$ garbage collection

Named Sets

- An object is a pair $\langle N, \mathcal{S} \rangle$.
  N is a set of elements, $\mathcal{S} : N \longrightarrow |\textbf{Symset}|$ is a map associating a symmetry to each element of N.

- An arrow from $\langle N_1, \mathcal{S}_1 \rangle \rightharpoonup \langle N_2, \mathcal{S}_2 \rangle$ is a pair $\langle h, \Sigma \rangle$.
  $h : N_1 \longrightarrow N_2$ is a function, and Σ is the representative of an equivalence class of injective relabellings

$$\Sigma(q) : [\mathcal{S}_2(h(q)) \rightarrow \mathcal{S}_1(q)]/_{\mathcal{S}_2(h(q))}$$

- A category of **sets** enriched with **local names**, their **symmetry**, and injective relabellings tracing the **history of names** along morphisms for each element

Definition 4.1 (Symset) *Let $Grp(S)$ be the set of permutation groups over S. The set of objects of the category* **Symset** *is $\bigcup_{S \in \mathcal{P}_{fin}(\omega)} Grp(S)$. An arrow from Φ_1 to Φ_2 is a set of functions $i \circ \Phi_1$ such that $i : dom(\Phi_1) \rightarrowtail dom(\Phi_2)$ and $\Phi_2 \circ i \subseteq i \circ \Phi_1$. We define $id_\Phi = id \circ \Phi = \Phi$ and $G \circ F = \{g \circ f \mid g \in G \wedge f \in F\}$.*

Equivalence between nominal sets and named sets

- From each permutation algebra we obtain a named set as the set of **canonical representatives** of an equivalence class:

$$\pi(q) \equiv q \quad \textbf{for all } \pi$$

- Example: $(\bar{x}y.0 \parallel \bar{y}x.0) \equiv (\bar{a}b.0 \parallel \bar{b}a.0) \equiv \bar{c}d.0 \parallel \bar{d}c.0$, ...
  all are represented by $\bar{x}y.0 \parallel \bar{y}x.0$

- From each named set we obtain a permutation algebra by **freely** re-generating the orbit:

$$\{\pi(q) \mid q \in N\}$$

- If we don't record the symmetry of q when we quotient, we will create **too many elements** when going back, e.g. because we consider $swap(x, y)(\bar{x}y.0 \parallel \bar{y}x.0)$ different from $\bar{x}y.0 \parallel \bar{y}x.0$.

Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

Four fundamental problems

When dealing with **automated verification** of basic name passing formalisms such as the π-calculus, we face four fundamental problems:

1. Dynamic allocation of names **(generation of fresh names)**
2. Deallocation of unused fresh names **(garbage collection)**
3. Composition of smaller systems into a bigger one
   (binding of local names among different systems)
4. Non-deterministic choice of a name over the infinite set of all names (both fresh and already known) **(name passing)**

We will see how these four problems can be solved using history-dependent automata, and how this is sufficient to model the π-calculus and the history-preserving semantics of place-transition Petri nets.

Abstraction "breaks" orbits

Generating fresh names

Back to permutation algebras:

- Abstraction *creates* new orbits
  $\Longrightarrow$ a faithful representation of abstraction must create new elements in a named set
- In principle, **one new orbit for each name** $i \in supp_{\mathcal{A}}(a)$. But this would allow to distinguish two symmetric names.
- **One new orbit for each name** $i \in supp_{\mathcal{A}}(a)/_{\equiv}$ where $\equiv$ is induced by the symmetry

$$i \equiv j \iff \exists \pi \in \mathcal{G}_{\mathcal{A}}(a).\pi(i) = j$$

The name abstraction functor

- Let $N = \langle Q_N, \mathcal{S}_N \rangle$. We define the abstraction functor H

$$Q_{H(N)} = Q_N \cup \left\{ \langle q, i \rangle \,\middle|\, q \in Q_N, i \in dom(\mathcal{S}_N(q))/_{\equiv} \right\}$$

- One new element for each "hideable" name
- The symmetry of each new element is the **subgroup** of the symmetry of q that fixes i

Exactly corresponds to

$\delta((A,\{\pi_A\})) = (a,\{\pi_A{}^{+1}\})$ $\quad\quad$ $\delta(f) = f$

The functor on arrows

- An element with an hidden name is mapped to an element with an hidden name **only if** the name is still present in the destination of $F = \langle h_F, \Sigma_F \rangle$

$$h_{H(F)}(\langle q, i \rangle) = \begin{cases} \langle h_F(q), j \rangle & \text{if } \exists \sigma \in \Sigma_F(q) . \sigma(j) = i \\ h_F(q) & \text{otherwise} \end{cases}$$

Garbage collecting unused names

In permutation algebras, it is easy to get an infinite system:

$$P(1) = (\nu x)\overline{1}x.P(1)$$

State	Support
$(\nu x)\overline{1}x.P(1)$	$supp = \{1\}$
$\downarrow$	
$(\nu x)\overline{2}x.P(2)$	$supp = \{1\}$
$\downarrow$	
$(\nu x)\overline{3}x.P(3)$	$supp = \{1\}$
$\downarrow$	
$\ldots$	$supp = \{1\}$

Figure: Infinite states in $\mathbf{FSAlg}^{\pi}$

Garbage Collection

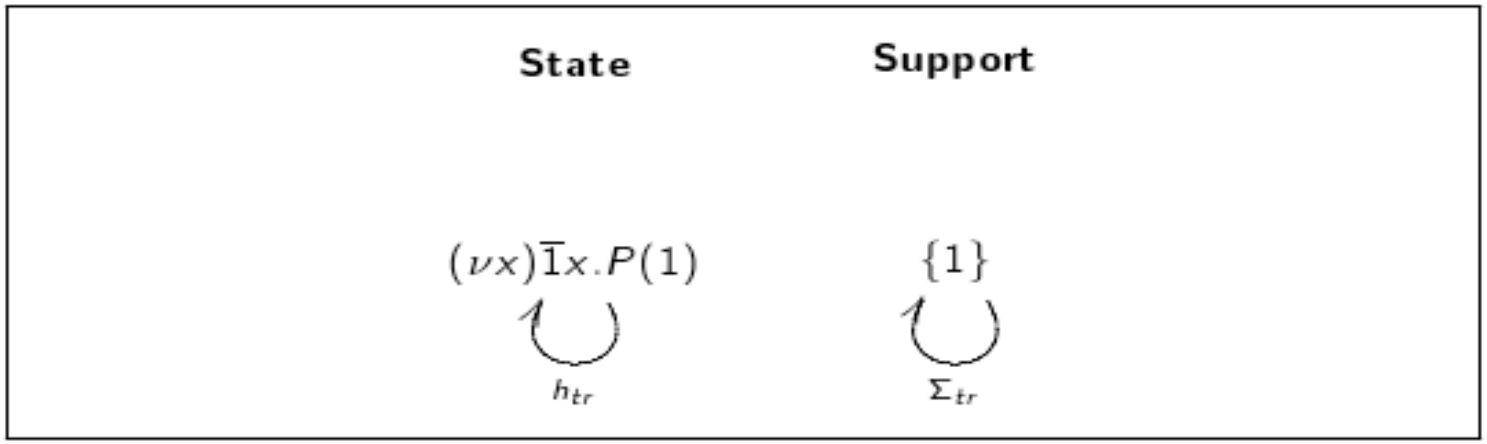


Figure: HD-automaton having finite states

Garbage Collection

Another example using presheaves: $R(1) = (\nu x)\bar{1}x.R(x)$

Support	Stage in the coalgebra	State
$\{1\}$	$\{1\}$	$(\nu x)\bar{1}x.R(x)$
		$\downarrow$
$\{*\}$	$\{1\} + 1 = \{1, *\}$	$(\nu x)\bar{*}x.R(x)$
		$\downarrow$
$\{*'\}$	$\{1, *\} + 1 = \{1, *, *'\}$	$(\nu x)\overline{*'}x.R(x)$
		$\downarrow$
		$\ldots$

Figure: Infinite states in $\mathbf{Set}^{\mathbb{I}}$

The product: intuition

Product in named sets

- If names are **local**, things are more complicated: we have to establish a mapping of names that identifies some names in the pair, and keeps other names distinguished, e.g. in the form of a **pair of injections into a common target**

$$(\bar{x}y.0 \parallel \bar{y}x.0) \xrightarrow{\{x\mapsto a, y\mapsto b\}} \{a, b\} \xleftarrow{\{x\mapsto a, y\mapsto b\}} (\bar{x}y.0 \parallel \bar{y}x.0)$$

$$\neq$$

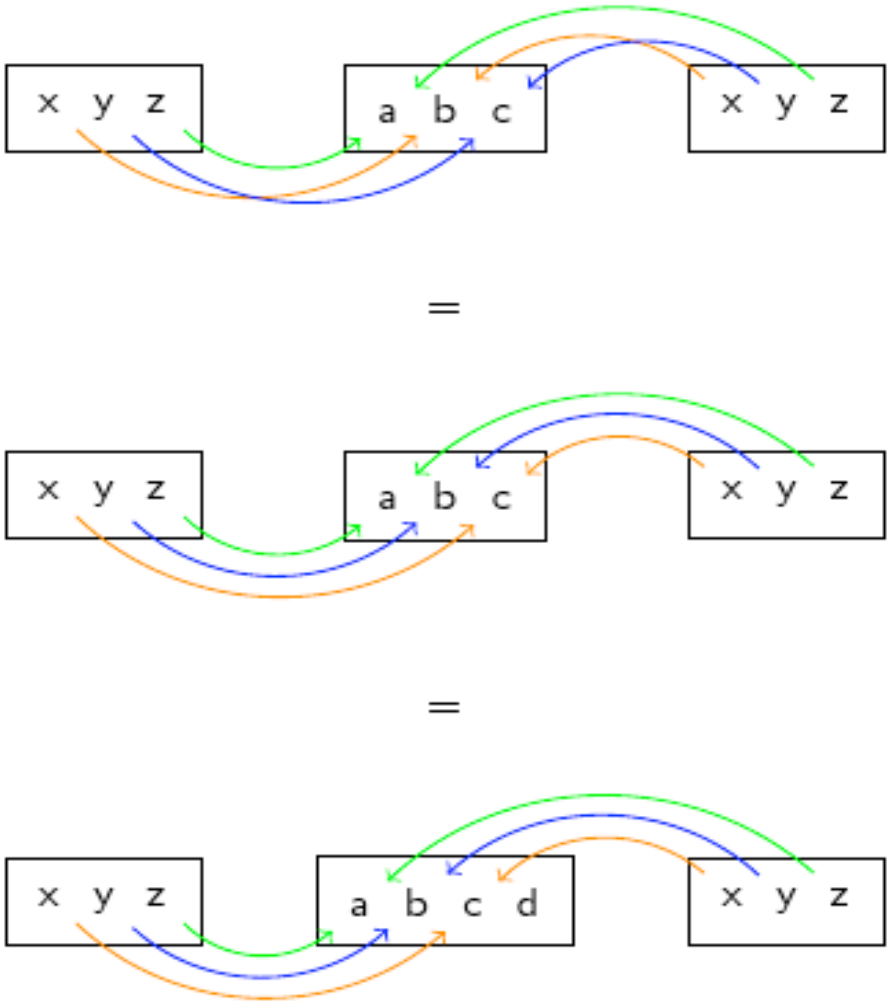
$$(\bar{x}y.0 \parallel \bar{y}x.0) \xrightarrow{\{x\mapsto a, y\mapsto b\}} \{a, b, c\} \xleftarrow{\{x\mapsto b, y\mapsto c\}} (\bar{x}y.0 \parallel \bar{y}x.0)$$

Name mappings as cospans

- To represent name mappings we use *cospans*, that is pairs of mappings with a common target
- These cospans must keep in account symmetries, so they are not injections, but rather arrows in Symset
- The target object must be **minimal** and **uniquely determined**: names that are not in the image of at least one arrow are not interesting

Multi-coproduct

The multi-coproduct $MCP(\Phi_1, \Phi_2)$ [Diers, 1979] is a categorical construction that identifies all isomorphic cospan, and only find the minimal ones



The categorical product

- The *categorical product* of $N = \langle Q_N, \mathcal{S}_N \rangle$ and $M = Q_M, \mathcal{S}_M$ is given by $\langle Q, \mathcal{S} \rangle$, where:

$$Q = \{ \quad \langle n, m, \langle in_1, in_2 \rangle \rangle \mid n \in Q_N \wedge m \in Q_M \\ \wedge \langle in_1, in_2 \rangle \in MCP(\langle \mathcal{S}_N(n), \mathcal{S}_M(m) \rangle) \quad \}$$

$$\mathcal{S}(\langle n, m, \langle in_1, in_2 \rangle \rangle) = cod(in_1) = cod(in_2)$$

- The projections $\pi_1 = \langle h_1, \Sigma_1 \rangle$ and $\pi_2 = \langle h_2, \Sigma_2 \rangle$ are defined as

$$h_1(t) = n \qquad \Sigma_1(t) = in_1$$
$$h_2(t) = m \qquad \Sigma_2(t) = in_2$$

Consequences on bisimulation and model checking

- The definition of the product has consequences on the definition of the bisimulation problem and the model checking problem
- **Bisimulation** becomes a ternary relation, employing two states and a name mapping between them. **History-preserving (causal) bisimulation is ordinary coalgebraic bisimulation**. Names that are not mapped are **redundant**.
- The **satisfaction relation** between states and (modal) formulas is a ternary relation in turn. Names that are not mapped are not involved in the particular instance of the satisfaction definition
- Consequence: the definition of satisfaction will be the standard one, but new algorithms will have to be invented to efficiently handle these ternary relations. Also for minimisation and bisimulation checking.

In real world: lack of a naming authority

- The model that we have presented is typical in situation where the **global** meaning of names is not established a priori
- Example: take two isolated networks, and connect their ports: **a mapping of the external ports between the two networks has to be established**: a way to "name" ports of each network "into" the other one.
- Example: code fragments in a programming language: **x=3+2** and **y=3+2** have absolutely the same meaning, **until** I relate two pieces of code. **int x** has the same meaning of **int y**.

Inputting names

The problem of non-deterministic choice quantified over the set of all names arises when modelling **name-passing** operations

- For example, input in the *early* semantics of the π-calculus: $x(y).P \xrightarrow{xz} P\,[^{z}/_{y}]$ **for all** z
- Method calls in object-oriented languages (because of generation of new objects)
- A common solution is to consider **all free names** plus **an additional name which represents all the fresh names that can be received**.
- We have a finite number of **free input transitions** plus a **new type** of transition: *bound input transitions* that are not present in the original semantics of the π-calculus.

Two bisimilar systems

Consider the following bisimilar systems:

$$P(x,y) = x(z).\bar{z}z.0 \parallel (\nu w)\bar{w}y.0 \qquad \textbf{redundant name } y$$
$$Q(x) = x(z).\bar{z}z.0$$

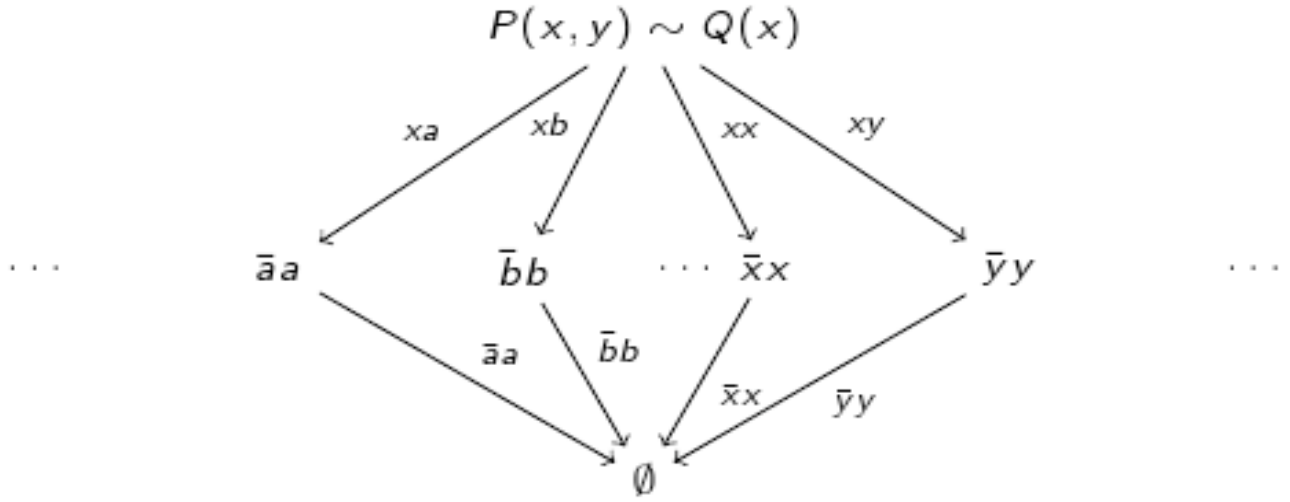


Figure: standard LTS

The approximation is incorrect

The approximation distinguishes P and Q even though they are bisimilar. There are **redundant transitions** that should not be taken into account.

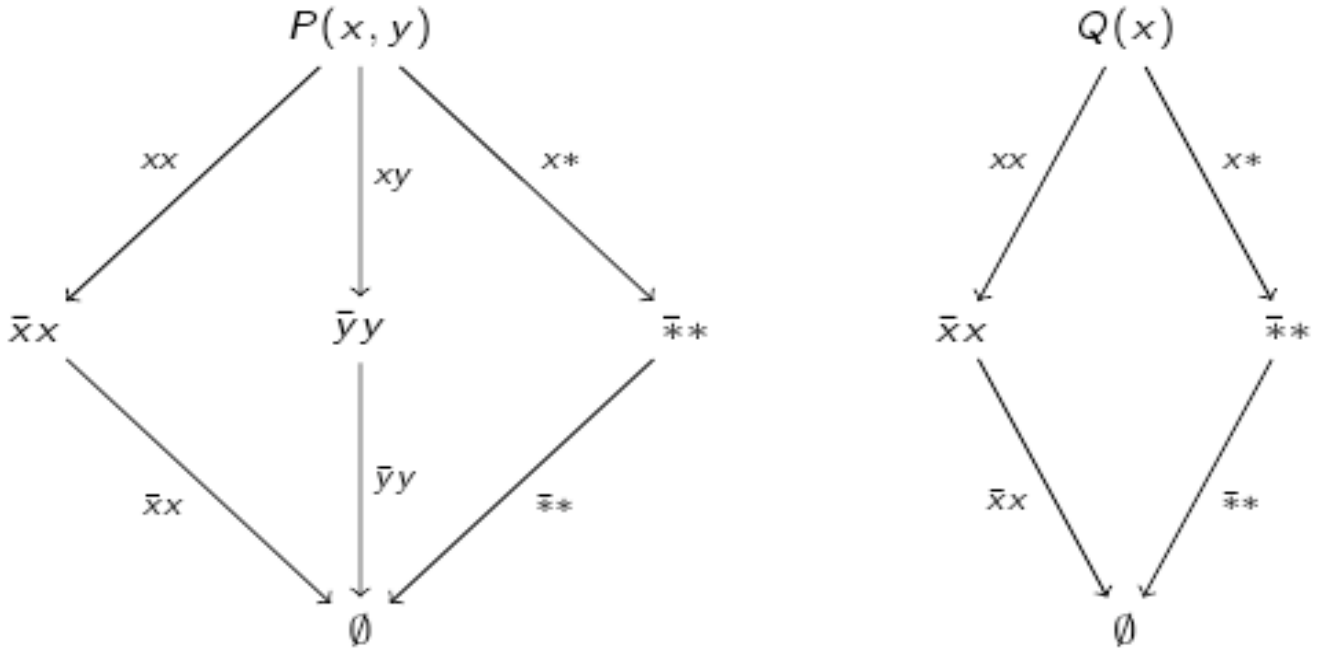


Figure: The incorrect approximation

The solution of MIHDA

Redundant names are not decidable

- Not only in the π-calculus! Example in pseudo-code: Is *a* redundant?
  if (nondecidable()) then a:=3 else skip;

- The solution given in Emilio Tuosto's thesis (MIHDA [Ferrari,Montanari,Tuosto - TCS 2005]): the notion of **dominated transition** identifies and prunes transitions that are proved to be **redundant in one step**

- The minimisation (partition refinement) algorithm then removes **all redundant transitions** at the ω step
  $\Longrightarrow$ Correct minimisation procedure and **side effect**: remove all redundant names

Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

A general formulation

- The solution of MIHDA is specific for the π-calculus, and in particular it seems tightly bound to the set of labels of the π-calculus
- We thus define a **finitely representable** subfunctor $\mathcal{P}_{fr}(X)$ of the **countable** power set $\mathcal{P}_{cnt}(X)$ whose representation **on** objects **can** be given as

$$\mathcal{P}_{fin}(\delta(X))$$

- The functor on arrows removes elements that are proved redundant **in one step**.

Two theorems

- We show that our functors are all **accessible**, this implies that they have a final coalgebra
- We show that the categories of coalgebras of **equivalent functors** are **equivalent.**
- We show that our functors are equivalent to the corresponding notions in permutation algebras and presheaves
- **Conclusion:** for each semantics that is given using permutation algebras, nominal sets or presheaves, using **product, coproduct, name abstraction, finite powerset and nondeterministic choice quantified over names,** we obtain **constructively** a semantics in named sets, and ...
- ... the **abstract** semantics can be **computed** in many cases, due to the garbage collection property of named sets, thus we can actually make use of fully abstract logics, model checking algorithms exploiting Stone duality, bisimulation checking methods using iteration along the terminal sequence and so on.

Roadmap

- Models for nominal process description languages
- The basic idea of named sets
- Permutation algebras
- Named sets
- Operations on named sets
  - Generating fresh names and garbage collecting unused ones
  - Product in named sets
  - Inputting names
- Generalizing MIHDA
- Conclusion

Extending the equivalence to richer index categories

- We have deeply studied and exploited the equivalence between a full subcategory of $\mathbf{Set}^{\mathbf{I}}$, a category of algebras over an infinite signature, and the category of named sets.
- The idea of presheaves (and algebras) is that objects of the index category (finite sets in this case) are *interfaces* that elements have, and arrows are *operations* that can be applied to these interfaces.
- **Future work:** enrich the interface. E.g. fusions, graphs, etc.
- Open problem: **how does binding work in the presence of fusions? What is abstraction of a graph?** We should investigate a generalised notion of binding.

Efficient algorithms for the product

- The structure of the product is fundamental in bisimulation checking and model checking
- It is already implicitly used in MIHDA, but not efficiently: the symmetry is represented using generators, but the permutation group is **unrolled** at each step of the algorithm, thus throwing away a potentially exponential gain
- Eugene M. Luks developed efficient algorithms for permutation groups, and preliminary results show applicability to efficient representations of the product of named sets
- Future work: implement these algorithms in a generalised version of MIHDA, and in a model checking algorithm.
- Optimal symmetry reduction in model checking!

Sessions and coordination languages

- One of the most important applications of name generation is that of *sessions* in service-oriented computing
- Verification for calculi with sessions suffer from the same problems that we mentioned, thus the solutions we propose are applicable in this area
- In particular, we expect to be able to implement formal verification techniques, based on history-dependent automata, for the *Service Calculus* (SC) [Ferrari, Guanciale, Strollo - FORTE 2006], [F., G., S. , Tuosto - FORTE 2007], [F., G., S., Ciancia - FORTE 2008] ... and for the coordination language NCP
- SC already has a visual editor and a Java implementation (JSCL). Particularly appealing: **introduce verification in the form of graphical tools** that e.g. visualize and allow to replay erroneous traces.