BRICS

Basic Research in Computer Science

BRICS NS-03-3 Aceto et al. (eds.): PA '03 Slide Reprints

Slide Reprints from the Workshop on

Process Algebra: Open Problems and Future Directions

PA '03

Bologna, Italy, 21–25 July, 2003

Luca Aceto
Zoltán Ésik
Willem Jan Fokkink
Anna Ingólfsdóttir
(editors)

BRICS Notes Series **NS-03-3**

ISSN 0909-3206 **November 2003**

See back inner page for a list of recent BRICS Notes Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `NS/03/3/`

Slide Reprints from the Workshop on

Process Algebra: Open Problems and Future Directions

Bologna, Italy, 21–25 July, 2003

Luca Aceto
Zoltán Ésik
Wan Fokkink
Anna Ingólfsdóttir

Foreword

This volume of the BRICS notes series contains reprints of the slides of most of the talks that were delivered during the workshop on "Process Algebra: Open Problems and Future Directions" that was held in the period 21–25 July, 2003, at the University Residential Centre of Bertinoro, Forlì, Italy.

This was a lively scientific event, held in a relaxed workshop atmosphere that allowed for an informal, but intense, discussion of the status of research in the field of process algebra, broadly construed. The workshop has witnessed the continuing vitality of this branch of concurrency theory, and we trust that, apart from being a celebration of over twenty years of research in this field, it will contribute to its healthy development by highlighting some open problems, and possible new avenues for research. We believe that the slides of the talks collected in this volume will offer the process algebra community at large some inspiration for reflection on past achievements, and some suggestions for further research. Our efforts in organizing this workshop will be amply rewarded by the solution of some of the open problems that were raised during the event, or by the further development of work along the future directions that were pointed out in Bertinoro.

As mentioned above, this workshop was held in the beautiful setting of the University Residential Centre of Bertinoro under the sponsorship of BICI, the *Bertinoro International Center for Informatics*. BICI is an association whose mission is to foster cutting-edge research and advanced education (at PhD and post-doctoral level) in Computer Science. BICI sponsored events, like our workshop, take place in Bertinoro at the University Residential Centre of the University of Bologna. Typical events sponsored or organized directly by BICI include thematic research workshops, strategic meetings charting new research agenda and advanced schools.

We welcome the establishment of such an association devoted to the development of research in Computer Science via the sponsorship of high quality events in an environment that offers excellent support, and a congenial atmosphere, for the hosting of research activities. We encourage our colleagues interested in organizing workshops on all aspects of Computer Science to consider the University Residential Centre of Bertinoro as a possible location for their events.

In addition to BICI, we received sponsorship from BRICS and CWI. We thank both these institutions for their generous financial assistance. Our gratitude goes to all of our colleagues that made the trip to Bertinoro, and contributed to the success of the workshop. Last, but not least, our thanks go to Elena Della Godenza (University Residential Centre of Bertinoro) for her tireless organizational and secretarial assistance at all times, and to Uffe Engberg (BRICS) for his work in the production of this volume.

LUCA ACETO (Aalborg, Denmark)
ZOLTÁN ÉSIK (Szeged, Hungary)
WAN FOKKINK (Amsterdam, The Netherlands)
ANNA INGÓLFSDÓTTIR (Aalborg, Denmark, and Reykjavík, Iceland)
Workshop Organizers and Editors of this Volume

Contents

Max-plus quasi-interpretations

Roberto Amadio

University of Marseille

- *Process algebra* broadly conceived as design principles for composing sequential programs.
- This talk is –not– about process algebra but the motivations are.

NB We are interested in composing programs –not– specifications.

Wish list 1: determinism

System behaviour is deterministic – explicit account of scheduling.

motivation debugging, portability, no locking,...

examples Kahn networks, cooperative threads.

Wish list 2: synchrony

Processes share a common time scale – there is notion of instant.

motivation can react to absence of reply, can share a signal for an instant, bounded buffers,...

examples Synchronous Kahn networks (Caspi-Pouzet), Cooperative threads with round robin and broadcast events (Boussinot).

Wish list 3: reactivity

A system receiving data of bounded size runs in bounded memory and reacts in bounded time.

motivation embedded programming.

examples Lustre, Esterel,...

Wish list 4: flexibility

Can handle arbitrary inductive data types and virtual machine accomodates mobile code.

- No model from the past seems to accomodate this.
- Future: design a coordination language.
- Present=this talk: what can be said of the sequential modules.

Context

- Complexity bounds for (first-order) functional programs.
- One 'classical' thread: *functional algebras* characterization of *small* complexity classes.
- A more 'technological' thread: worry also about *algorithms* and automatic extraction of *certificates*.

Example: insertion sort over binary words

$\mu t.(\epsilon : t, 0 : t \to t, 1 : t \to t)$ (binary words)

$$insert_0(x) = 0(x)$$

$$\begin{aligned} & insert_1(x) = \\ & x = \epsilon \Rightarrow 1(\epsilon) \\ & x = 1(x') \Rightarrow 1(1(x')) \\ & x = 0(x') \Rightarrow 0(insert_1(x')) \end{aligned}$$

$$\begin{aligned} & sort(l) = \\ & l = \epsilon \Rightarrow \epsilon \\ & l = \mathsf{i}(x) \Rightarrow insert_{\mathsf{i}}(sort(x)) \quad \mathsf{i} = 0, 1 \end{aligned}$$

Ramification (Bellantoni-Cook and Leivant)

- $f(\vec{x}; \vec{y})$: split arguments in *Normal* ($\vec{x}$) and *Safe* ($\vec{y}$).
- $N \leq S$: *Normal* can be regarded as a *subtype* of *Safe*.
- $f(\mathsf{i}x \ldots ; \ldots) \Rightarrow h(\ldots ; f(x, \ldots ; \ldots), \ldots)$.
  Recurrence parameters are *Normal*, *Result of a recurrence* is *Safe* ($\Rightarrow$ typing of *exponential* fails).
- *Constructors* are overloaded, sending safe to safe and normal to normal.
- *Composition*: $g(h_1(\vec{x}; \_); h_2(\vec{x}; \vec{y}))$.

Problem (cf. Caseiro): Some simple algorithms such as insertion sort do not type.

A first-order functional language

- Inductive types

$$\mu t.(\ldots \mathsf{c} : \tau_1, \cdots, \tau_n \to t, \ldots)$$

- Values, patterns, expressions

$$\begin{aligned} v &::= \mathsf{c}(v, \ldots, v) \\ p &::= x \mid \mathsf{c}(p, \ldots, p) \\ e &::= x \mid \mathsf{c}(e, \ldots, e) \mid f(e, \ldots, e) \end{aligned}$$

- Functions definitions by pattern matching and evaluation by value.

$$\begin{aligned} & f(x_1, \ldots, x_n) = \\ & \ldots \\ & x_1 = p_1, \ldots, x_n = p_n \quad \Rightarrow e \\ & \ldots \end{aligned}$$

Some history: bounded recursion on notation (Cobham)

$$\begin{aligned} & f(x, \vec{y}) && = \\ & x = \epsilon && \Rightarrow g(\vec{y}) \\ & x = \mathsf{i}x' && \Rightarrow h_{\mathsf{i}}(f(x', \vec{y}), x', \vec{y}) \qquad \mathsf{i} = 0, 1 \\ & \text{with} && |f(x, \vec{y})| \leq P(|x|, |\vec{y}|), P \text{ polynomial.} \end{aligned}$$

Problem Has to stick to *primitive recursion* and *guess* the polynomial P.

Insertion sort does not type

$$insert_0(x;) = 0(x)$$

$$\begin{aligned} & insert_1(x;) = \\ & x = \epsilon \Rightarrow 1(\epsilon) \\ & x = 1(x') \Rightarrow 1(1(x')) \\ & x = 0(x') \Rightarrow 0(insert_1(x';)) \end{aligned}$$

$$\begin{aligned} & sort(l;) = \\ & l = \epsilon \Rightarrow \epsilon \\ & l = \mathsf{i}(x) \Rightarrow insert_{\mathsf{i}}(sort(x;);) \quad \mathsf{i} = 0, 1 \end{aligned}$$

$insert_1$ waits for normal but gets safe.

Jones' no cons condition

General recursive programs coupled with 'implicit' way to bound the size of the results.

- No constructors of positive arity on the right-hand side of the rule.
- Enough to characterize PTIME *problems*.
- Simple algorithm such as list reverse cannot be represented.

Hofmann's type system for *in-place update*

- Relies on an –empty– resource type ρ and *affine* typing.
- An element of resource type is understood as a memory cell.
- Constructors take an extra-argument of type ρ. Also functions may get extra-arguments of type ρ.
- In a rule $x_1 = p_1, \ldots, x_n = p_n \Rightarrow e$, resources have to be balanced:
$$\Gamma \vdash p_i, i = 1, \ldots, n \quad \Rightarrow \quad \Gamma \vdash_{\mathit{aff}} e$$
- Data transformations are *non-size increasing* and language can be compiled so that no dynamic heap memory allocation is required.

Example: insertion sort with resource types

$$\begin{array}{ll} W & = \mu t.(\epsilon : t, \mathsf{0} : \rho, t \to t, \mathsf{1} : \rho, t \to t) \\ \mathit{insert}_{\mathsf{i}} & : \rho, W \to W \quad \mathsf{i} = 0,1 \\ \mathit{sort} & : W \to W \end{array}$$

$\mathit{insert}_{\mathsf{0}}(r, x) = \mathsf{0}(r, x)$

$\ldots$

$\mathit{sort}(l) =$

$l = \epsilon \Rightarrow \epsilon$

$l = \mathsf{i}(r, x) \Rightarrow \mathit{insert}_{\mathsf{i}}(r, \mathit{sort}(x)) \quad \mathsf{i} = 0, 1$

Problem Would like to –infer– the resource types.

Quasi-interpretations (Marion *et al.*)

Given a program interpret constructors and functions of arity n as follows:

$$q_{\mathsf{c}} = \begin{cases} 0 & \mathsf{c} \text{ constant} \\ d + \Sigma_{i=1,\ldots,n} x_i & \text{otherwise, with } d \geq 1 \end{cases}$$

$$q_f : (\mathbf{Q}^+)^n \to \mathbf{Q}^+ \text{ monotonic and } q_f \geq \pi_i$$

Quasi-interpretation (continued)

Obvious extension of assignment q to expressions:

$$q_x = x$$
$$q_{\mathsf{c}(e_1,\ldots,e_n)} = q_{\mathsf{c}}(q_{e_1}, \ldots, q_{e_n})$$
$$q_{f(e_1,\ldots,e_n)} = q_f(q_{e_1}, \ldots, q_{e_n})$$

An assignment q is a quasi-interpretation if:

for every rule $f(p_1, \ldots, p_n) \Rightarrow e$

$$q_f(q_{p_1}, \ldots, q_{p_n}) \geq q_e$$

NB Quasi-interpretations are inspired by *polynomial simplification interpretations* for termination proofs.

Quasi-interpretation for the insertion sort

It admits the following quasi-interpretation:

$$q_{\mathsf{i}} = x + 1, \quad q_{\mathit{sort}} = x, \quad q_{\mathit{insert}_{\mathsf{i}}} = x + 1.$$

For instance, the rule

$$\mathit{sort}(\mathsf{i}(x)) \Rightarrow \mathit{insert}_{\mathsf{i}}(\mathit{sort}(x))$$

satisfies

$$q_{\mathit{sort}}(q_{\mathsf{i}}(x)) = x + 1 \geq x + 1 = q_{\mathit{insert}_{\mathsf{i}}}(q_{\mathit{sort}}(x))$$

Basic properties of quasi-interpretations

1. $|v| \leq q_v \leq d|v|$, for v value, d constant.
2. $e \mapsto v$ then $q_e \geq q_v \geq |v|$.
3. If there is a quasi-interpretation q then $f(v_1, \ldots, v_n)$ can be evaluated with an activation record of size B.
4. Or, equivalently by Cook's theorem, in time 2^B where $B = O(q_{f(v_1,\ldots,v_n)})$.

NB In particular, if q_f is *linear* in the size n of the input then the size of an *activation record* is $O(n)$ and the program can be run in time $2^{O(n)}$.

A simple call-by-value evaluator

$Eval(e) = \mathsf{case}$

$e \equiv E[f(v_1, \ldots, v_n)], f(p_1, \ldots, p_n) \Rightarrow e',$ and $\sigma(p_j) = v_j$:

  let $v' = Eval(\sigma(e'))$ in $Eval(E[v'])$

e value : e

else : $Return\ \bot$

NB This program can be run on a linearly bounded APDA and, by Cook's theorem, it can be transformed to run in EXPTIME.

An evaluator with *memoization*

$Eval_m(e) = \mathsf{case}$

$e \equiv E[f(v_1, \ldots, v_n)], f(p_1, \ldots, p_n) \Rightarrow e',$ and $\sigma(p_j) = v_j$:

  $(new, v'') := Insert(f(v_1, ..., v_n));$

  case

  new : let $v' = Eval_m(\sigma(e'))$ in

    $Update(f(v_1, ..., v_n), v');$

    $Eval_m(E[v'])$

  $\neg new, v'' \neq \bot$: $Eval_m(E[v''])$

  else : $Return\ \bot$

e value : e

else : $Return\ \bot$

Quasi-interpretation for no cons programs

A program conforming to Jones' restriction admits the following multi-linear quasi-interpretation

$$q_{\mathsf{c}} = 1 + \Sigma_{i=1,\ldots,n} x_i \qquad q_f = max(x_1, \ldots, x_n)\ .$$

NB Affine typing may fail here.

Search space: max-plus polynomials

- We shift from the algebra $(+, \times)$ to the algebra $(max, +)$.
- Work over $\mathbf{Q}^+_{max} = \mathbf{Q}^+ \cup \{-\infty\}$. $-\infty$ is the unit of max and 0 is the unit of $+$.
- Distribution: $x + max(y, z) = max(x + y, x + z)$.
- Exponentiation: αx.
- For a –given– degree the *synthesis problem* can be expressed as the validity of a $\exists\forall$ Pressburger formula.

NB We look for something more efficient…

Quasi-interpretation for programs with affine typing

If a program has an *affine typing* then its *erasure* of resource arguments admits the following multi-linear quasi-interpretation:

$$q_{\mathsf{c}} = 1 + \Sigma_{i=1,\ldots,n} x_i \qquad q_f = r(f) + \Sigma_{i=1,\ldots,n} x_i$$

where $r(f)$ is the number of resource arguments of f.

Lower bound on complexity of synthesis

Prop The synthesis problem is NP-hard, and it stays so for any combination of the following:

1. Rules of bounded size (for a small bound).
2. Max-plus polynomials of bounded degree $d \geq 1$.
3. Uniform choice of constructors' coefficients.

Outline proof lower bound

1. Write rules so that can force $q_f = max(x_1, \ldots, x_n)$.
2. Reduction from $3 - SAT$:
   - Non-uniform choice of constructors: code literals as coefficients of unary constructors.
   - Uniform choice: write rules so that can force $q_g = max(x_1 + a_1, x_2 + a_2)$ and use coefficients a_1, a_2 to represent literals.

Multi-linear max-plus polynomials

- Multi-linear = Degree of every variable is at most 1.
- A generic multi-linear of n variables is determined by 2^n coefficients. *E.g.* for $n = 2$:

$$max(a_0, x_1 + a_1, x_2 + a_2, x_1 + x_2 + a_{1,2})$$

- One can always normalise so that

$$a_0 \geq a_1, a_2 \geq a_{1,2}$$

  and then compare polynomials by comparing the coefficients.

Upper bound on complexity of synthesis

Prop The synthesis problem for multi-linear polynomials for programs with rules of bounded size is NP-complete.

1. Compute the interpretations of $q_{f(p_1,\ldots,p_n)}$ and q_e and reduce to the satisfaction of a system of inequalities over $\mathbf{Q}^+_{max}$.

2. Use *non-determinism* to eliminate *max* from $q_{f(p_1,\ldots,p_n)}$ on the *left-hand* side of the inequality.

$$max(A, B) \geq C \text{ becomes } (A \geq C \wedge A \geq B) \vee (B \geq C \wedge B \geq A)$$

Eliminate *max* on the *right-hand* side q_e in polynomial time. Idea:

$$A \geq max(B, C) \text{ becomes } A \geq z, z \geq B, z \geq C$$

Upper bound (continued)

3. Get a system with constraints of the shape:

$$x = -\infty \qquad y \geq 1$$

$$x + \Sigma_{j=1,\ldots,l}\alpha_j y_j \geq \Sigma_{j=1,\ldots,n}\beta_j x_j + \Sigma_{j=1,\ldots,l}\gamma_j y_j$$

4. Send to $-\infty$ all the variables for which no $x \geq 0$ constraint can be inferred. Idea on *boolean* variables: satisfaction of formulae $\bigvee_{j \in J} x_j$ or $x \Rightarrow \bigvee_{j \in J} x_j$ can be decided efficiently.

5. Hence reduce to a *linear programming* problem over $\mathbf{Q}^+$ (it is possible to look for *optimal* solutions).

NB If the size of the rules is not bound then the method requires *exponential space* just to write the solution.

Lower bounds on expressivity: Qbf

$$qbf(\phi) = \quad check(\phi, nil)$$

$$\begin{array}{ll} check(\phi, l) = & \\ \phi = \mathsf{v}(x) & \Rightarrow mem(x, l) \\ \phi = \mathsf{o}(\phi', \phi'') & \Rightarrow or(check(\phi', l), check(\phi'', l)) \\ \phi = \mathsf{all}(x, \phi') & \Rightarrow and(check(\phi', \mathsf{cons}(x, l)), check(\phi', l)) \end{array}$$

Quasi-interpretation:

$$q_{\mathsf{v}} = x + 1, \qquad q_{\mathsf{o}} = q_{\mathsf{all}} = x + y + 1, \quad q_{qbf} = x,$$
$$q_{or} = q_{mem} = max(x, y), \quad q_{check} = \phi + l$$

Quasi-interpretations and termination ordering

- Quasi-interpretations are most useful when combined with a termination ordering.
- Consider the class of programs that admit a polynomially bound quasi-interpretation (usual polynomial here).
- Marion et al: the programs that terminate by product recursive path order characterize PTIME and those that terminate by lexicographic recursive path order characterize PSPACE.

Some perspective

- Cobham: Primitive recursion plus polynomial bound on size gives you polynomial time. Yes, but how do you find the polynomial?
- Ramification: You can use a type system, the polynomial is implicit. Yes, but many algorithms will not type!
- Hofmann: Use a type system to bound the size of the data. You'll loose functions which have super-linear space growth but gain some algorithms.
- Max-plus quasi-interpretations: Instead of types, use *small polynomials* to bound the size of data.

NB Hofmann's type system for *in-place update* and Jones' *no cons* syntactic restriction correspond to certain *polynomial* classes of multi-linear quasi-interpretations.

Lower bound on expressivity for general recursion: exponential time TM

- Can also simulate TM running in $2^{O(n)}$.
- Define

$$T : \mathit{Input} \times \mathit{Step} \times \mathit{Position} \to \mathit{State} \times \mathit{Letter}$$

- $T(x, s, p) = (q, a)$ iff the machine with input x after s steps arrives in state q with character a at position p.
- s, p can be stored in space $O(|x|)$ and we can do basic arithmetic modulo $2^{O(|x|)}$.
- $T(x + 1, s, p)$ can be defined recursively in terms of $T(x, s, p - 1)$, $T(x, s, p)$, $T(x, s, p + 1)$.

NB Again, this is a rephrasing of Cook's theorem (from EXPTIME to APDA).

TU/e technische universiteit eindhoven

Over 30 years of process algebra: past, present and future

Jos Baeten, TU/e

TU/e technische universiteit eindhoven

What is a process algebra?

From universal algebra:

A group has a signature $(G,*,u,^{-1})$ with laws

- $a*(b*c) = (a*b)*c$
- $u*a = a = a*u$
- $a*a^{-1} = a^{-1}*a = u$

TU/e technische universiteit eindhoven

Definition

A process algebra is any mathematical structure satisfying the PA axioms

A process is an element of a process algebra

A process algebra allows calculation on/with processes

We consider only concurrency

TU/e technische universiteit eindhoven

History (situation in 1970)

Denotational semantics (Scott - Strachey)

Operational semantics (McCarthey)

Axiomatic semantics (Floyd - Hoare)

? Parallel composition

TU/e technische universiteit eindhoven

Main differences

A program is an input/output function

A state is a valuation of variables

TU/e technische universiteit eindhoven

Hans Bekič (1936 - 1982)

IBM Vienna

Towards a mathematical theory of processes

December 1971

TU/e technische universiteit eindhoven

A quote:

Our plan to develop an algebra of processes may be viewed as a high-level approach: we are interested in how to compose complex processes from simpler (still arbitrarily complex) ones.

Null, action, or, //

TU/e technische universiteit eindhoven

Law for quasi-parallel composition

(A // B)ξ =

(cases Aξ: null → Bξ
 (f,A') → f,(A' // B))

or

(cases Bξ: null → Aξ
 (g,B') → g,(A // B'))

TU/e technische universiteit eindhoven

In a letter from 1975:

fulfil certain desirable equivalences, such as:

a;0 = a   a;(b;c) = (a;b);c   a//b = b//a

etc.

A lecture on this in 1974 even contains a “left-parallel” operator, with laws !

TU/e technische universiteit eindhoven

Robin Milner (1934), from 1973:

Non-terminating programs with side effects

Non-deterministic programs

Parallel programs

*   ? ||

TU/e technische universiteit eindhoven

Calculus of Communicating Systems

Gradually developed 1973 - 1980

Static laws

Ports: names and co-names

τ is communication trace

Expansion law

Bisimulation by David Park 1981

TU/e technische universiteit eindhoven

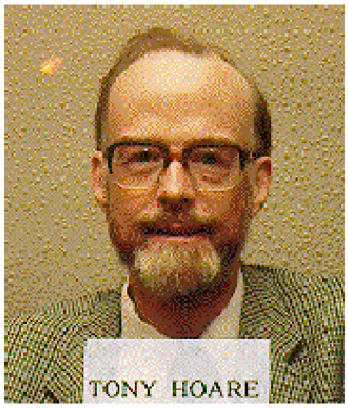
Tony Hoare (1934), from 1976:

Communicating Sequential Processes:

message passing

guarded command language

trace theory



Theoretical CSP, 1984:

With Stephen Brookes and Bill Roscoe

Failure semantics

τ-jump and two alternative composition operators

$\tau x + y = \tau x + \tau(x + y)$ $\Delta x + \Delta y = \Delta(x + y)$

technische universiteit eindhoven

Jan Bergstra & Jan Willem Klop

1982, from work of Jaco de Bakker and Jeff Zucker on metric concurrency theory

1. PROCESS ALGEBRAS

In this section we introduce process algebras and their projections, fix some terminology and notations, and establish some useful algebraical identities valid in process algebras.

1.1. <u>Process algebras: preliminaries.</u>

1.1.1. <u>DEFINITION</u>. Let $A = \{a_i \mid i \in I\}$ be some set of atomic "actions".

A *process algebra over* A is a structure $\mathcal{A} = \langle A, +, \cdot, \lfloor\!\lfloor, \underline{a}_i\ (i \in I)\rangle$ where A is a set containing A, the $\underline{a}_i$ are constant symbols corresponding to the $a_i \in A$, and + (*union*), $\cdot$ (*concatenation* or *composition*), $\lfloor\!\lfloor$ (*left merge*) satisfy for all $x,y,z \in A$ and $a \in A$ the following axioms:

PA1	$x+y = y+x$
PA2	$x+(y+z) = (x+y)+z$
PA3	$x+x = x$
PA4	$(xy)z = x(yz)$
PA5	$(x+y)z = xz+yz$
PA6	$(x+y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$
PA7	$ax \lfloor\!\lfloor y = a(x \lfloor\!\lfloor y + y \lfloor\!\lfloor x)$
PA8	$a \lfloor\!\lfloor y = ay$

1.1.1.1. <u>NOTATION</u>. We write xy instead of $x \cdot y$ and a instead of $\underline{a}$.

1.1.1.2. <u>REMARK</u>. Note the absorption law for + and note that there is no left distributive law $z(x+y) = zx+zy$. Also there is no '0' satisfying $x+0 = x$, $0x = x0 = x$, since this would lead to

$$xy = (x+0)y = xy+0y = xy+y,$$

contrary to our intentions (to have the 'isomorphism' described in Section 4). (However, see Section 3.)

1.1.2. <u>DEFINITION</u>. The operator $\|$ (*merge*) is defined by

$$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x.$$

TU/e technische universiteit eindhoven

Fixed point semantics in process algebra

- Published in 10 years of concurrency semantics, 1992
- Process algebra in strict sense
- ACP in 1983

TU/e technische universiteit eindhoven

Other process algebras

- MEIJE, Austry & Boudol
- LOTOS, Brinksma
- ATP, Hennessy
- Trace theory, Rem

TU/e technische universiteit eindhoven

The variety of process algebra

Baeten, Bergstra, Hoare, Milner, Parrow & De Simone

1991

Uniform notation is desirable?

TU/e technische universiteit eindhoven

Developments

Tooling
Decidability - complexity - expressivity
Verification techniques
Data
Time
Mobility
Probabilities - stochastics

TU/e technische universiteit eindhoven

What is left to do?

Verification techniques - upscaling

Equational reasoning together with model checking and theorem proving

Extended tooling

TU/e technische universiteit eindhoven

Timing

Integrated theory

Applications - upscaling

Abstraction of actions vs. abstraction of timing

Approximation

TU/e technische universiteit eindhoven

Mobility

Unravel concepts

Integrated theory - notion of equality

Practical use

TU/e technische universiteit eindhoven

Probabilities - stochastics

Integrated theory

Applications

Abstraction

Approximation

Performance analysis

TU/e technische universiteit eindhoven

Hybrid process algebra

Just at beginning

Connection with dynamic control

TU/e technische universiteit eindhoven

Theory

Relationship with other concurrency models

Comparison of process algebras

Axiomatizations that are complete, ω-complete

Decidability - complexity

TU/e technische universiteit eindhoven

Conclusion

There is enough to do

Process algebra is alive and kicking

Theory and applications

On the Usability of Process Algebra: An Architectural View

Marco Bernardo

University of Urbino - Italy

Process Algebra

- Semantics of concurrent programs.
- Formal description technique for modeling and verifying computer, communication and software systems.
- Specifications consisting of a sequence of possibly recursive defining equations of the form
  $\texttt{A}(\mathit{formal\_par\_list};\ \mathit{local\_var\_list}) = \texttt{E}$
- Algebraic operators:

$$
\begin{array}{rl}
\texttt{E} ::= & \texttt{stop} \\
\mid & \texttt{A}(\mathit{actual\_par\_list}) \\
\mid & \texttt{a.E} \\
\mid & \texttt{a?}(\mathit{var\_list})\texttt{.E} \\
\mid & \texttt{a!}(\mathit{expr\_list})\texttt{.E} \\
\mid & \texttt{E} + \texttt{E} \\
\mid & \texttt{E/L} \\
\mid & \texttt{E}\backslash\texttt{H} \\
\mid & \texttt{E}[\varphi] \\
\mid & \texttt{E} \parallel_{\texttt{S}} \texttt{E}
\end{array}
$$

Strengths and Weaknesses

+ System modeling is **compositional** thanks to a small number of constructs for building larger descriptions up from smaller ones.

+ Structural operational **semantics** that precisely defines for each process term the state transition graph that it stands for.

+ Syntax-oriented and semantics-oriented behavioral reasoning via **equivalences** that capture the notion of same behavior, possibly abstracting from unnecessary details.

+ Deal with **multiple aspects** like mobility, performability, real time, and security.

– *They are* **difficult** *to learn and use by practitioners.*

– *Their* **technicalities** — *the synchronization discipline — often obfuscate the way in which the systems are modeled.*

An Architectural View

- Need to support a friendly **component-oriented** way of modeling systems within PA −−> the designer can reason in terms of components and their interactions while abstracting from PA technicalities.
- Need to **integrate** the use of PA in the right phase of the system development cycle. (Requirement analysis? Architectural design? Component design? Implementation? Deployment? Testing? Maintainance?).
- **Architectural design level**: A **precise** document, used in all the subsequent phases of the system development, must be defined to describe the **structure** of the system as well as its **behavior** at a high level of abstraction.
- Analyzing the **system properties** at this level is beneficial for the whole system development cycle in terms of time and money.
- Both goals achievable using a revised PA.

Guidelines

- Separation of concerns between *behavior* and *topology*.
- Typing activities through explicit qualifiers:
  - *internal activities* vs. *interactions*;
  - *input interactions* vs. *output interactions*;
  - *architectural interactions* vs. *local interactions*.
- Classifying communications:
  - *1-1*;
  - *conjunctive 1-m*;
  - *disjunctive 1-m*.
- Dealing with *parametricity*.
- Supporting *hierarchy*.

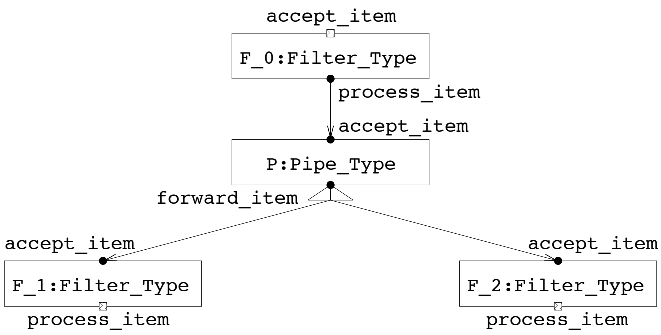
PADL: A PA-Based ADL

- Define each architectural element type (AET) by specifying its parameters, its behavior, the qualifiers of its activities, and the forms of communication in which they can be involved.
- The behavior of an AET is expressed through a list of sequential PA defining equations, with the occurring actions representing the activities of the AET.
- Declare the instances of each architectural element type (AEI) that form the system topology.
- Establish which activities of the AEIs are architectural interactions.
- Attach interactions of different AEIs to make the AEIs interact according to the system topology.

Textual Notation

```
ARCHI_TYPE Pipe_Filter_Type(void)
  ARCHI_ELEM_TYPES
    ELEM_TYPE Filter_Type(void)
      BEHAVIOR Filter_0(void; void) = choice {
                                        accept_item . Filter_1(),
                                        fail . repair . Filter_0()
                                      };
               Filter_1(void; void) = choice {
                                        accept_item . Filter_2(),
                                        process_item . Filter_0(),
                                        fail . repair . Filter_1()
                                      };
               Filter_2(void; void) = choice {
                                        process_item . Filter_1(),
                                        fail . repair . Filter_2()
                                      }
      INPUT_INTERACTIONS UNI accept_item
      OUTPUT_INTERACTIONS UNI process_item
    ELEM_TYPE Pipe_Type(void)
      BEHAVIOR Pipe(void; void) = accept_item . forward_item . Pipe()
      INPUT_INTERACTIONS UNI accept_item
      OUTPUT_INTERACTIONS OR forward_item
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  ARCHI_TOPOLOGY
    ARCHI_ELEM_INSTANCES
      F_0, F_1, F_2 : Filter_Type();
      P             : Pipe_Type()
    ARCHI_INTERACTIONS
      F_0.accept_item;
      F_1.process_item;
      F_2.process_item
    ARCHI_ATTACHMENTS
      FROM F_0.process_item TO P.accept_item;
      FROM P.forward_item   TO F_1.accept_item;
      FROM P.forward_item   TO F_2.accept_item
END
```

Graphical Notation



Translation Semantics into PA

- *First step:* the semantics of each AEI is the behavior of the corresponding AET, where:
  - every action that is not an interaction is made unobservable;
  - every or-interaction is turned into a choice among as many fresh uni-interactions as there are attachments involving the or-interaction.
- Example:

$$
\begin{aligned}
[\![\mathtt{F\_0}]\!] &= \mathtt{Filter\_0}/\{\mathtt{fail},\mathtt{repair}\}\\
[\![\mathtt{F\_1}]\!] &= \mathtt{Filter\_0}/\{\mathtt{fail},\mathtt{repair}\}\\
[\![\mathtt{F\_2}]\!] &= \mathtt{Filter\_0}/\{\mathtt{fail},\mathtt{repair}\}\\
[\![\mathtt{P}]\!] &= \textit{or-rewrite}(\mathtt{Pipe})
\end{aligned}
$$

  where *or-rewrite*(`Pipe`) is given by:

```
Pipe'(void; void) = accept_item . choice
                                  {
                                    forward_item_1 . Pipe'(),
                                    forward_item_2 . Pipe'()
                                  }
```

- *Second step:* the semantics of the whole system description is the parallel composition of the semantics of its AEIs according to the specified attachments.
- Since the parallel composition operator allows only actions with the same name to synchronize, attached interactions need to be relabeled to the same fresh action.
- Example:

$$
\begin{aligned}
[\![\mathtt{Pipe\_Filter\_Type(void)}]\!] = \; & [\![\mathtt{F\_0}]\!][\mathtt{process\_item} \mapsto \mathtt{a}] \,\|_{\{\mathtt{a}\}}\\
& [\![\mathtt{P}]\!][\mathtt{accept\_item} \mapsto \mathtt{a},\\
& \quad \mathtt{forward\_item\_1} \mapsto \mathtt{a\_1},\\
& \quad \mathtt{forward\_item\_2} \mapsto \mathtt{a\_2}] \,\|_{\{\mathtt{a\_1}\}}\\
& \quad [\![\mathtt{F\_1}]\!][\mathtt{accept\_item} \mapsto \mathtt{a\_1}] \,\|_{\{\mathtt{a\_2}\}}\\
& \quad\quad [\![\mathtt{F\_2}]\!][\mathtt{accept\_item} \mapsto \mathtt{a\_2}]
\end{aligned}
$$

Modeling Families of Systems

- An *architectural style* defines a vocabulary of components and connectors and a set of constraints on how they should be combined.
- Architectural styles developed over the years as designers recognized the value of specific organizational principles and structures for certain classes of systems:
  - call-and-return systems (main program and subroutines, object-oriented, client-server, hierarchical layers);
  - dataflow systems (pipe-filter);
  - independent components (event systems);
  - virtual machines (interpreters);
  - repositories (databases, hypertexts).
- Should enable the designers to specify, analyze, plan, and monitor the construction of systems with high levels of efficiency and confidence.

Architectural Types

- The formal description of an architectural style is useful to analyze the properties common to all of its instances.
- Difficult because of at least two degrees of freedom:
  - Variability of the component/connector internal behavior.
  - Variability of the system topology.
- An *architectural type* is an intermediate notion allowing the component/connector internal behavior and the system topology to vary in a controlled way.

- Generation of the instances of an architectural type by invoking its definition.

- Parameter passing:

  – Actual AETs preserving the observable behavior of the formal AETs according to some notion of behavioral equivalence (must be weak and compositional).

  – Actual topology complying with the formal topology according to some allowed extension (exogenous, endogenous, and/or):

    * Actual AEIs.

    * Actual architectural interactions.

    * Actual attachments.

- Hierarchical modeling: the behavior of an AET can be an invocation to a previously defined architectural type.

Behavioral Conformity

- Every actual AEI must be weakly bisimulation equivalent to the corresponding formal AEI (up to injective relabelings unifying their corresponding interactions).

- This guarantees the architectural type invocation to be weakly bisimulation equivalent to the architectural type definition (up to injective relabelings unifying their corresponding interactions).

- The complexity of the check is linear in the number of AETs (instead of being exponential in the number of AEIs).

- Example: a pipe-filter system with perfect filters:

```
Pipe_Filter_Type(;
                 Perfect_Filter_Type, Pipe_Type;
                 F_0, F_1, F_2 : Perfect_Filter_Type(),
                   P : Pipe_Type();
                 F_0.accept_item,
                   F_1.process_item, F_2.process_item;
                 FROM F_0.process_item TO P.accept_item,
                   FROM P.forward_item TO F_1.accept_item,
                   FROM P.forward_item TO F_2.accept_item;
                 )
```

  without actual values for the variables and actual names for the architectural interactions.

- Definition of the new actual AET:

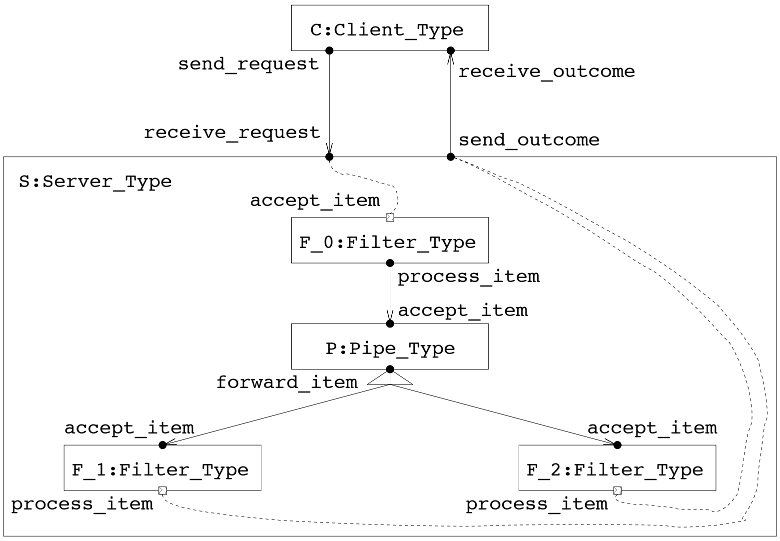
```
ELEM_TYPE Perfect_Filter_Type(void)
  BEHAVIOR P_Filter_0(void; void) = accept_item . P_Filter_1();
           P_Filter_1(void; void) = choice {
                                      accept_item . P_Filter_2(),
                                      process_item . P_Filter_0()
                                    };
           P_Filter_2(void; void) = process_item . P_Filter_1()
  INPUT_INTERACTIONS UNI accept_item
  OUTPUT_INTERACTIONS UNI process_item
```

- Behaviorally conforms to the corresponding formal AET −−> the invocation is a legal instance of the definition of the architectural type.

Hierarchical Modeling

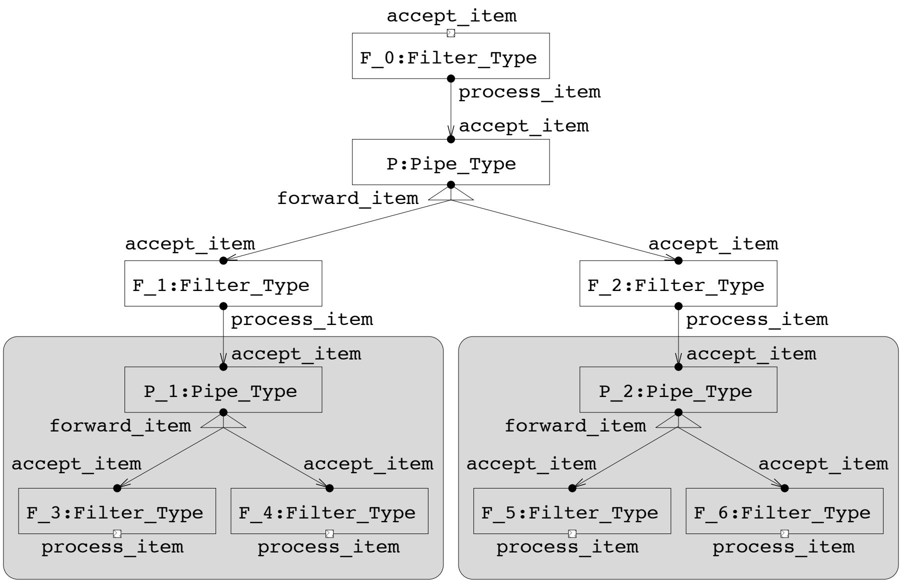
```
ARCHI_TYPE Client_Server_Type(void)
  ARCHI_ELEM_TYPES
    ELEM_TYPE Client_Type(void)
      BEHAVIOR Client(void; void) =
                 send_request . receive_outcome . Client()
      INPUT_INTERACTIONS UNI receive_outcome
      OUTPUT_INTERACTIONS UNI send_request
    ELEM_TYPE Server_Type(void)
      BEHAVIOR Server(void; void) =
>                Pipe_Filter_Type(;
>                                 Filter_Type, Pipe_Type;
>                                 F_0, F_1, F_2 : Filter_Type(),
>                                   P : Pipe_Type();
>                                 F_0.accept_item,
>                                   F_1.process_item, F_2.process_item;
>                                 FROM F_0.process_item TO P.accept_item,
>                                   FROM P.forward_item TO F_1.accept_item,
>                                   FROM P.forward_item TO F_2.accept_item;
>                                 receive_request,
>                                   send_outcome, send_outcome)
      INPUT_INTERACTIONS UNI receive_request
      OUTPUT_INTERACTIONS UNI send_outcome
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  ARCHI_TOPOLOGY
    ARCHI_ELEM_INSTANCES
      C : Client_Type();
      S : Server_Type()
    ARCHI_INTERACTIONS
    ARCHI_ATTACHMENTS
      FROM C.send_request TO S.receive_request;
      FROM S.send_outcome TO C.receive_outcome
END
```

Exogenous Topological Extensions

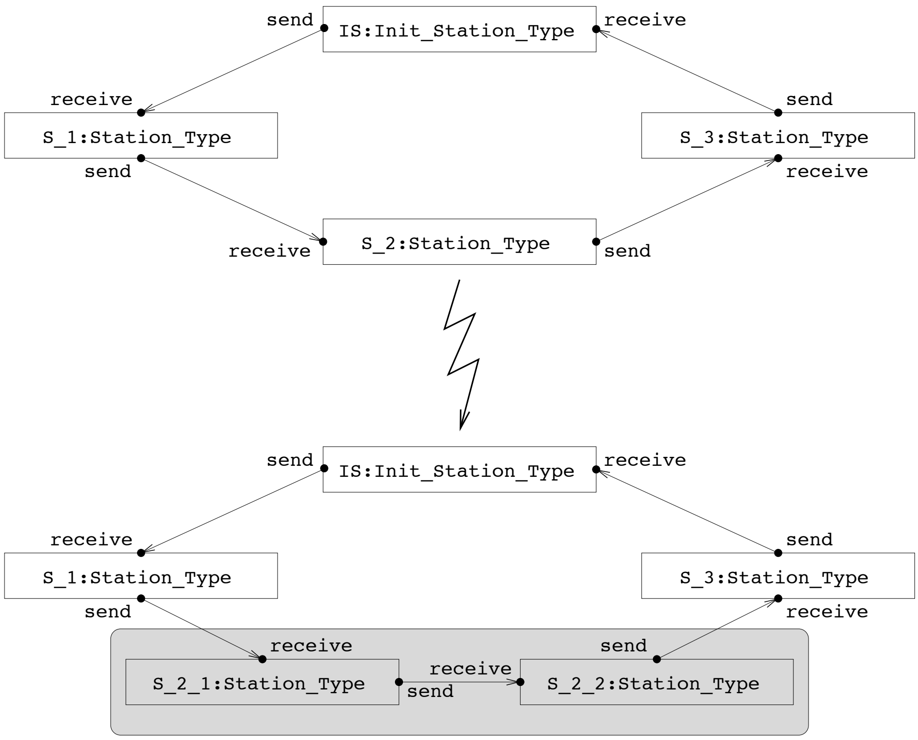


- The topological extension occurs at some of the AEIs forming the border of the topology of the architectural type.
- An AEI is part of the border of the topology if it has an architectural interaction.
- Every addendum must comply with the original topology.

Endogenous Topological Extensions



- The topological extension occurs within the topology of the architectural type.
- Some of the AEIs of the topology are replaced with other AEIs.
- Every replacement must comply with the original topology.



Comparison with PA

- Separation of concerns between behavior and topology, instead of encoding both of them through the parallel composition operator and the related synchronization sets.

- The intended use of every action is made clear through its explicit qualifiers, instead of having to be inferred from the synchronization sets.

- Error-prone situations can easily be detected (e.g.: no attachment between two output or two input interactions, no attachment between two interactions of the same AEI, no multiple attachments involving the same uni-interaction, no internal action involved in an attachment, no isolated groups of AEIs).

- Only the simpler operators (action prefix and choice) can be used.

- Higher degree of specification reuse (both at the component and at the system level).

Comparison with Related Work

- The textual notation and the translation semantics are inspired by Wright:

  - No distinction between components and connectors to avoid trivial connectors.

  - Exploiting the hiding operator instead of specifying ports and roles.

  - Qualifiers of activities and communications.

  - Support for architectural types.

- The graphical notation is inspired by flow graphs, suitably extended to represent the qualifiers of activities and communications.

Component-Oriented Analysis

- For modeling purposes PADL is easier to use than PA. What about analysis?
- All the analysis techniques developed for PA can be reused for PADL. Is it enough?
- Architectural mismatch detection:
  - Verify properties **compositionally**, i.e. infer the properties of the whole system from the properties of its components.
  - Provide component-oriented **diagnostic information** in case of violation.

Performance of Architectural Designs

- The designer may need to **choose** among several **alternative architectures** for the system, with the choice being driven especially by **performance considerations**.
- For a specific architecture of the system, the designer may want to understand whether its **performance can be improved** and, if so, it would be desirable for the designer to have some **diagnostic information** that guide the modification of the architecture itself.
- Need for a **practical methodology** that allows for a **quick prediction, improvement, and comparison** of the performance of different architectures for the system under construction.

Modeling with Æmilia

- Æmilia is an extension of PADL based on the stochastic process algebra EMPA_{gr}.
- Action extensions:
  - `<a, λ>` has an exponentially distributed duration of rate λ (race policy);
  - `<a, inf(l, w)>` has zero duration, priority level `l`, and weight `w` (generative preselection policy);
  - `<a, *(l, w)>` has unspecified duration, priority level `l`, and weight `w` (reactive preselection policy).
- Exponentially timed and immediate actions can synchronize only with passive actions −−> every set of attached interactions can contain at most one nonpassive interaction.
- Rates, weights, and priorities can be parameters of architectural types and AETs.
- Performance evaluation via Markov chains.

```
ARCHI_TYPE Pipe_Filter_Type(void;
                            rate μ0, μ1, μ2, φ0, φ1, φ2, ρ0, ρ1, ρ2,
                            weight p_routing)
  ARCHI_ELEM_TYPES
    ELEM_TYPE Filter_Type(void; rate μ, φ, ρ)
      BEHAVIOR Filter_0(void; void) =
                 choice { <accept_item, *> . Filter_1(),
                          <fail, φ> . <repair, ρ> . Filter_0() };
               Filter_1(void; void) =
                 choice { <accept_item, *> . Filter_2(),
                          <process_item, μ> . Filter_0(),
                          <fail, φ> . <repair, ρ> . Filter_1() };
               Filter_2(void; void) =
                 choice { <process_item, μ> . Filter_1(),
                          <fail, φ> . <repair, ρ> . Filter_2() }
      INPUT_INTERACTIONS UNI accept_item
      OUTPUT_INTERACTIONS UNI process_item
    ELEM_TYPE Pipe_Type(void; weight p)
      BEHAVIOR Pipe(void; void) =
                 <accept_item, *> .
                   choice { <forward_item_1, inf(1, p)> . Pipe(),
                            <forward_item_2, inf(1, 1 - p)> . Pipe() }
      INPUT_INTERACTIONS UNI accept_item
      OUTPUT_INTERACTIONS UNI forward_item_1; forward_item_2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  ARCHI_TOPOLOGY
    ARCHI_ELEM_INSTANCES
      F_0 : Filter_Type(; μ0, φ0, ρ0);
      F_1 : Filter_Type(; μ1, φ1, ρ1);
      F_2 : Filter_Type(; μ2, φ2, ρ2);
      P   : Pipe_Type(; p_routing)
    ARCHI_INTERACTIONS
      F_0.accept_item; F_1.process_item; F_2.process_item
    ARCHI_ATTACHMENTS
      FROM F_0.process_item TO P.accept_item;
      FROM P.forward_item   TO F_1.accept_item;
      FROM P.forward_item   TO F_2.accept_item
END
```

Analysis with Queueing Networks

- Markov chains are state-based performance models: not suited for the architectural design level.
- QNs are ***structured*** performance models:
  - The system components are elucidated.
  - Computation of typical average performance indices both at the system level and at the component level.
  - Fast solution algorithms for some classes of QNs.
  - Symbolic analysis possible in some cases.

Combining Æmilia and QNs

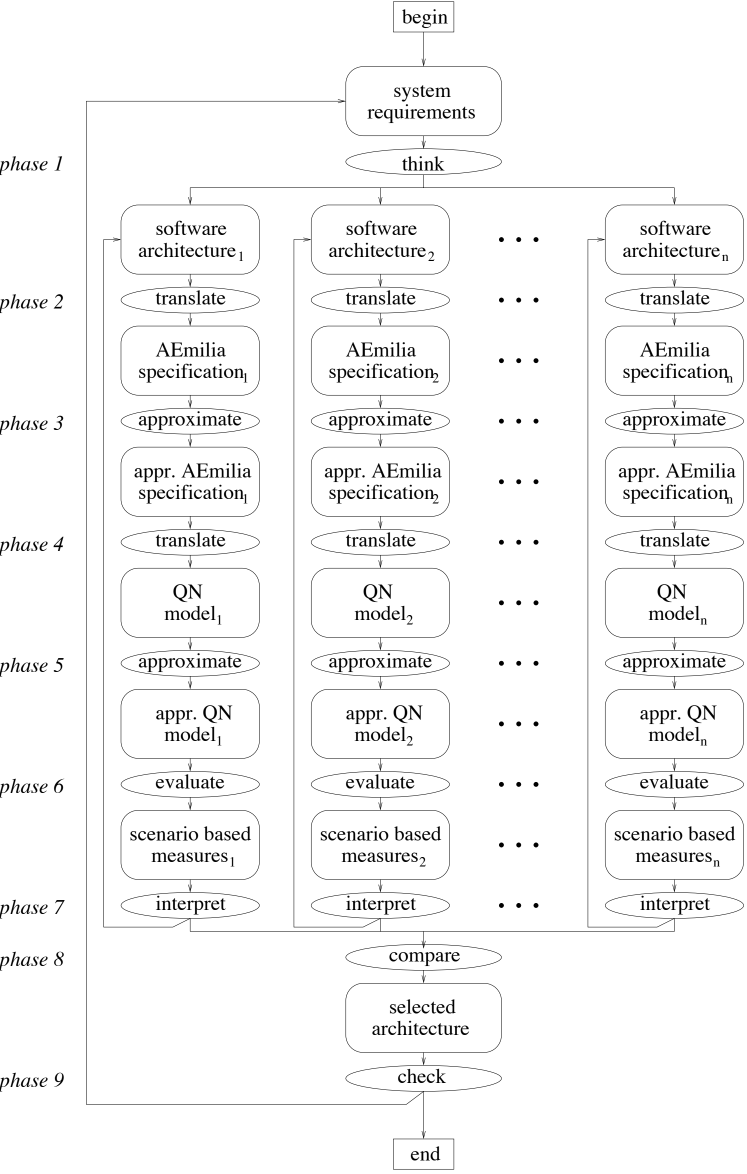
- Action-based, component-oriented, general-purpose formal specification language
  vs.
  queue-oriented graphical notation for performance modeling only, with some details expressed in natural language.
- Only the Æmilia specifications of a reasonably wide class can be translated into QN models.
- The AEIs of an Æmilia specification cannot be mapped to QN service centers, but to finer parts called *QN basic elements*: arrival processes, buffers, fork processes, join processes, and service processes.
- Use *syntax restrictions* to make sure that all the AEIs of an Æmilia specification can be translated into QN basic elements.
- Complexity of the translation linear in the number of AEIs of the Æmilia specification.

Methodology

- Objective: quick prediction, improvement, and comparison of the performance of different architectural designs.
- How to use in practice the combination of Æmilia and QNs?
- Multi-phase methodology with feedback.
- Variable number of alternative designs.
- Approximations.
- Focus on four specific, average performance indices providing insights for the achievement of general performance requirements.
- Computed at the component level and at the system level.

Average Performance Indices

- *Throughput*: measure of the productivity of the components; singles out the components that are bottlenecks.
- *Utilization*: measure of the relative usage of computational resources by the components; provides information useful at deployment time.
- *Mean queue length*: measure of the average size of data repositories; avoids component execution blocking (under-sized buffers) and waste of memory (over-sized buffers).
- *Mean response time*: measure of the average running time of the components; predicts the QoS perceived by the users.



Comparing Compiler Architectures

- Five phases: lexical analysis, parsing, type checking, code optimization, and code generation.

- Two classes of programs (optimization).

- Different architectures: sequential, pipeline, concurrent.

- Comparing them in some scenarios of interest, based on:
  - mean number of programs compiled per unit of time;
  - average fraction of time during which the compiler is being used;
  - mean number of programs in the compiler system;
  - mean compilation time.

- Application of the methodology.

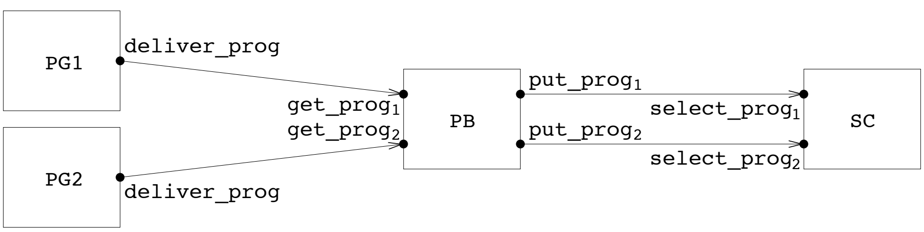
Sequential Compiler

- Only one program at a time can be compiled.

- Each of the five phases introduces an exponentially distributed delay: $\mu_{\mathtt{l}}, \mu_{\mathtt{p}}, \mu_{\mathtt{c}}, \mu_{\mathtt{o}}, \mu_{\mathtt{g}}$.

- The arrival processes of the two classes of programs are Poisson processes with rates λ_1 and λ_2.

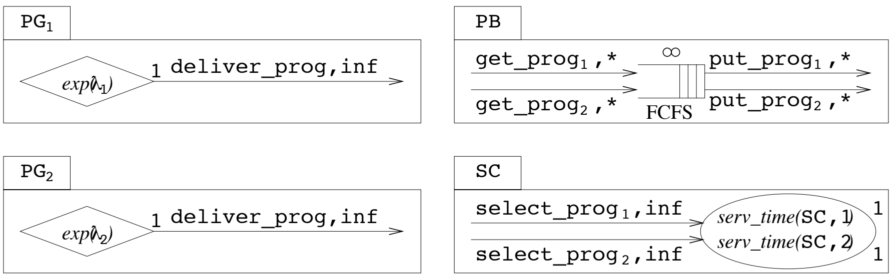
- Compiler system comprising two program generators and a buffer.

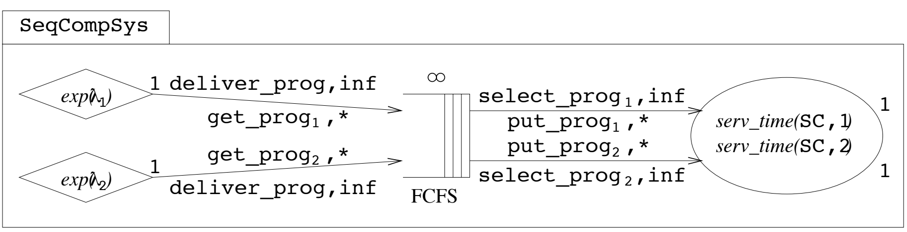


```
ARCHI_TYPE               SeqCompSys(void; rate λ_1, λ_2, μ_1, μ_p, μ_c, μ_o, μ_g)
 ARCHI_ELEM_TYPES
  ELEM_TYPE              ProgGenT(void; rate λ)
   BEHAVIOR              ProgGen(void; void) =
                           <generate_prog, λ>.<deliver_prog, inf>.ProgGen()
   INPUT_INTERACTIONS
   OUTPUT_INTERACTIONS   UNI deliver_prog

  ELEM_TYPE              ProgBufferT(integer h_1, h_2; void)
   BEHAVIOR              ProgBuffer(integer h_1, h_2; void) =
                           choice
                           {
                             <get_prog_1, *>.ProgBuffer(h_1 + 1, h_2),
                             <get_prog_2, *>.ProgBuffer(h_1, h_2 + 1),
                             cond(h_1 > 0) -> <put_prog_1, *>.ProgBuffer(h_1 - 1, h_2),
                             cond(h_2 > 0) -> <put_prog_2, *>.ProgBuffer(h_1, h_2 - 1)
                           }
   INPUT_INTERACTIONS    UNI get_prog_1; get_prog_2
   OUTPUT_INTERACTIONS   UNI put_prog_1; put_prog_2

  ELEM_TYPE              SeqCompT(void; rate μ_1, μ_p, μ_c, μ_o, μ_g)
   BEHAVIOR              SeqComp(void; void) =
                           choice
                           {
                             <select_prog_1, inf>.<recognize_tokens, μ_1>.
                               <parse_phrases, μ_p>.<check_phrases, μ_c>.
                               <optimize_code, μ_o>.<generate_code, μ_g>.SeqComp(),
                             <select_prog_2, inf>.<recognize_tokens, μ_1>.
                               <parse_phrases, μ_p>.<check_phrases, μ_c>.
                               <generate_code, μ_g>.SeqComp()
                           }
   INPUT_INTERACTIONS    UNI select_prog_1; select_prog_2
   OUTPUT_INTERACTIONS
 ARCHI_TOPOLOGY
  ARCHI_ELEM_INSTANCES   PG_1 : ProgGenT(; λ_1);
                         PG_2 : ProgGenT(; λ_2);
                         PB : ProgBufferT(0, 0; );
                         SC : SeqCompT(; μ_1, μ_p, μ_c, μ_o, μ_g)
  ARCHI_INTERACTIONS
  ARCHI_ATTACHMENTS      FROM PG_1.deliver_prog TO PB.get_prog_1;
                         FROM PG_2.deliver_prog TO PB.get_prog_2;
                         FROM PB.put_prog_1 TO SC.select_prog_1;
                         FROM PB.put_prog_2 TO SC.select_prog_2
END
```





- Scenario-specific parameters: $\lambda_{\texttt{seq},1}, \lambda_{\texttt{seq},2}, \mu_{\texttt{seq},1}, \mu_{\texttt{seq,p}}, \mu_{\texttt{seq,c}}, \mu_{\texttt{seq,o}}, \mu_{\texttt{seq,g}}$.

- Approximations to get a QS M/M/1:
  - Single arrival process with rate $\lambda_{\texttt{seq}} = \lambda_{\texttt{seq},1} + \lambda_{\texttt{seq},2}$ (probabilities $\lambda_{\texttt{seq},1}/\lambda_{\texttt{seq}}$ and $\lambda_{\texttt{seq},2}/\lambda_{\texttt{seq}}$).
  - Exponential service time for the first class with rate $\mu_{\texttt{seq},1}$ such that $\mu_{\texttt{seq},1}^{-1} = \mu_{\texttt{seq},1}^{-1} + \mu_{\texttt{seq,p}}^{-1} + \mu_{\texttt{seq,c}}^{-1} + \mu_{\texttt{seq,o}}^{-1} + \mu_{\texttt{seq,g}}^{-1}$.
  - Exponential service time for the second class with rate $\mu_{\texttt{seq},2}$ such that $\mu_{\texttt{seq},2}^{-1} = \mu_{\texttt{seq},1}^{-1} + \mu_{\texttt{seq,p}}^{-1} + \mu_{\texttt{seq,c}}^{-1} + \mu_{\texttt{seq,g}}^{-1}$.
  - Single class of programs with rate $\mu_{\texttt{seq}}$ such that $\mu_{\texttt{seq}}^{-1} = (\lambda_{\texttt{seq},1}/\lambda_{\texttt{seq}}) \cdot \mu_{\texttt{seq},1}^{-1} + (\lambda_{\texttt{seq},2}/\lambda_{\texttt{seq}}) \cdot \mu_{\texttt{seq},2}^{-1}$.

- Stability: $\rho_{\texttt{seq}} = \lambda_{\texttt{seq}}/\mu_{\texttt{seq}} < 1$.

- Sequential compiler throughput:
$$\overline{X}_{\texttt{seq}} = \lambda_{\texttt{seq}}$$

- Sequential compiler utilization:
$$\overline{U}_{\texttt{seq}} = \rho_{\texttt{seq}}$$

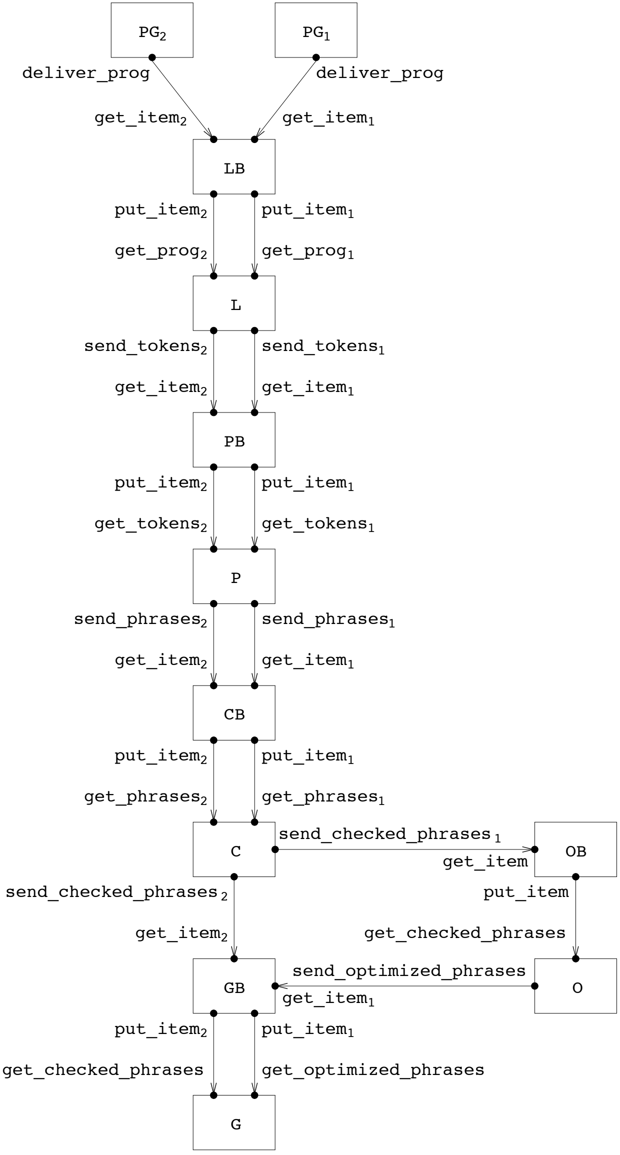
- Mean number of programs in the sequential compiler system:
$$\overline{N}_{\texttt{seq}} = \rho_{\texttt{seq}}/(1 - \rho_{\texttt{seq}})$$

- Mean sequential compilation time:
$$\overline{R}_{\texttt{seq}} = 1/[\mu_{\texttt{seq}} \cdot (1 - \rho_{\texttt{seq}})]$$

Pipeline Compiler

- Simultaneous compilation of several programs at different stages.

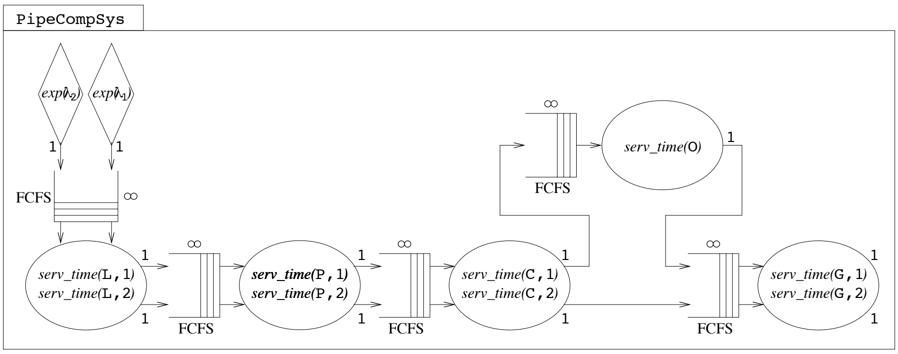
- The five phases are carried out by five distinct components, each having its own buffer, operating in parallel on different programs.




```
ARCHI_TYPE              PipeCompSys(void; rate λ_1, λ_2, μ_l, μ_p, μ_c, μ_o, μ_g)
 ARCHI_ELEM_TYPES
  ELEM_TYPE             ProgGenT(void; rate λ)
   BEHAVIOR             ProgGen(void; void) =
                          <generate_prog, λ>.<deliver_prog, inf>.ProgGen()
   INPUT_INTERACTIONS
   OUTPUT_INTERACTIONS  UNI deliver_prog
  ELEM_TYPE             OneClassBufferT(integer h; void)
   BEHAVIOR             OneClassBuffer(integer h; void) =
                          choice
                          {
                            <get_item, *>.OneClassBuffer(h + 1),
                            cond(h > 0) -> <put_item, *>.
                                             OneClassBuffer(h − 1)
                          }
   INPUT_INTERACTIONS   UNI get_item
   OUTPUT_INTERACTIONS  UNI put_item
  ELEM_TYPE             TwoClassesBufferT(integer h_1, h_2; void)
   BEHAVIOR             TwoClassesBuffer(integer h_1, h_2; void) =
                          choice
                          {
                            <get_item_1, *>.TwoClassesBuffer(h_1 + 1, h_2),
                            <get_item_2, *>.TwoClassesBuffer(h_1, h_2 + 1),
                            cond(h_1 > 0) -> <put_item_1, *>.
                                              TwoClassesBuffer(h_1 − 1, h_2),
                            cond(h_2 > 0) -> <put_item_2, *>.
                                              TwoClassesBuffer(h_1, h_2 − 1)
                          }
   INPUT_INTERACTIONS   UNI get_item_1; get_item_2
   OUTPUT_INTERACTIONS  UNI put_item_1; put_item_2
  ELEM_TYPE             LexerT(void; rate μ_l)
   BEHAVIOR             Lexer(void; void) =
                          choice
                          {
                            <get_prog_1, inf>.<recognize_tokens, μ_l>.
                              <send_tokens_1, inf>.Lexer(),
                            <get_prog_2, inf>.<recognize_tokens, μ_l>.
                              <send_tokens_2, inf>.Lexer()
                          }
   INPUT_INTERACTIONS   UNI select_prog_1; select_prog_2
   OUTPUT_INTERACTIONS  UNI send_tokens_1; send_tokens_2
```

```
ELEM_TYPE                ParserT(void; rate μ_p)
 BEHAVIOR                Parser(void; void) =
                          choice
                          {
                           <get_tokens_1, inf>.<parse_phrases, μ_p>.
                             <send_phrases_1, inf>.Parser(),
                           <get_tokens_2, inf>.<parse_phrases, μ_p>.
                             <send_phrases_2, inf>.Parser()
                          }
 INPUT_INTERACTIONS      UNI get_tokens_1; get_tokens_2
 OUTPUT_INTERACTIONS     UNI send_phrases_1; send_phrases_2
ELEM_TYPE                CheckerT(void; rate μ_c)
 BEHAVIOR                Checker(void; void) =
                          choice
                          {
                           <get_phrases_1, inf>.<check_phrases, μ_c>.
                             <send_checked_phrases_1, inf>.Checker(),
                           <get_phrases_2, inf>.<check_phrases, μ_c>.
                             <send_checked_phrases_2, inf>.Checker()
                          }
 INPUT_INTERACTIONS      UNI get_phrases_1; get_phrases_2
 OUTPUT_INTERACTIONS     UNI send_checked_phrases_1; send_checked_phrases_2
ELEM_TYPE                OptimizerT(void; rate μ_o)
 BEHAVIOR                Optimizer(void; void) =
                           <get_checked_phrases, inf>.
                             <optimize_phrases, μ_o>.
                             <send_optimized_phrases, inf>.Optimizer()
 INPUT_INTERACTIONS      UNI get_checked_phrases
 OUTPUT_INTERACTIONS     UNI send_optimized_phrases
ELEM_TYPE                GeneratorT(void; rate μ_g)
 BEHAVIOR                Generator(void; void) =
                          choice
                          {
                           <get_optimized_phrases, inf>.
                             <generate_code, μ_g>.Generator(),
                           <get_checked_phrases, inf>.
                             <generate_code, μ_g>.Generator()
                          }
 INPUT_INTERACTIONS      UNI get_optimized_phrases; get_checked_phrases
 OUTPUT_INTERACTIONS
```

```
ARCHI_TOPOLOGY
 ARCHI_ELEM_INSTANCES   PG_1 : ProgGenT(; λ_1);
                        PG_2 : ProgGenT(; λ_2);
                        LB : TwoClassesBufferT(0, 0; );
                        L : LexerT(; μ_l);
                        PB : TwoClassesBufferT(0, 0; );
                        P : ParserT(; μ_p);
                        CB : TwoClassesBufferT(0, 0; );
                        C : CheckerT(; μ_c);
                        OB : OneClassBufferT(0; );
                        O : OptimizerT(; μ_o);
                        GB : TwoClassesBufferT(0, 0; );
                        G : GeneratorT(; μ_g);
 ARCHI_INTERACTIONS
 ARCHI_ATTACHMENTS      FROM PG_1.deliver_prog TO LB.get_item_1;
                        FROM PG_2.deliver_prog TO LB.get_item_2;
                        FROM LB.put_item_1 TO L.get_prog_1;
                        FROM LB.put_item_2 TO L.get_prog_2;
                        FROM L.send_tokens_1 TO PB.get_item_1;
                        FROM L.send_tokens_2 TO PB.get_item_2;
                        FROM PB.put_item_1 TO P.get_tokens_1;
                        FROM PB.put_item_2 TO P.get_tokens_2;
                        FROM P.send_phrases_1 TO CB.get_item_1;
                        FROM P.send_phrases_2 TO CB.get_item_2;
                        FROM CB.put_item_1 TO C.get_phrases_1;
                        FROM CB.put_item_2 TO C.get_phrases_2;
                        FROM C.send_checked_phrases_1 TO OB.get_item;
                        FROM C.send_checked_phrases_2 TO GB.get_item_2;
                        FROM OB.put_item TO O.get_checked_phrases;
                        FROM O.send_optimized_phrases TO GB.get_item_1;
                        FROM GB.put_item_1 TO G.get_optimized_phrases;
                        FROM GB.put_item_2 TO G.get_checked_phrases
END
```



- Scenario-specific parameters: $\lambda_{\mathtt{pipe},1}$, $\lambda_{\mathtt{pipe},2}$, $\mu_{\mathtt{pipe},\mathtt{l}}$, $\mu_{\mathtt{pipe},\mathtt{p}}$, $\mu_{\mathtt{pipe},\mathtt{c}}$, $\mu_{\mathtt{pipe},\mathtt{o}}$, $\mu_{\mathtt{pipe},\mathtt{g}}$.

- Approximation to get a product form open QN composed of five QSs M/M/1: single arrival process with rate $\lambda_{\mathtt{pipe}} = \lambda_{\mathtt{pipe},1} + \lambda_{\mathtt{pipe},2}$.

- At equilibrium the arrival rate for the lexer, the parser, the checker, and the generator is $\lambda_{\mathtt{pipe}}$ while for the optimizer is $\lambda_{\mathtt{pipe},1}$.

- The probability that a program leaving the checker enters the optimizer (resp. the generator) is $\lambda_{\mathtt{pipe},1}/\lambda_{\mathtt{pipe}}$ (resp. $\lambda_{\mathtt{pipe},2}/\lambda_{\mathtt{pipe}}$).

- Stability: $\lambda_{\texttt{pipe}} < \min(\mu_{\texttt{pipe,l}}, \mu_{\texttt{pipe,p}}, \mu_{\texttt{pipe,c}}, \mu_{\texttt{pipe,o}} \cdot (\lambda_{\texttt{pipe}}/\lambda_{\texttt{pipe,1}}), \mu_{\texttt{pipe,g}})$.

- Phase j throughput:
$$\begin{aligned} \overline{X}_{\texttt{pipe},j} &= \lambda_{\texttt{pipe}} \quad \text{for } j \neq \texttt{o} \\ \overline{X}_{\texttt{pipe,o}} &= \lambda_{\texttt{pipe,1}} \end{aligned}$$

- Phase j utilization:
$$\overline{U}_{\texttt{pipe},j} = \rho_{\texttt{pipe},j}$$

- Mean number of programs in phase j:
$$\overline{N}_{\texttt{pipe},j} = \rho_{\texttt{pipe},j}/(1 - \rho_{\texttt{pipe},j})$$

- Mean duration of phase j:
$$\overline{R}_{\texttt{pipe},j} = 1/[\mu_{pipe,j} \cdot (1 - \rho_{\texttt{pipe},j})]$$

- Pipeline compiler throughput:
$$\overline{X}_{\texttt{pipe}} = \overline{X}_{\texttt{pipe,g}}$$

- Pipeline compiler utilization:
$$\overline{U}_{\texttt{pipe}} = 1 - \prod_j (1 - \overline{U}_{\texttt{pipe},j})$$

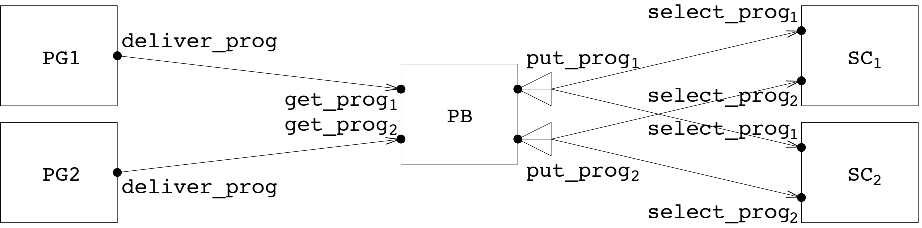
- Mean number of programs in the pipeline compiler system:
$$\overline{N}_{\texttt{pipe}} = \sum_j \overline{N}_{\texttt{pipe},j}$$

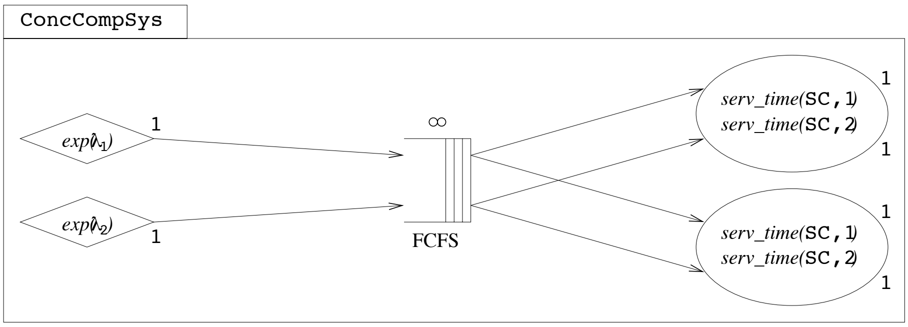
- Mean pipeline compilation time:
$$\overline{R}_{\texttt{pipe}} = \frac{\lambda_{\texttt{pipe,1}}}{\lambda_{\texttt{pipe}}} \cdot \sum_j \overline{R}_{\texttt{pipe},j} + \frac{\lambda_{\texttt{pipe,2}}}{\lambda_{\texttt{pipe}}} \cdot \sum_{j \neq \texttt{o}} \overline{R}_{\texttt{pipe},j}$$

Concurrent Compiler

- Two sequential monolithic compilers operating in parallel on two different programs.

- The programs are taken from a shared buffer.






```
ARCHI_TYPE              ConcCompSys(void; rate λ1, λ2, μ1, μp, μc, μo, μg)
 ARCHI_ELEM_TYPES
  ELEM_TYPE             ProgGenT(void; rate λ)
   BEHAVIOR             ProgGen(void; void) =
                          <generate_prog, λ>.<deliver_prog, inf>.ProgGen()
   INPUT_INTERACTIONS
   OUTPUT_INTERACTIONS  UNI deliver_prog
  ELEM_TYPE             ProgBufferT(integer h1, h2; void)
   BEHAVIOR             ProgBuffer(integer h1, h2; void) =
                          choice
                          {
                            <get_prog1, *>.ProgBuffer(h1 + 1, h2),
                            <get_prog2, *>.ProgBuffer(h1, h2 + 1),
                            cond(h1 > 0) -> <put_prog1, *>.ProgBuffer(h1 − 1, h2),
                            cond(h2 > 0) -> <put_prog2, *>.ProgBuffer(h1, h2 − 1)
                          }
   INPUT_INTERACTIONS   UNI get_prog1; get_prog2
   OUTPUT_INTERACTIONS  OR put_prog1; put_prog2
  ELEM_TYPE             SeqCompT(void; rate μ1, μp, μc, μo, μg)
   BEHAVIOR             SeqComp(void; void) =
                          choice
                          {
                            <select_prog1, inf>.<recognize_tokens, μ1>.
                              <parse_phrases, μp>.<check_phrases, μc>.
                              <optimize_code, μo>.<generate_code, μg>.SeqComp(),
                            <select_prog2, inf>.<recognize_tokens, μ1>.
                              <parse_phrases, μp>.<check_phrases, μc>.
                              <generate_code, μg>.SeqComp()
                          }
   INPUT_INTERACTIONS   UNI select_prog1; select_prog2
   OUTPUT_INTERACTIONS
 ARCHI_TOPOLOGY
  ARCHI_ELEM_INSTANCES  PG1 : ProgGenT(; λ1);
                        PG2 : ProgGenT(; λ2);
                        PB : ProgBufferT(0, 0; );
                        SC1, SC2 : SeqCompT(; μ1, μp, μc, μo, μg)
  ARCHI_INTERACTIONS
  ARCHI_ATTACHMENTS     FROM PG1.deliver_prog TO PB.get_prog1;
                        FROM PG2.deliver_prog TO PB.get_prog2;
                        FROM PB.put_prog1 TO SC1.select_prog1;
                        FROM PB.put_prog1 TO SC2.select_prog1;
                        FROM PB.put_prog2 TO SC1.select_prog2;
                        FROM PB.put_prog2 TO SC2.select_prog2
END
```

- Scenario-specific parameters: $\lambda_{\texttt{conc},1}$, $\lambda_{\texttt{conc},2}$, $\mu_{\texttt{conc},1}$, $\mu_{\texttt{conc},\texttt{p}}$, $\mu_{\texttt{conc},\texttt{c}}$, $\mu_{\texttt{conc},\texttt{o}}$, $\mu_{\texttt{conc},\texttt{g}}$.

- Approximations to get a QS M/M/2 similar to those for the sequential architecture.

- Stability: $\rho_{\texttt{conc}} = \lambda_{\texttt{conc}}/(2 \cdot \mu_{\texttt{conc}}) < 1$.

- Concurrent compiler throughput:
$$\overline{X}_{\texttt{conc}} = \lambda_{\texttt{conc}}$$

- Concurrent compiler utilization:
$$\overline{U}_{\texttt{conc}} = 2 \cdot \rho_{\texttt{conc}}/(1 + \rho_{\texttt{conc}})$$

- Mean number of programs in the concurrent compiler system:
$$\overline{N}_{\texttt{conc}} = 2 \cdot \rho_{\texttt{conc}}/(1 - \rho^2_{\texttt{conc}})$$

- Mean concurrent compilation time:
$$\overline{R}_{\texttt{conc}} = 1/[\mu_{\texttt{conc}} \cdot (1 - \rho^2_{\texttt{conc}})]$$

Scenario-Based Comparison

- Fair comparison: $\mu_{\texttt{seq},j} = \mu_{\texttt{pipe},j} = \mu_{\texttt{conc},j} \equiv \mu_j$ for all $j \in \{\texttt{1}, \texttt{p}, \texttt{c}, \texttt{o}, \texttt{g}\}$.

- Preservation of the frequency of each class of programs: $\lambda_{\texttt{seq},c}/\lambda_{\texttt{seq}} = \lambda_{\texttt{pipe},c}/\lambda_{\texttt{pipe}} = \lambda_{\texttt{conc},c}/\lambda_{\texttt{conc}} \equiv p_c$ for all $c \in \{\texttt{1}, \texttt{2}\}$.

- Throughput: mean number of programs that are compiled per unit of time.

- Two scenarios: light load and heavy load.

- Light load: the specific architecture does not really matter, as the relations among the three throughputs directly depend on the relations among the three arrival rates: $\overline{X}_{t_1} \, \mathcal{R} \, \overline{X}_{t_2}$ if and only if $\lambda_{t_1} \, \mathcal{R} \, \lambda_{t_2}$ for all $t_1, t_2 \in \{\texttt{seq}, \texttt{pipe}, \texttt{conc}\}$ and $\mathcal{R} \in \{<, =, >\}$.

- Heavy load: all the architectures work close to their maximum throughputs, which can be derived from the corresponding stability conditions:
$$\begin{aligned} \lambda_{\texttt{seq}} &\cong \overline{X}_{\texttt{seq,max}} = \mu_{\texttt{seq}} \\ \lambda_{\texttt{pipe}} &\cong \overline{X}_{\texttt{pipe,max}} = \min(\mu_{\texttt{1}}, \mu_{\texttt{p}}, \mu_{\texttt{c}}, \mu_{\texttt{o}}/p_{\texttt{1}}, \mu_{\texttt{g}}) \\ \lambda_{\texttt{conc}} &\cong \overline{X}_{\texttt{conc,max}} = 2 \cdot \mu_{\texttt{conc}} \end{aligned}$$

- Consider three sub-scenarios.

- The five compilation phases have approximatively the same average duration μ^{-1}:
$$\begin{aligned} \overline{X}_{\texttt{seq,max}} &\cong (4 + p_{\texttt{1}})^{-1} \cdot \mu \\ \overline{X}_{\texttt{pipe,max}} &\cong \mu \\ \overline{X}_{\texttt{conc,max}} &\cong 2 \cdot (4 + p_{\texttt{1}})^{-1} \cdot \mu \end{aligned}$$
It follows that:
$$\begin{aligned} \overline{X}_{\texttt{pipe,max}}/\overline{X}_{\texttt{seq,max}} &\cong 4 + p_{\texttt{1}} \\ \overline{X}_{\texttt{pipe,max}}/\overline{X}_{\texttt{conc,max}} &\cong 2 + 0.5 \cdot p_{\texttt{1}} \\ \overline{X}_{\texttt{conc,max}}/\overline{X}_{\texttt{seq,max}} &\cong 2 \end{aligned}$$
The pipeline architecture wins.

- The average duration of a compilation phase is several orders of magnitude greater than the average duration of the other phases:
$$\begin{aligned} \overline{X}_{\texttt{seq,max}} &\cong \mu_{\texttt{1}} \\ \overline{X}_{\texttt{pipe,max}} &= \mu_{\texttt{1}} \\ \overline{X}_{\texttt{conc,max}} &\cong 2 \cdot \mu_{\texttt{1}} \end{aligned}$$
It follows that:
$$\begin{aligned} \overline{X}_{\texttt{pipe,max}}/\overline{X}_{\texttt{seq,max}} &\cong 1 \\ \overline{X}_{\texttt{conc,max}}/\overline{X}_{\texttt{pipe,max}} &\cong 2 \\ \overline{X}_{\texttt{conc,max}}/\overline{X}_{\texttt{seq,max}} &\cong 2 \end{aligned}$$
The concurrent architecture wins.

- The average durations range between a minimum value $\mu_{\texttt{min}}^{-1}$ and a maximum value $\mu_{\texttt{max}}^{-1}$ that are several orders of magnitude apart:
$$\begin{aligned} (4 + p_{\texttt{1}}) \cdot \tfrac{\mu_{\texttt{min}}}{\mu_{\texttt{max}}} \leq \tfrac{\overline{X}_{\texttt{pipe,max}}}{\overline{X}_{\texttt{seq,max}}} &\leq 4 + p_{\texttt{1}} \\ (4 + p_{\texttt{1}}) \cdot \tfrac{\mu_{\texttt{min}}}{\mu_{\texttt{max}}} / 2 \leq \tfrac{\overline{X}_{\texttt{pipe,max}}}{\overline{X}_{\texttt{conc,max}}} &\leq (4 + p_{\texttt{1}}) / 2 \\ 2 \leq \tfrac{\overline{X}_{\texttt{conc,max}}}{\overline{X}_{\texttt{seq,max}}} &\leq 2 \end{aligned}$$
The concurrent architecture is always twice as faster as the sequential one, while the pipeline architecture can perform worse than the other two.

References

[1] M. Bernardo, P. Ciancarini, and L. Donatiello, *"Architecting Families of Software Systems with Process Algebras"*, in ACM Trans. on Software Engineering and Methodology 11:386-426, 2002.

[2] M. Bernardo and F. Franzè, *"Exogenous and Endogenous Extensions of Architectural Types"*, in Proc. of the *5th Int. Conf. on Coordination Models and Languages (COORDINATION 2002)*, LNCS 2315:40-55, York (UK), 2002.

[3] M. Bernardo and F. Franzè, *"Architectural Types Revisited: Extensible And/Or Connections"*, in Proc. of the *5th Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2002)*, LNCS 2306:113-128, Grenoble (France), 2002.

[4] A. Aldini and M. Bernardo, *"A General Deadlock Detection Approach for Software Architectures"*, to appear in Proc. of the *12th Int. Formal Methods Europe Symp. (FME 2003)*, LNCS, Pisa (Italy), 2003.

[5] M. Bernardo, L. Donatiello, and P. Ciancarini, *"Stochastic Process Algebra: From an Algebraic Formalism to an Architectural Description Language"*, in *Performance Evaluation of Complex Systems: Techniques and Tools*, LNCS 2459:236-260, 2002.

[6] S. Balsamo, M. Bernardo, and M. Simeoni, *"Performance Evaluation at the Software Architecture Level"*, in *Formal Methods for Software Architectures*, LNCS 2804:209-260, 2003.

[7] M. Bernardo, *"TwoTowers 3.0: Enhancing Usability"*, to appear in Proc. of the *11th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2003)*, IEEE-CS Press, Orlando (FL), October 2003

Axiomatizing process algebra with time:

real-time and stochastic-time

Mario Bravetti
Università di Bologna

Parts are joint work with: Roberto Gorrieri

1. Basic priority: maximal progress

- Extension of the *standard Milner's sound and complete axiomatization* when a new "δ" prefix is introduced representing a *time delay*.
- If we assume that time may elapse only when no standard action can be performed (*maximal progress assumption*) then such extension is not trivial.
- Tecnically, we assume a simple form of *priority of "τ" actions over "δ" actions*: visible actions are interpreted as representing *potential for execution* only.

1.1 The basic calculus

- Similarly as for standard CCS we start from axiomatizing a *basic calculus with recursion*:

  $P ::= \underline{0} \;|\; \pi.P \;|\; P + P \;|\; RecX.P \;|\; X$

  where "π.*P*" is either "α.*P*" (α is *a* or τ) or "δ.*P*".
- The operational semantics of this calculus is a simple variant of the standard one:

$$\frac{P \xrightarrow{\delta} P' \quad Q \not\xrightarrow{\tau}}{P + Q \xrightarrow{\delta} P'}$$

- model of *"δ.P + τ.Q" is isomorphic to "τ.Q"*.

1.2 Weak bisimulation equivalence

- The notion of equivalence is *just standard Milner's observational bisimulation equivalence* where *"δ" is treated as a visible action*.
- Technically, when we'll consider static operators we'll need *a slightly different treatment of "δ" and visible actions*: in observational congruence after an (intial) "δ" step we do not consider weak bisimulation, but still observational congruence.
- In any case it is a *conservative extension of Milner's observational congruence*.
- It is a *congruence* for our prioritized calculus!

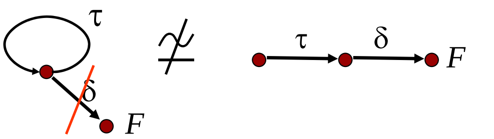
1.3 Problems with standard axiomatization

- We have problems with the *soundness of the axioms for unguarded recursion* (the second one):

  (Ung1) $RecX.(X + E) = RecX.E$

  (Ung2) $RecX.(\tau.X + E) = RecX.\tau.E$

  (Ung3) $RecX.(\tau.(X + E) + F) = RecX.(\tau.X + E + F)$
- The problem arises when *E is "δ.F"* in (Ung2).



1.4 A solution based on "scope"

- The role of (Ung2) is important! It *equates τ-divergent expressions to non-divergent ones*.
- A previous proposal (Hermanns – Lorey '98) solved the problem by using an *equivalence which is sensible to τ-divergence*.
- Idea: introducing a new operator *"pri(E)"* which computes the *prioritized behavior of E*! (removes initial "δ" transitions and subsequent behaviors).
- The new axiom is:

  (Ung2) $RecX.(\tau.X + E) = RecX.\tau.pri(E)$

1.5 A complete axiomatization

- Priority:

(Pri1)	$pri(\underline{0})$	=	$\underline{0}$
(Pri2)	$pri(\alpha.E)$	=	$\alpha.E$
(Pri3)	$pri(\delta.E)$	=	$\underline{0}$
(Pri4)	$pri(E + F)$	=	$pri(E) + pri(F)$
(Pri5)	$pri(pri(E))$	=	$pri(E)$
(Pri6)	$\tau.E + F$	=	$\tau.E + pri(F)$

- Note: "$\tau.E + \delta.F = \tau.E$" is a special case

- Unguarded recursion:

(Ung1)	$RecX.(X + E)$	=	$RecX.E$
(Ung2)	$RecX.(\tau.X + E)$	=	$RecX.\tau.pri(E)$
(Ung3)	$RecX.(\tau.(X+E)+F)$	=	$RecX.(\tau.X+E+F)$
(Ung4)	$RecX.(\tau.(pri(X)+E)+F)$	=	$RecX.(\tau.X+E+F)$

- (Ung4) is needed to *remove weakly unguarded occurences of pri(X)* introduced by new (Ung2)
- The proof is based on unique solution of standard equation sets whose characterization is a variant of the standard Milner's one.

1.6 Problems with parallel operator

- If *local priority* are assumed (e.g. in "$\tau.E \mid \delta.F$", δ is not pre-empted) then we obtain a calculus for which observational equivalence *is a congruence*:
  - we need to deal with locations in the semantics!
- In the case of *global priority* (e.g. in "$\tau.E \mid \delta.F$", δ is pre-empted):

$$\frac{P \xrightarrow{\delta} P' \quad Q \not\xrightarrow{\tau}}{P \mid\mid Q \xrightarrow{\delta} P' \mid\mid Q}$$

observational equivalence *is not a congruence*!

1.7 The problem with global priority

- $RecX.\tau.\ X \simeq \tau.\underline{0}$ but

  $RecX.\tau.\ X \mid\mid \delta.\underline{0} \not\simeq \tau.\underline{0} \mid\mid \delta.\underline{0}$
- The problem with congruence in the global priority approach is related to the behavior of parallel in the presence of *processes which may execute neither "τ" prefixes, nor "δ" prefixes*.
- Such processes (among which is "$\underline{0}$") are managed as allowing *any amount of time to elapse* before executing visible prefixes (if any).

- On the contrary, *observational equivalence* treats them as *processes which do not allow time to elapse*: e.g. "$\tau.\underline{0}$" is equivalent to "$RecX.\tau.\ X$", i.e. time deadlock.
- A possible solution, adopted, e.g., in Hermanns-Lohrey '98, is *to consider a finer notion of equivalence which is sensitive to τ-divergence*, so to get a congruence.

2. Discrete real-time

- In the discrete real-time approach elapsing of time is represented, exactly as in our basic calculus, by a *special action "δ" called a "tick"*.
- In this context *simple solution* to the problem of introducing parallel in the basic calculus:
  - *adopting the particular form of priority in the Hennessy-Regan calculus which is neither local nor really global, but is specialized for time.*

2.1 The Hennessy-Regan approach

- The parallel operator in such a calculus allows for time to elapse in a process only when the other process *may explicitely allow* for time to pass via δ transitions:

$$\frac{P \xrightarrow{\delta} P' \quad Q \xrightarrow{\delta} Q'}{P \,||_S\, Q \xrightarrow{\delta} P' \,||_S\, Q'}$$

- This is similar to global priority, but *changes the interpretation of processes which may execute neither "τ" prefixes nor "δ" prefixes* as desired: e.g. now "$\underline{0}$" is correctly treated as a time-deadlock from the parallel operator as well!

2.2 New prefix and choice operators

- The calculus that we consider (for specifications):

$$P ::= \underline{0} \;\big|\; \pi\underline{.}P \;\big|\; P \underline{+} P \;\big|\; P \,||_S\, P \;\big|\; P/L \;\big|\; RecX.P \;\big|\; X$$

- Old "π . *P*" and "*P* + *Q*" are auxiliary operators
- The *new operator "π . P"* is defined to be *"RecX (δ. X + π.P)" if "π" is visible* (it must be explicitely allowed to be delayed), "π . *P*" otherwise.
- The *new operator "P + Q"* is defined in such a way that the execution of *delays* by "P" or "Q" (now used just to represent time passage) *does not resolve the choice* (they must synchronize).

2.3 Axiomatization of new operators

- Complete axiomatizations for the new "π . *P*" and "*P* + *Q*" operators and the parallel operator are produced by turning terms into *normal forms*:
  terms of the basic calculus
- For parallel and "new" choice this is done in a standard way by introducing *auxiliary operators*:
  - left and syncronization merge for parallel
  - analogous of *synchronization merge* for choice

3. Markovian stochastic time

- Elapsing of time is represented by *special prefixes "λ"* (λ is a real number), denoting *time delays with a probabilistic duration*:
  - a (continuous) exponential distribution with parameter "λ" (intuitively *the speed* of the delay).
- Limitation to exponential distributions:
  - *parallel* of delays is simply *their interleaving*
  - we get a simple *Continuous-time Markov Chain*
- We consider a trivial variant of the basic calculus:
  - *numerical "λ" prefixes replace the "δ" prefix* and "λ" transitions are matched according to *standard Markovian bisimulation*.

3.1 Introducing a parallel operator

- Extending the basic Markovian calculus with the parallel operator is not trivial and presents exactly the same problems that we have explained.
- If we consider a parallel operator with a *global priority mechanism* then we have to *modify the notion of weak bisimultion equivalence* that we consider so to get a congruence.
- This is exactly the case of Hermann's *calculus of Interactive Markov Chains* which adopts a notion of *bisimulation sensible to τ-divergence*.

3.2 A technique à la Hennessy-Regan

- An alternative way is to adopt a technique similar to the Hennessy-Regan one where we say that *time is allowed to pass for a process only if the other one may explicitely make it pass*:

$$\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\lambda'}}{P \,||_S\, Q \xrightarrow{\lambda} P' \,||_S\, Q}$$

- This has the advantage of relying on a *simpler and coarser notion of equivalence*.

3.3 The calculus

- The calculus that we consider (for specifications):

  $P ::= \underline{0} \mid \pi\underline{.}P \mid P \underline{+} P \mid P \parallel_S P \mid P/L \mid RecX.P \mid X$

  where "$\pi\underline{.}P$" is either "$\alpha\underline{.}P$" (α is a or τ) or "$\lambda\underline{.}P$".
- Old "π . P" and "$P + Q$" are auxiliary operators. The new operators "$\pi \underline{.} P$" and "$P \underline{+} Q$" are defined similarly as for the discrete real-time case.
- The new prefix and choice operators allow for an even coarser notion of equivalence which *abstracts from exponential selfloops* (they can never be unfolded by an operator like "+").

3.4 Complete axiomatization

- Complete axiomatization is produced by turning terms into *normal forms*:
  - *terms of the basic (Markovian) calculus*
- For parallel and "new" choice this is done in a standard way by introducing *auxiliary operators*:
  - left and syncronization merge for parallel
  - analogous of *left merge* for choice
- A completely new axiom characterizes *abstraction from selfloops in Markovian calculi*:

  (ExpRec) $RecX.(\lambda.X + \lambda'.P + Q) = RecX.(\lambda'.P + Q)$

4. Non-atomic time delays

- In order to represent *more complex time models*, we need to consider semantics where:
  - delays *not executed atomically* in a single transition
  - but that *start in a given state, evolve through several states and terminate in another state*
- This is needed, e.g., to represent *continuous real-time* (as in Alur and Dill's timed automata) or *stochastic-time with general distributions*.

4.1 Not a new problem!

- Considered a simple calculus like:

  $P ::= \underline{0} \mid \pi.P \mid P + P \mid P \parallel_S P \mid P/L \mid RecX.P \mid X$

  where "$\pi.P$" is either "$\alpha.P$" (α is a or τ) or "$\delta.P$".
- How to represent the execution of a time delay "δ" as the combination of the two events of:

  delay start *delay termination*

  in such a way that termination of a given delay is *uniquely related* to its start?
- Already considered in the literature: *ST semantics* (van Glabbeek and Vaandrager '87)

4.2 Decide & axiomatize ST semantics

- We have introduced *3 techniques* for decideing and axiomatizing ST semantics:
  - *static name technique*:
    - statically generates a name for every action according to its *syntactical position* in the term (location w.r.t. parallel)
    - allows ST bisimulation to be decided for finite-state terms
  - *dynamic name technique*:
    - dynamically generates a *canonical name* for every starting action according to the *order of execution* of actions: smallest number not in use by actions of the same type
    - ST bisimulation in terms of standard bisimulation: complete axiomatization and decidability for finite-state terms
  - *stack technique*:
    - based on *pointers*: same properties of dynamic name technique for an algebra including semantic action refinement

4.3 ST semantics for time delays

- Techniques based on names particularly adequate for timed models:
  - they keep the relationship between delay start and terminations by *producing unique names which are like clock names in a timed automata*.
- In particular, *dynamic name techinque* allows us:
  - to simply use *standard (weak) bisimulation*
  - to produce axiomatizations via a *standard approach based on left and synchronization merge*:

    is based on the technique of *levelwise renaming*: canonical names are recomputed every time delays are taken out from the scope of a parallel operator *in the rule for left merge*.

5. Continuous real-time

- Elapsing of time can be represented by *delay prefixes "D"*, where D represents a *set of non-negative real numbers*: the possible durations for the delay.
- D can, e.g., be an interval or a set of intervals obtained via a set of constraints.
- Since we are in a continuous domain we want to obtain *models based on clocks* (like a *timed automata*) *where time elapsing is not explicit*, but expressed symbolically via start and termination of clocks.

5.1 The calculus

- The calculus that we consider (for specifications):

$$P ::= \underline{0} \mid \pi\underline{.}P \mid P \underline{+} P \mid P \parallel_S P \mid P/L \mid RecX.P \mid X$$

where "$\pi \underline{.} P$" is either "$\alpha \underline{.} P$" or *"$D \underline{.} P$"*

- "$\pi . P$" and "$P + Q$" are auxiliary operators.
- We apply *ST semantics with dynamic names* to delay prefixes, thus producing clock names D_i:
  - *"i"* is a number generated by the semantics to *distinguish clocks derived from delays of the same type* (with the *same set of possible durations D*).

5.2 Equivalence and axiomatization

- Equivalence is just *Milner's observational congruence* (where D prefixes are considered to be visible actions): the effect is that *it matches delays with the same set of possible durations D*.
- A complete axiomatization is produced by by turning terms into *normal forms*:
  - *terms of a basic calculus (where we use "$\pi . P$" and "$P + Q$" operators and D_i^+, D_i^- prefixes)*

  by *combining the two techniques* related to priority (maximal progress) and ST semantics.

6. General stochastic time

- Elapsing of time is represented by *delay prefixes "f"*, where f is a *general probability distribution* over non-negative real numbers: it expresses the probabilistic duration of the delay.
- Since we consider continuous general distributions we want to obtain *models based on clocks where time elapsing is not explicit*, but expressed symbolically via start and termination of clocks: a *Generalized Semi-Markov Process*

6.1 The calculus

- The calculus that we consider (for specifications):

$$P ::= \underline{0} \mid \pi\underline{.}P \mid P \underline{+} P \mid P \parallel_S P \mid P/L \mid RecX.P \mid X$$

where "$\pi \underline{.} P$" is either "$\alpha \underline{.} P$" or *"$<f,w> \underline{.} P$"*

- "$\pi . P$" and "$P + Q$" are auxiliary operators.
- We apply *ST semantics with dynamic names* to delays $<f,w>$, thus producing clock names f_i:
  - *"i" distinguishes clocks derived from delays of the same type* (with the same distribution f).
  - the *weight "w" is associated to the transition of start*

6.2 Equivalence and axiomatization

- Equivalence is just *Milner's observational congruence* combined with *standard probabilistic bisimulation* for start of delays: the effect is that *it matches delays with the same distribution f*.
- A complete axiomatization is produced by by turning terms into *normal forms*:
  - *terms of a basic calculus (where we use "$\pi . P$" and "$P + Q$" operators and $<f_i^+,w>$, f_i^- prefixes)*

  by *combining the two techniques* related to priority (maximal progress) and ST semantics.

Open problems and future directions

- The continuous real-time and general stochastic time models could support the possibility of *aggregating*, besides τ actions, also *time delays*. How to express this in the semantics and the equivalence is a difficult open problem (a possibility could be using the stack technique).
- When the capability of expressing time is used in real case studies, often it turns out that an elegant way to *express internal/external probability and priority* is also needed. In spite of the several solutions proposed we still miss a very elegant one.

References

- M. Bravetti, R. Gorrieri ``A Complete Axiomatization for Observational Congruence of Prioritized Finite-State Behaviors", in Proc. of the *27th Int. Colloquium on Automata, Languages and Programming (ICALP 2000)* , U. Montanari, J.D.P. Rolim and E. Welzl editors, LNCS 1853:744-755, Geneva (Switzerland), July 2000.
- M. Bravetti, R. Gorrieri ``Deciding and Axiomatizing Weak ST Bisimulation for a Process Algebra with Recursion and Action Refinement", in ACM Transactions on Computational Logic 3(4):465-520, 2002.
- M. Bravetti, R. Gorrieri ``The Theory of Interactive Generalized Semi-Markov Processes", in Theoretical Computer Science 282(1):5-32, 2002.
- M. Bravetti, ``Specification and Analysis of Stochastic Real-Time Systems", PhD Thesis, University of Bologna, Padova and Venezia, February 2002.
- M. Bravetti, ``Revisiting Interactive Markov Chains", in Proc. of the *3rd Workshop on Models for Time-Critical Systems (MTCS 2002)* , ENTCS 68(5), Brno (Czech Republic), August 2002.

SEMANTIC THEORIES
FOR ASYNCHRONOUS CALCULI

Ilaria Castellani
INRIA Sophia Antipolis

Process Algebra workshop

Bertinoro,
July 21-25, 2003

UNSOLVED PROBLEMS

1) Axiomatisation of weak asynchronous bisimulation:
   enough to add the 2 laws suggested in [ACS 96]?
   Problem: τ-laws need full guarded choice.

2) Axiomatisation of asynchronous must testing:
   - which normal forms?
   - need for proof rules?

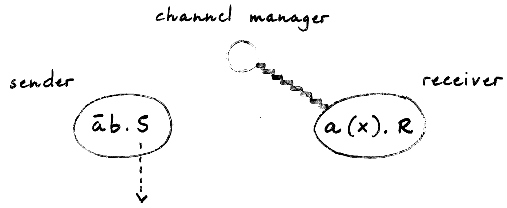
Some laws suggested in [CH 98].

1

Motivations for asynchronous π-calculus

Asynchronous communication

- Simpler implementation:



=> basis for PICT, join-calculus, etc.

- In the π-calculus: simpler syntax, same expressiveness:

  encodings: synchronous $\rightarrow$ asynchronous

- Asynchronous observation:

  appropriate for assessing encodings

2

Aim & Overview

Asynchronous calculi (like async. π) need appropriate semantic framework:

- Asynchronous bisimulation [Honda, Tokoro, Yoshida, Amadio, Cast., Sangiorgi]

↳ **Asynchronous Testing**

(à la De Nicola & Hennessy)

- async. notions of $\underset{\sim}{\sqsubset}_{\mathbf{may}}$ and $\underset{\sim}{\sqsubset}_{\mathbf{must}}$
- alternative characterisations
- axiomatisation

3

The language TACCS

Asynchronous **CCS**, adapted for Testing

$p ::=$	0	
$\mid$	$a.p$	input prefix
$\mid$	$\hat{a}.p$	asynchronous output
$\mid$	$\bar{a}$	atom
$\mid$	$p \parallel p$	parallel
$\mid$	$p \oplus p$	internal choice
$\mid$	$p + p$	external choice

4

Semantics of TACCS

$\alpha, \beta = a, \bar{a}, \tau$

Input $\quad a.p \xrightarrow{a} p$

Output $\quad \hat{a}.p \xrightarrow{\tau} \bar{a} \parallel p$ $\qquad$ Atom $\quad \bar{a} \xrightarrow{\bar{a}} 0$

Parallel:

$$\frac{p \xrightarrow{a} p', \; q \xrightarrow{\bar{a}} q'}{p \parallel q \xrightarrow{\tau} p' \parallel q'} \qquad \frac{p \xrightarrow{\beta} p'}{p \parallel q \xrightarrow{\beta} p' \parallel q}$$

5

Semantics (ctd)

Int $\quad p \oplus q \xrightarrow{\tau} p \quad \dots$

Ext $\quad \dfrac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \dfrac{p \xrightarrow{\tau} p'}{p+q \xrightarrow{\tau} p'+q}$

$$\dfrac{p \xrightarrow{\bar{a}} p'}{p+q \xrightarrow{\tau} \bar{a} \parallel p'}$$

outputs are asynchronous wrt +

6

New rule for +

- **CCS** : + is a synchronising oper.

$$\dfrac{p \xrightarrow{\bar{a}} p'}{p+q \xrightarrow{\bar{a}} p'}$$

- **TACCS** : asynchronous world

$$\dfrac{p \xrightarrow{\bar{a}} p'}{p+q \xrightarrow{\tau} \bar{a} \parallel p'}$$

the output is consumed later

7

New rule for +

- **CCS** : + is a synchronising oper.

$$\dfrac{p \xrightarrow{\bar{a}} p'}{p+q \xrightarrow{\bar{a}} p'}$$

- **TACCS** : buffer - insensitive

$$0 \simeq a.\bar{a}$$

with CCS rule :

$$0 \not\sqsubseteq_{must} a.\bar{a}$$

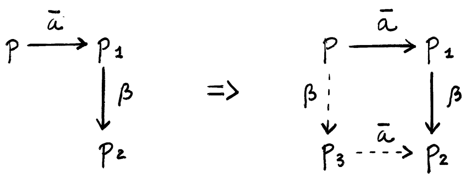
must $\bar{a} + (\omega \oplus \omega)$ $\qquad$ ~~must~~ $\bar{a} + (\omega \oplus \omega)$

8

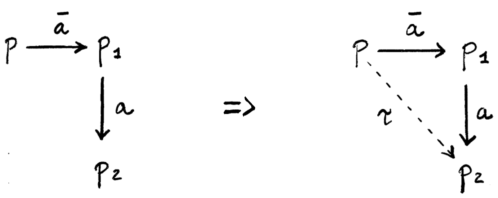
Asynchrony of outputs

Properties of asynchronous actions :
[Selinger 97]

Backward commutativity



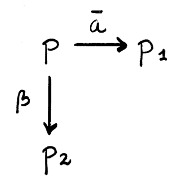
Feedback



9

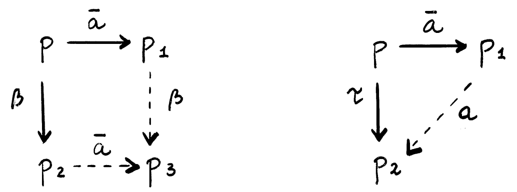
Async. of outputs (ctd)

Forward commutativity



$\Rightarrow$

either $\beta = \bar{a} \wedge p_1 = p_2$ or **2 cases :**



10

TESTING SCENARIO

Observe via **tests** : processes with special **"success"** action ω

Ex. $e = \omega$

$e' = a.\omega + b$

$\underline{p \textbf{ may } e}$ if $\exists$ maximal computation

$$p \parallel e \xrightarrow{\tau}{}^{*} p' \parallel e' \xrightarrow{\tau}{}^{*} \cdots$$

such that $e' \xrightarrow{\omega}$

$\underline{p \textbf{ must } e}$ if $\forall$ max. comp. $e' \xrightarrow{\omega}$

11

Testing scenario (ctd)

- **Testing preorders** :

$\underline{p \sqsubseteq_{may} q}$ if $\forall e \; (p \text{ may } e \Rightarrow q \text{ may } e)$

$\underline{p \sqsubseteq_{must} q}$ if $\forall e \; (p \text{ must } e \Rightarrow q \text{ must } e)$

$\underline{p \sqsubseteq q}$ if $p \sqsubseteq_{may} q \wedge p \sqsubseteq_{must} q$

Equivalences : $\simeq \; = \; \sqsubseteq \cap \sqsupseteq$, ...

12

Testing preorders : examples

$a \sqsubseteq 0$

must $\hat{a}.\omega$

$0 \sqsubseteq_{may} a$

$0 \not\sqsubseteq_{must} a$

$\not$must $a.\omega \parallel \bar{a}$

13

Testing preorders : examples

$a \lesssim 0$

must $\hat{a}.\omega$

$0 \lesssim_{may} a$ $\qquad$ $0 \not\lesssim_{must} a$

Deadlock on a $\begin{cases} \text{async}: & (a.\omega \parallel \bar{a}) \\ \text{sync.}: & (\omega \oplus \omega) + \bar{a} \end{cases}$

---

equated with buffering on a :

$0 \simeq a.\bar{a}$

must $(a.\omega \parallel \bar{a})$

14

Characterisation of may (synchronous)

Observable sequences $\stackrel{s}{\Longrightarrow}$: $\qquad (\mu = a, \bar{a})$

$p \stackrel{\varepsilon}{\Longrightarrow} p$

$p \stackrel{\tau}{\longrightarrow} p',\ p' \stackrel{s}{\Longrightarrow} p'' \quad \Rightarrow \quad p \stackrel{s}{\Longrightarrow} p''$

$p \stackrel{\mu}{\longrightarrow} p',\ p' \stackrel{s}{\Longrightarrow} p'' \quad \Rightarrow \quad p \stackrel{\mu s}{\Longrightarrow} p''$

Fact: $p \stackrel{s}{\Longrightarrow} \iff p \text{ may } \bar{s}.\omega$

$\mathcal{L}(p) = \{ s \mid \exists p'.\ p \stackrel{s}{\Longrightarrow} p' \}$

May testing :

$p \lesssim_{may} q \iff \mathcal{L}(p) \subseteq \mathcal{L}(q)$

15

Characterisation of may

$a \lesssim_{may} 0$

but $\mathcal{L}(a) \not\subseteq \mathcal{L}(0)$

$p \stackrel{s}{\Longrightarrow} \;\not\Leftarrow\; p \text{ may } \bar{s}.\omega$

$0 \parallel \hat{a}.a.\omega \stackrel{\tau}{\longrightarrow}{}^{*} 0 \parallel \omega$, but $0 \stackrel{a\bar{a}}{\not\Longrightarrow}$

16

Characterisation of may

13

$a \lesssim_{may} 0$

but $\mathcal{L}(a) \not\subseteq \mathcal{L}(0)$

$p \stackrel{s}{\Longrightarrow} \;\not\Leftarrow\; p \text{ may } \bar{s}.\omega$

$0 \parallel \hat{a}.a.\omega \stackrel{\tau}{\longrightarrow}{}^{*} 0 \parallel \omega$, but $0 \stackrel{a\bar{a}}{\not\Longrightarrow}$

Asynchronous sequences $\stackrel{s}{\Longrightarrow}_a$:

$p \stackrel{b}{\longrightarrow}_a \bar{b} \parallel p \qquad \forall$ input b

$\mathcal{L}^a(p) = \{ s \mid \exists p'.\ p \stackrel{s}{\Longrightarrow}_a p' \}$

17

Characterisation of may

$a \sqsubseteq_{may} 0$

but $\mathcal{L}(a) \not\subseteq \mathcal{L}(0)$

$$p \overset{s}{\Longrightarrow}_a \iff p \text{ may } \bar{s}.\omega$$

$0 \parallel \hat{a}.a.\omega \overset{\tau}{\longrightarrow}{}^{*} 0 \parallel \omega$ and $0 \overset{a\bar{a}}{\Longrightarrow}_a$

Asynchronous sequences $\overset{s}{\Longrightarrow}_a$:

$p \overset{b}{\longrightarrow}_a \bar{b} \parallel p \qquad \forall$ input b

$\mathcal{L}^a(p) = \{ s \mid \exists p'.\ p \overset{s}{\Longrightarrow}_a p' \}$

May testing:

$$p \sqsubseteq_{may} q \iff \mathcal{L}^a(p) \subseteq \mathcal{L}^a(q)$$

18

Characterisation of must (synchronous)

Ready set:

$R(p) = \{ \alpha \mid \alpha \neq \tau,\ p \overset{\alpha}{\longrightarrow} \}$

Acceptance set of p after s:

$\mathcal{A}(p,s) = \{ R(p') \mid p \overset{s}{\Longrightarrow} p' \overset{\tau}{\not\longrightarrow} \}$

$p \ll q$ if $\forall s,\ \forall A \in \mathcal{A}(q,s)$
$\exists A' \in \mathcal{A}(p,s)$ s.t. $A' \subseteq A$

Must testing

$$p \sqsubseteq_{must} q \iff p \ll q$$

19

Characterisation of must

Remark 1

Ready sets restricted to outputs

$a.\bar{a} \sqsubseteq_{must} 0$

$\mathcal{A}(p,\varepsilon) = \{\{a\}\} \not\subset\subset \mathcal{A}(q,\varepsilon) = \{\emptyset\}$

Remark 2

In $\mathcal{A}(p,s)$ s should be asynchronou

$0 \sqsubseteq_{must} a.\bar{a}$

$\mathcal{A}(p,a) = \emptyset \not\subset\subset \mathcal{A}(q,a) = \{\{\bar{a}\}\}$

$$\mathcal{O}^a(p,s) = \{ O(p') \mid p \overset{s}{\Longrightarrow}_a p' \overset{\tau}{\not\longrightarrow} \}$$

20

Char. of must (ctd)

Remark 3

$a.\bar{b} \ll 0$

$\mathcal{O}^a(p,\varepsilon) = \{\emptyset\} \subset\subset \mathcal{O}(q,\varepsilon) = \{\emptyset\}$

$a.\bar{b} \not\sqsubseteq_{must} 0$

must $\bar{a} \parallel b.\omega$

$\mathcal{O}^a_I(p,s) = \{ O(p'') \mid p \overset{s}{\Longrightarrow}_a p' \Downarrow_I p'' \}$

$p \ll q$ if $\forall s,\ \forall O \in \mathcal{O}(q,s),\ \forall I$ s.t. ..
$\exists O' \in \mathcal{O}^a_I(p,s)$ s.t. $O' \setminus \bar{I} \subseteq O$

Must testing

$$p \sqsubseteq_{must} q \iff p \ll q$$

21

AXIOMATISATION

Both preorders:

- Standard laws of testing:

$$x \oplus y \leqslant x + y$$

$$\vdots$$

- New **laws for asynchrony**:

$$\text{A1.} \quad \hat{a}.x = \bar{a} \parallel x$$
$$\text{A2.} \quad a.(\bar{a} \parallel x) + x = x$$

May testing: add

$$x \leqslant x \oplus y$$

A2. can be derived from $a.(\bar{a} \parallel x) \leqslant x$

22

AXIOMATISATION (ctd)

Must testing: add

$$x \oplus y \leqslant x$$

- Another asynchrony law:

$$\text{A3.} \quad x + \hat{a}.y \leqslant (x + \hat{a}.y) \oplus \hat{a}.y$$

- **Conditional rules** R1, R2, R3:

$$\vdots$$

$$\text{R3.} \quad \frac{x + y \leqslant x}{(x + \hat{a}.z) \oplus y \leqslant x}$$

Ex. $(b + \hat{a}.0) \oplus 0 \leqslant b$

23

FUTURE / RELATED WORK

- Complete axiomatisation for $\sqsubseteq_{must}$
- Extension of the theory to the asynchronous π-calculus

Related work

- Boreale et al. (98, 99):

  Axiomatisation of may testing over asynchronous CCS and π

24

Back to asynchronous bisimulation

(it was defined on the asynchronous π-calc. with input and τ guarded sums, but we consider here the case of asynchronous CCS)

25

ASYNCHRONOUS CCS (ACCS)

$$G ::= 0 \mid \mu.P \mid G+G \qquad \mu = a, \tau$$
$$P ::= \bar{a} \mid P|Q \mid P\backslash a \qquad \alpha,\beta = a, \bar{a}, \tau$$

$$\mu.P \xrightarrow{\mu} P \qquad \bar{a} \xrightarrow{\bar{a}} 0$$

$$\frac{G_1 \xrightarrow{\mu} G_1'}{G_1 + G_2 \xrightarrow{\mu} G_1'} \; \dots \qquad \frac{P \xrightarrow{\alpha} P', \; \alpha \neq a, \bar{a}}{P\backslash a \xrightarrow{\alpha} P'\backslash a}$$

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \; \dots \qquad \frac{P \xrightarrow{a} P', \; Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

26

Axiomatization for ACCS (strong case)

Asynchrony law

$$a.(\bar{a} \mid P) + \tau.P = \tau.P$$

Normal forms

$$\Pi\, \bar{a}_i \mid (\Sigma\, \tau.P_j + \Sigma\, a_k.Q_k)$$
$$Q_k \not\equiv \bar{a}_k.P_j$$

Weak case ?

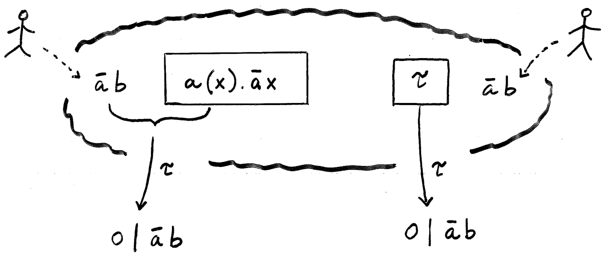
Asynchrony laws

$$a.(\bar{a} \mid G) + G = G$$
$$a.(\bar{a} \mid a.P) = a.P$$

27

Asynchronous bisimulation $\sim_a$

Intuition: asynchronous observer cannot observe the inputs of a process



Bisimulation $\sim_a$ ... $P\,R\,Q$ implies:

1) Usual condition for outputs and τ

2) $P \xrightarrow{ab} P' \implies$

$$\begin{cases} \text{either } Q \xrightarrow{ab} Q' \wedge P'\,R\,Q' \\ \text{or } Q \xrightarrow{\tau} Q' \wedge P'\,R\,(Q' \mid \bar{a}b) \end{cases}$$

28

CONCLUSIONS

Weak asyn. bisim. : take full guarded choice

$$\frac{G \xrightarrow{\bar{a}} G'}{G + G'' \xrightarrow{\tau} \bar{a} \mid G'}$$

and add new axiom : $\bar{a} = \tau\bar{a}$?

Asyn. must testing : next edition of the workshop ...

Thanks to organizers !

29

An Equational Axiomatization of Milner Bisimulation in Kleene Stars

Flavio Corradini

Università di L'Aquila, Italy
Dipartimento di Informatica
flavio@di.univaq.it

Joint Work with:
Rocco De Nicola
Anna Labella

Regular Expressions

Fix an alphabet
A={a,b,c,…}

The set of regular expressions is defined by:
E::= 0 | 1 | a | E+E | E•E | E*, a ∈ A

Regular Languages
- L[0] = ∅
- L[1] = {λ}
- L[a] = {a}
- L[E+F] = L[E] ∪ L[F]
- L[E•F] = L[E] • L[F]
- L[E*] = (L[E])*

Language (Trace) Equivalence
E = F if L[E] = L[F]

Axioms for Language Equivalence

Salomaa's Axiomatization

X+Y=Y+X	(X•Y) •Z=X• (Y•Z)
(X+Y)+Z=X+(Y+Z)	X•1=X=X•1
X+0=X	X•0=0=0•X
X+X=X	
(X+Y) •Z=(X•Z)+(Y•Z)	X*=1+X•X*
X •(Y+Z)=(X•Y)+(X•Z)	X*=(1+X)*

$$\frac{Y = X \bullet Y + Z \text{ and } X \text{ does not possess the e.w.p.}}{Y = X^* \bullet Z}$$

Where a regular expression E possesses the e.w.p., written ewp(E), if
- ewp(1), ewp(E*),
- ewp(E) ∨ ewp(F) implies ewp(E+F)
- ewp(E) ∧ ewp(F) implies ewp(E • F)

The "Non Deterministic" Interpretation of Regular Expressions

Regular Expressions and their Interpretation

E::=	0 |	Deadlock
	1 |	Successful Termination
	a (a ∈ A) |	Basic Process
	E+E |	Non Deterministic Composition
	E•E |	Sequential Composition
	E*	Recursive Definition

Operational Semantics (via labelled transition systems):

$E \xrightarrow{a} F$ "Process" E becomes F after performing "action" a

$E\ \surd$ "Process" E can immediately terminate

Observational Semantics

Bisimulation Equivalence

The Operational Semantics

Act $\dfrac{}{a \xrightarrow{a} 1}$

Ter $\dfrac{}{1\ \surd}$

Sum $\dfrac{E \xrightarrow{a} E'}{E+F \xrightarrow{a} E'}$ + symmetric

TSum $\dfrac{E\ \surd}{E+F\ \surd}$

Seq1 $\dfrac{E \xrightarrow{a} 1}{E \bullet F \xrightarrow{a} F}$

TSeq $\dfrac{E\ \surd,\ F\ \surd}{E \bullet F\ \surd}$

Seq2 $\dfrac{E \xrightarrow{a} E' \neq 1}{E \bullet F \xrightarrow{a} E' \bullet F}$

Seq3 $\dfrac{E\ \surd,\ F \xrightarrow{a} F'}{E \bullet F \xrightarrow{a} F'}$

Kle1 $\dfrac{E \xrightarrow{a} 1}{E^* \xrightarrow{a} E^*}$

Kle2 $\dfrac{E \xrightarrow{a} E' \neq 1}{E^* \xrightarrow{a} E' \bullet E^*}$

TKle $\dfrac{}{E^*\ \surd}$

The Observational Semantics and Milner's Open Problem

E ∼ F iff ∀a ∈ A

(i) $E\ \surd$ iff $F\ \surd$

(ii) $E \xrightarrow{a} E'$ implies $F \xrightarrow{a} F'$ and $E' \sim F'$

(iii) $F \xrightarrow{a} F'$ implies $E \xrightarrow{a} E'$ and $E' \sim F'$

Milner's Open Problem (JCSS' 84)

Is Salomaa Axiomatization without axioms

X • (Y+Z)=(X•Y)+(X•Z)
X • 0 = 0

complete with respect to bisimulation?

The question is still open…

but interesting results are due to Aceto, Fokkink, Ingolfsdottir, Zantema…

Wan Fokking axiomatization regards regular expressions

- without 0 and 1
- with "Binary Kleene Stars" : E*F (where ¬F√)

Ex: a*b and a*b*c are "well-formed" regular expressions, a*b* is not

X+Y=Y+X	(X•Y)•Z=X• (Y•Z)
(X+Y)+Z=X+(Y+Z)	(X+Y)•Z=(X•Z)+(Y•Z)
X+X=X	X*(Y•Z)=(X*Y)•Z
X*Y=Y+X•X*Y	
(X+Y)*•Z=X*(Z+Y•(X+Y)*Z)	

Fokkink's Complete Axiomatization

"Our Question"

Is there a finite equational axiomatization of Milner's bisimulation over regular expressions with 0 and/or 1?

Consider 1, first.

We strenghten Salomaa's Empty Word Propert:

E* possesses the hereditary non empty word property if there is no E' such that $E \xrightarrow{a1} \dots \xrightarrow{an} E' \sim 1+F \wedge F \xrightarrow{a}$

Ex: a*b* has the hnewp as ((1+a)b)*.
(b(1+a))* does not have the hnewp.

Main result

Under the hereditary non empty word property assumption, the set of axioms:

X+Y=Y+X	(X•Y)•Z=X• (Y•Z)
(X+Y)+Z=X+(Y+Z)	(X+Y)•Z=(X•Z)+(Y•Z)
X+X=X	X•1=1•X=X
X*=1+X•X*	
(X+Y)*=X*(1+Y•(X+Y)*)	

provides a sound and complete finite equational axiomatization of Milner's bisimulation.

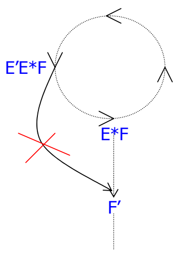
F.Corradini: A Step Forward Towards Equational Axiomatizations of Milner Bisimulation in Kleene Star. Proceedings of "Fixed Points in Computer Science", FICS 2000.

F. Corradini, R. De Nicola. A. Labella: An Equational Axiomatization of Bisimulation over Regular Expressions. Journal of Logic and Computation, 12, pp. 301-320, 2002.

Key Observations for the Proof

(i) We need a well-founded ordering over regular expressions E < F which makes sure:
 (a) E' < E*F, where E', F' derivatives of E, F respectively
 (b) F' < E*F

(ii) In a cycle E* F, …, E' E* F, … a derivative of F cannot be an immediate derivative of E' E* F (recall the hnewp).



(iii) Decomposition of Regular Expressions
E • F ~ G • F implies E ~ G

Proof by Structural Induction

Is our Axiomatization Complete for (general) Regular Expressions?

Consider
(a+b)* and a*(ba*)*

they are bisimilar but --- (a+b)*= a*(ba*)* --- cannot be proven in our axiom system!

The formal proof resorts on the following property.

Property:
If E ~ F and (G+H)*, where ¬ (G ~ H) appears in E then (X+Y+Z)* appears in F with G ~ X and H ~ Y

If we abondon the hnewp then such a property does not hold anymore…

Note: (a+b)* and a*(ba*)* do not even have Salomaa's ewp!

Some Concluding Remarks

- Language Equivalence is not finitely axiomatizable even in the language we consider.
 (Redko's counterexample $a^* \sim_{trace} (a^n)^*(1+a+\dots+a^{n-1})$ applies also under the hnewp)
- If star expressions do not possess the hnewp then "non deterministic behaviours" are not preserved by bisimulation.
- If star expressions do not posses the hnewp the our axiom system is no more complete

Our Conjecture

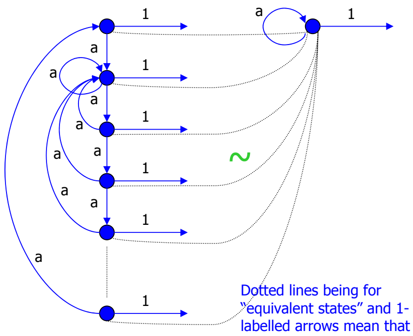
The set of regular expressions (without 0) with hnewp is the largest language for which bisimulation admits a finite equational axiomatization.

Consider:

E= (a + 1)* and

F= (a(a(…(a(a+1)+1)…)+1)+1)*

p times, p prime



Dotted lines being for "equivalent states" and 1-labelled arrows mean that the source states have the e.w.p.

Axioms for 0

- Milner's 0 Object

  It satisfies the axioms:
  X + 0 = X
  0 • X = 0

  but neither
  X • 0 = 0, nor
  X • 0 = X

A result by Sewel states that bisimulation cannot be finitely axiomatizable.
An instance of his counterexample is: a* • 0 ~ (a • a)* •0

Note 1: a* • 0 and (a • a)* •0 have the hnewp.

Note 2: a* • 0 ~ (a • a)* • 0
E • F ~ G • F does not imply E ~ G

- But if we take "0" which satisfies
  X + 0 = X
  0 • X = 0
  X • 0 = 0

  then bisimulation can be finitely axiomatized.

Klaim, Formulae and Contexts

Rocco De Nicola
Dip. Sistemi e Informatica
Università di Firenze
denicola@dsi.unifi.it

Joint, work mainly, with M. Loreti
but also with R. Pugliese, G.L. Ferrari, L. Bettini

Outline

- Motivations
- Klaim
- μ-Klaim
- A Logic for μ-Klaim
- Systems Properties
- Open Nets
- Contexts
- Approximation and Refinements
- A couple of results
- Conclusions
- The Klaim Project
  - On Going Work
  - Software
  - References

Global Systems

- Are *Distributed Systems* with distinguishing features such as:
  - Wide area distribution
  - Variable interconnection structures
  - (Physical and Logical) Mobility
  - Latency and bandwidth issues
  - Failures

Programming Global Systems

Explicit Primitives for

- Distribution
  computing over different (explicit) localities
- Mobility
  moving agents and computations over localities
- Concurrency
  considering parallel and non-deterministic computations
- Access Rights
  maintaining privacy and integrity of data

Objectives

Developing a simple programming language for network aware and migrating applications with a tractable semantic theory that permits programs verification.

Klaim

Kernel Language for Agent Interaction and Mobility

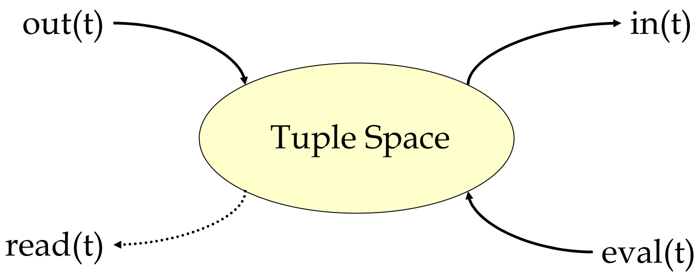
- Process Calculus Flavored
- Linda based communication model:
  - Asynchronous communication;
  - Via tuple space.
- Explicit use of *localities*:
  - Multiple distributed tuple spaces.
- Code mobility.

Linda Communication Model

- Tuples ("foo", 10+5, !x)
  - Formal Fields
  - Actual Fields
- Pattern Matching:
  - Formal fields match any field of the same type
  - Actual fields match if identical

    ("foo", 10+5, true) matches (!s, 15, !b)

Linda Communication Model



Philosophers dining with Linda

Phi(int i) { while true {

think();

in("ticket"); **in**("fork", i); **in**("fork", i+1%5);

eat();

out("fork", i); **out**("fork", i+1%5); **out**("ticket")}}

main() {

int I; for{ (i=0; i<5,i++)

out("fork", i); **eval**(Phi(i));

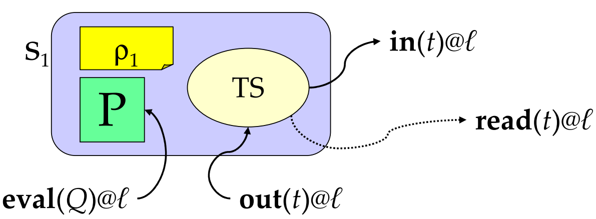
if (i<4) **out**("ticket")}}

From a Linda based process calculus to Klaim

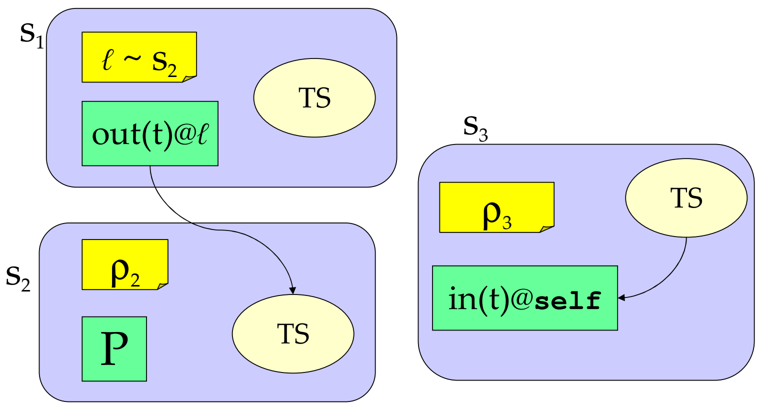
- Localities to model distribution
  - Physical Locality (sites)
  - Logical Locality (names for sites)
  - A distinct name *self* indicates the site a process is on.
- Allocation Environment to associate sites to logical locality
  - This avoids the programmers to know the exact physical structure.

Klaim Nodes

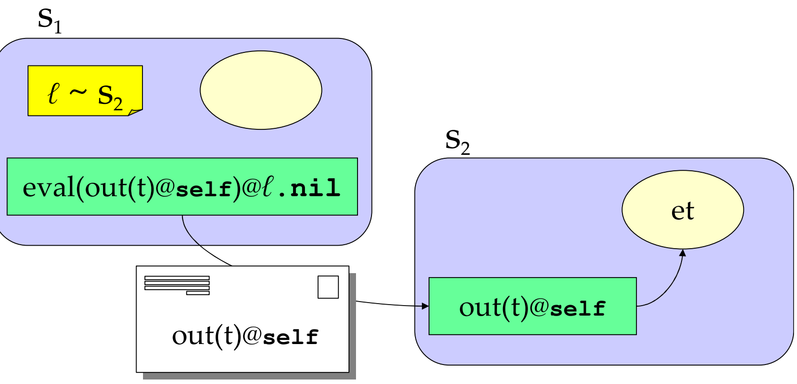
- Name (phys. loc.)
- Tuple space
- Processes
- Environment



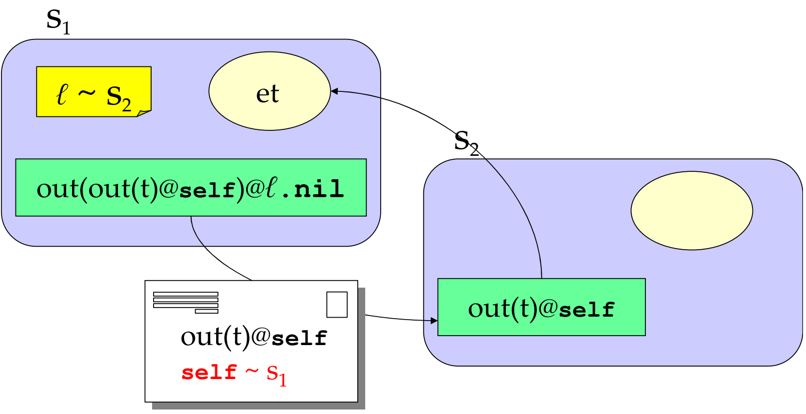
Klaim Nets



Dynamic Scoping



Static Scoping



Klaim Processes

P	$::=$	$\mathbf{nil}$	(null process)
	$\mid$	$a.P$	(action prefixing)
	$\mid$	$P_1 \mid P_2$	(parallel composition)
	$\mid$	X	(process variable)
	$\mid$	$A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{e} \rangle$	(process invocation)

$$a \quad ::= \quad \mathbf{out}(t)@\ell \;\big|\; \mathbf{in}(t)@\ell \;\big|\; \mathbf{read}(t)@\ell \;\big|\; \mathbf{eval}(P)@\ell \;\big|\; \mathbf{newloc}(u)$$

$$t \quad ::= \quad f \;\big|\; f, t$$

$$f \quad ::= \quad e \;\big|\; P \;\big|\; \ell \;\big|\; !\,x \;\big|\; !\,X \;\big|\; !\,u$$

Nets

N	$::=$	$s ::_{\rho} P$	(node)
	$\mid$	$N_1 \parallel N_2$	(net composition)

μ_Klaim: A core calculus for Klaim

- We take away from Klaim:
  - distinction between logical and physical localities/addresses (no allocation environment)
  - higher order communication (no process in tuples)

R. De Nicola ProcessAlgebras@Bertinoro 17

μ_Klaim syntax

- Nets

$$N \quad ::= \quad l :: P \;\Big|\; N_1 \parallel N_2$$

- Processes

$$P \quad ::= \quad \mathbf{nil} \;\Big|\; a.P \;\Big|\; a.P_1 + a.P_2 \;\Big|\; P_1 \mid P_2 \;\Big|\; A \quad (A \triangleq P)$$

- Actions

$$a \quad ::= \quad \mathbf{read}(T)@\ell \;\Big|\; \mathbf{in}(T)@\ell \;\Big|\; \mathbf{out}(t)@\ell \;\Big|\; \mathbf{eval}(P)@\ell \;\Big|\; \mathbf{newloc}(u)$$

R. De Nicola ProcessAlgebras@Bertinoro 18

Tuples and Templates

Templates	T	$::=$	$F \mid F, T$
Tem.Fields	F	$::=$	$f \mid\, !\,x \mid\, !\,u$
Tuples	t	$::=$	$f \mid f, t$
Tuple Fields	f	$::=$	$e \mid \ell$
Expressions	e	$::=$	$V \mid x \mid \ldots$

R. De Nicola ProcessAlgebras@Bertinoro 19

Matching Rules

(M$_1$) $match(V, V) = \epsilon$ (M$_2$) $match(!\,x, V) = [^V/_x]$

(M$_3$) $match(l, l) = \epsilon$ (M$_4$) $match(!\,u, l) = [^l/_u]$

(M$_5$) $$\frac{match(F, f) = \sigma_1 \quad match(T, t) = \sigma_2}{match(\ (F, T)\ ,\ (f, t)\ ) = \sigma_1 \circ \sigma_2}$$

R. De Nicola ProcessAlgebras@Bertinoro 20

Structural Congruence

(Com)	$N_1 \parallel N_2 \equiv N_2 \parallel N_1$
(Assoc)	$(N_1 \parallel N_2) \parallel N_3 \equiv N_1 \parallel (N_2 \parallel N_3)$
(Abs)	$l::P \equiv l::(P \mid \mathbf{nil})$
(PrInv)	$l::A \equiv l::P \quad \text{if } A \triangleq P$
(Clone)	$l::(P_1 \mid P_2) \equiv l::P_1 \parallel l::P_2$

μ-Klaim Semantics (1)

$$\frac{[\![t]\!] = et}{l::\mathbf{out}(t)@l'.P \parallel l'::P' \xrightarrow{\mathrm{o}(l,et,l')} l::P \parallel l'::P' \parallel l'::\langle et \rangle}$$

$$l::\mathbf{eval}(Q)@l'.P \parallel l'::P' \xrightarrow{\mathrm{e}(l,\ ,l')} l::P \parallel l'::P'|Q$$

$$\frac{match([\![T]\!], et) = \sigma}{l::\mathbf{in}(T)@l'.P \parallel l'::\langle et \rangle \xrightarrow{\mathrm{i}(l,et,l')} l::P\sigma \parallel l'::\mathbf{nil}}$$

$$\frac{match([\![T]\!], et) = \sigma}{l::\mathbf{read}(T)@l'.P \parallel l'::\langle et \rangle \xrightarrow{\mathrm{r}(l,et,l')} l::P\sigma \parallel l'::\langle et \rangle}$$

$$\frac{l' \notin L}{L \vdash l::\mathbf{newloc}(u).P \xrightarrow{\mathrm{n}(l,-,l')} L \cup \{l'\} \vdash l::P[l'/u] \parallel l'::\mathbf{nil}}$$

μ-Klaim Semantics (2)

(PAR)
$$\frac{L \vdash N_1 \xrightarrow{a} L' \vdash N_1'}{L \vdash N_1 \parallel N_2 \xrightarrow{a} L' \vdash N_1' \parallel N_2}$$

(STRUCT)
$$\frac{N \equiv N_1 \quad L \vdash N_1 \xrightarrow{a} L' \vdash N_2 \quad N_2 \equiv N'}{L \vdash N \xrightarrow{a} L' \vdash N'}$$

Dining philosophers in Klaim (1st attempt)

- There is a locality for each fork (lf_i)
  - The *i*-th fork is *free* if <*free*> is in the tuple space at locality lf_i
- There is a locality for each philosopher (phi_i)
  - at phi_i we have the process:

$$\begin{aligned} & Philosopher_i \triangleq \\ & \quad \#think.... \\ & \quad \mathbf{in}(free)@lf_i.\ \ \mathbf{in}(free)@lf_{|i+1|_n}. \\ & \quad \#eat.... \\ & \quad \mathbf{out}(free)@lf_i.\ \ \mathbf{out}(free)@lf_{|i+1|_n}. \\ & \quad Philosopher_i \end{aligned}$$

The dining philosophers…

- The complete system is then:

$$\begin{aligned} & lf_0 :: \langle free \rangle \parallel \\ & phi_0 :: Philosopher_0 \parallel \\ & lf_1 :: \langle free \rangle \parallel \\ & phi_1 :: Philosopher_1 \parallel \\ & lf_2 :: \langle free \rangle \parallel \\ & \ldots \\ & lf_{n-1} :: \langle free \rangle \parallel \\ & phi_{n-1} :: Philosopher_{n-1} \end{aligned}$$

Establishing Spatial Properties

- A system is often composed of identifiable subsystems.
  - "A message is sent from Alice to Bob."
  - "The protocol is split between two participants."
  - "A specific value (free) is present at a specific locality (lf_i) ."
- The above properties correspond to a spatial arrangement of processes in different places.
  - We look for a logic that allows us to specify and verify these properties.

A Modal Logic for Klaim

- A variant of HML with recursion where:
  - *State formulae* specify the resource distribution and availability.
  - Modal operators are indexed by *label predicates* that:
    - Describe the actual use of resources
    - Express *spatial properties of systems*

Formulae syntax

$$\begin{array}{rcl} \phi & ::= & \mathbf{tt} \\ & | & et@l \\ & | & \langle \mathcal{A} \rangle \phi \\ & | & \phi_1 \vee \phi_2 \\ & | & \neg \phi \\ & | & \kappa \\ & | & \nu\kappa.\phi \end{array}$$

State formula ($et@l$)

Label predicates ($\langle \mathcal{A} \rangle \phi$)

Derivable operators

$$[A]\phi \triangleq \neg\langle A \rangle \neg\phi$$

$$\phi_1 \wedge \phi_2 \triangleq \neg(\neg\phi_1 \vee \neg\phi_2)$$

$$\mu\kappa.\phi \triangleq \neg\nu\kappa.\neg\phi[\kappa/\neg\kappa]$$

Satisfaction relation

Let $\mathcal{A}$ be an action label, a be an action and σ a substitution:

- N |= <$\mathcal{A}$>ϕ if and only if:
  $\exists a, \sigma : (a, \sigma) \models \mathcal{A}. \exists N' : N \succ\!\xrightarrow{a} N'. N' \models \phi\{\sigma\}$
- N |= $\mathcal{A}$]ϕ if and only if:
  $\forall a, \sigma : (a, \sigma) \models \mathcal{A}. \forall N' : N \succ\!\xrightarrow{a} N'. N' \models \phi\{\sigma\}$
- N |= `et@l` , if and only if there exists a tuple `et` located at `l`.
- The relations for the other operators are the expected ones

Label predicates ($\mathcal{A}, \mathcal{A}_1, \mathcal{A}_2, \dots$)

- Basic labelpredicates:
  - $\mathtt{Src}(\widetilde{\ell})$ $\mathtt{Trg}(\widetilde{\ell})$
- Abstract action predicates:

$$\mathtt{O}(\ell_1, et, \ell_2) \quad \mathtt{E}(\ell_1, \mathtt{pp}, \ell_2) \quad \mathtt{N}(\ell_1, -, \ell_2) \quad \mathtt{I}(\ell_1, et, \ell_2) \quad \mathtt{R}(\ell_1, et, \ell_2)$$

- Operators:

$$\mathcal{A}_1 \cup \mathcal{A}_2 \quad \mathcal{A}_1 \cap \mathcal{A}_2 \quad \mathcal{A}_1 - \mathcal{A}_2 \quad \forall u.\mathcal{A} \quad \forall x.\mathcal{A}$$

Interpretation of label predicates

$\mathbb{L}[\![\circ]\!] = Lab$ (the set of all labels)

$\mathbb{L}[\![\mathtt{O}(\ell_1, et, \ell_2)]\!] = \{(\mathbf{o}(l_1, et, l_2); \emptyset)\}$

$\mathbb{L}[\![\mathtt{I}(\ell_1, et, \ell_2)]\!] = \{(\mathbf{i}(l_1, et, l_2); \emptyset)\}$

$\mathbb{L}[\![\mathtt{R}(\ell_1, et, \ell_2)]\!] = \{(\mathbf{r}(l_1, et, l_2); \emptyset)\}$

$\mathbb{L}[\![\mathtt{E}(\ell_1, \mathtt{pp}, \ell_2)]\!] = \{(\mathbf{e}(l_1, P, l_2); \emptyset) | P \in \mathbb{P}[\![\mathtt{pp}]\!]\}$

$\mathbb{L}[\![\mathtt{N}(\ell_1, -, \ell_2)]\!] = \{(\mathbf{n}(l_1, -, l_2); \emptyset)\}$

$\mathbb{L}[\![\mathcal{A}_1 \cup \mathcal{A}_1]\!] = \mathbb{L}[\![\mathcal{A}_1]\!] \cup \mathbb{L}[\![\mathcal{A}_2]\!]$

$\mathbb{L}[\![\mathcal{A}_1 \cap \mathcal{A}_1]\!] = \{(a; \sigma_1 \cdot \sigma_2) | (a; \sigma_1) \in \mathbb{L}[\![\mathcal{A}_1]\!], (a; \sigma_2) \mathbb{L}[\![\mathcal{A}_2]\!]\}$

$\mathbb{L}[\![\mathcal{A}_1 - \mathcal{A}_2]\!] = \{(a; \sigma) | (a; \sigma) \in \mathbb{L}[\![\mathcal{A}_1]\!], \forall \sigma' \ (a; \sigma') \notin \mathbb{L}[\![\mathcal{A}_2]\!]\}$

$\mathbb{L}[\![\forall u.\mathcal{A}]\!] = \bigcup_{l \in \mathcal{L}} \{(a; \sigma \cdot [l/u]) | (a; \sigma) \in \mathbb{L}[\![\mathcal{A}[l/u]]\!]\}$

$\mathbb{L}[\![\forall x.\mathcal{A}]\!] = \bigcup_{v \in Val} \{(a; \sigma \cdot [v/x]) | (a; \sigma) \in \mathbb{L}[\![\mathcal{A}[v/x]]\!]\}$

$\mathbb{L}[\![\mathtt{Src}(\widetilde{\ell})]\!] = \{(a; \emptyset) | \mathtt{source}(a) \in \{\widetilde{\ell}\}\}$

$\mathbb{L}[\![\mathtt{Trg}(\widetilde{\ell})]\!] = \{(a; \emptyset) | \mathtt{target}(a) \in \{\widetilde{\ell}\}\}$

Process and action predicates

Describe: *process intentions*

$$\mathtt{pp} \ ::= \ \mathtt{1_P} \mid \mathtt{ap} \rightarrow \mathtt{pp} \mid \mathtt{pp} \wedge \mathtt{pp}$$

$$\mathtt{ap} \ ::= \ \mathtt{o}(t)@\ell \mid \mathtt{i}(T)@\ell \mid \mathtt{r}(T)@\ell \mid \mathtt{e}(\mathtt{pp})@\ell \mid \mathtt{n}(u)$$

Two Simple Properties

- $\mathtt{i}(!u)@l \rightarrow \mathtt{e}(\mathtt{1_P})@u \rightarrow \mathtt{1_P}$
  is satisfied by all processes that read a locality name from locality l and spawn a process for evaluation at the read locality.
- $\forall u_1.\mathtt{E}(l_1, \mathtt{i}(!u)@u_1 \rightarrow \mathtt{e}(\mathtt{1_P})@u \rightarrow \mathtt{1_P}, l_2)$
  is satisfied by a labeled transition executed by a process located at l_1 that evaluates at l_2 a process that reads a locality name from a generic locality and spawns a process for evaluation at the read locality.

Abstract interpretation of processes

- Relation ($\overset{act}{\underset{\mathcal{V}}{\Longrightarrow}}$) specifies potential actions that are not preceded by actions that bind names in the set of variables $\mathcal{V}$
- It relies on a relation that permits silent transition of actions that do not bind variables of a given set ($\mathcal{V}$)
  - $act.P \rightarrow_{\mathcal{V}} P$ (if act does not bind variables in $\mathcal{V}$)
  - $P|Q \rightarrow_{\mathcal{V}} P \quad P|Q \rightarrow_{\mathcal{V}} Q \qquad P+Q \rightarrow_{\mathcal{V}} P \quad P+Q \rightarrow_{\mathcal{V}} Q$
  - If $A \triangleq P$ then $A \rightarrow_{\mathcal{V}} P$
- Relation $\overset{act}{\underset{\mathcal{V}}{\Longrightarrow}}$ is defined as follows:

$$act.P \overset{act}{\underset{\mathcal{V}}{\Longrightarrow}} P \qquad \frac{P \rightarrow_{\mathcal{V}} P' \quad P' \overset{act}{\underset{\mathcal{V}}{\Longrightarrow}} Q}{P \overset{act}{\underset{\mathcal{V}}{\Longrightarrow}} Q}$$

ac

Intepretation of process and action predicates

$\mathbb{P}[\![\mathtt{1_P}]\!] = Proc$ (the set of all processes)

$\mathbb{P}[\![\mathtt{ap} \rightarrow \mathtt{pp}]\!] = \{P | \exists act, P_1, P_2 :$
$P \equiv_\alpha P_1, P_1 \overset{act}{\underset{\mathtt{fv}(\mathtt{ap}\rightarrow\mathtt{pp})}{\Longrightarrow}} P_2, act \in \mathbb{A}[\![\mathtt{ap}]\!], P_2 \in \mathbb{P}[\![\mathtt{pp}]\!]\}$

$\mathbb{P}[\![\mathtt{pp_1} \wedge \mathtt{pp_2}]\!] = \mathbb{P}[\![\mathtt{pp_1}]\!] \cap \mathbb{P}[\![\mathtt{pp_2}]\!]$

$\mathbb{A}[\![\mathtt{o}(t)@\ell]\!] = \{\mathbf{out}(t)@\ell\}$

$\mathbb{A}[\![\mathtt{i}(T)@\ell]\!] = \{\mathbf{in}(T)@\ell\} \qquad \mathbb{A}[\![\mathtt{r}(T)@\ell]\!] = \{\mathbf{read}(T)@\ell\}$

$\mathbb{A}[\![\mathtt{e}(\mathtt{pp})@\ell]\!] = \{\mathbf{eval}(Q)@\ell \mid Q \in \mathbb{P}[\![\mathtt{pp}]\!]\}$

$\mathbb{A}[\![\mathtt{n}(u)]\!] = \{(\mathbf{newloc}(u) | u \in VLoc\}$

Interpretation of formulae

$\mathbb{M}[\![\mathtt{true}]\!]\epsilon\sigma = Net$

$\mathbb{M}[\![\kappa]\!]\epsilon\sigma = \epsilon(\kappa)\sigma$

$\mathbb{M}[\![\langle t\rangle @l]\!]\epsilon\sigma = \{N | N \equiv N_1 \parallel l :: \langle \mathcal{T}[\![\, t\,]\!]\sigma\rangle\}$

$\mathbb{M}[\![\langle \mathcal{A}\rangle\phi]\!]\epsilon\sigma = \{N | \exists a, \sigma', N'.\ N \overset{a}{\rightarrowtail} N', (a, \sigma') \in \mathbb{L}[\![\mathcal{A}\{\sigma\}]\!], N' \in \mathbb{M}[\![\phi]\!]\epsilon\sigma' \cdot \sigma\}$

$\mathbb{M}[\![\phi_1 \vee \phi_2]\!]\epsilon\sigma = \mathbb{M}[\![\phi_1]\!]\epsilon\sigma \cup \mathbb{M}[\![\phi_2]\!]\epsilon\sigma$

$\mathbb{M}[\![\neg\phi]\!]\epsilon\sigma = Net - \mathbb{M}[\![\phi]\!]\epsilon\sigma$

$\mathbb{M}[\![\nu\kappa.\phi]\!]\epsilon\sigma = \bigcup\{g | g \subseteq f^{\phi}_{\kappa,\epsilon}(g)\}$ where $f^{\phi}_{\kappa,\epsilon}(g) = \mathbb{M}[\![\phi]\!]\epsilon \cdot [\kappa \mapsto g]$

Domains...

- *VLog* is the set of logical variables
- *Subst* is the set of substitutions
- *Nets* is the set of nets
- *Env (logical environments)* is

$$Env \subseteq [VLog \rightarrow Subst \rightarrow 2^{Net}]$$

Interpretation functions...

- Formulae: $\mathbb{M}[\![\cdot]\!] : \Phi \rightarrow Env \rightarrow Subst \rightarrow 2^{Net}$
- Label predicates: $LPred \rightarrow Lab \times Subst$
- Process predicates: $PPred \rightarrow Proc$

Properties of dining philosophers

- Deadlock freedom:

$$liveness = \nu\kappa.\langle\circ\rangle\mathbf{tt} \wedge [\circ]\kappa$$

- Philosopher at ph_i accesses only lf_i and $lf_{|i+1|n}$:

$$wellbehaviour = \neg\mu\kappa.\langle \mathtt{Src}(\{phi_i\})\rangle - \mathtt{Trg}(\{lf_i, lf_{|i+1|_n}\})\mathbf{tt} \vee \langle\circ\rangle\kappa$$

- The i[th] philosopher does not eat:

$$\begin{array}{lcl} noeat & = & [\mathtt{I}(phi_i, free, lf_i)]\mathbf{ff}\vee \\ & & \neg(\langle \mathtt{I}(phi_i, free, lf_i)\rangle \mu\kappa.\langle \mathtt{I}(phi_i, free, lf|i+1|_n)\rangle\mathbf{tt}\vee \\ & & \quad \langle \circ - \mathtt{I}(phi_i, free, lf|i+1|_n)\rangle\kappa) \end{array}$$

- The i[th] philosopher eventually eat:

$$nostarvation = \nu\kappa.(\neg noeat) \wedge [\circ]\kappa$$

Properties satisfaction - 1

The specification of dining philosophers in Klaim (1[st] attempt)

- does satisfy
  - wellbehavior
- does not satisfy
  - Liveness
  - noeat
  - nostarvation

A deadlock-free implementation (Philosophers)

$$\begin{array}{lll} Philosopher_i \stackrel{\triangle}{=} & & \\ \quad \mathbf{in}(''OK'')@lf|i+1|_n & & + \\ \quad \mathbf{out}(''KO'')@lf_{|i+1|_n} & & \quad (\\ \quad (& & \quad \mathbf{in}(''KO'')@lf_i. \\ \quad\quad (& & \quad \mathbf{out}(''KO'')@lf_i. \\ \quad\quad \mathbf{in}(''OK'')@ls_i. & & \quad \mathbf{in}(''KO'')@lf_{|i+1|_n}. \\ \quad\quad \mathbf{in}(free)@lf_i. & & \quad \mathbf{out}(''OK'')@lf_{|i+1|_n}. \\ \quad\quad \mathbf{in}(free)@lf_{|i+1|_n}. & l & \quad Philosopher_i \\ \quad\quad \mathbf{out}(free)@lf_i. & & \quad) \\ \quad\quad \mathbf{out}(free)@lf_{|i+1|_n}. & &) \\ \quad\quad \mathbf{in}(''KO'')@lf_{|i+1|_n}. & & \\ \quad\quad \mathbf{out}(''OK'')@lf_{|i+1|_n}. & & \\ \quad\quad \mathbf{out}(''OK'')@lf_i. & & \\ \quad\quad Philosopher_i & & \\ \quad\quad) & & \end{array}$$

A deadlock-free implementation (full system)

$$\begin{array}{l} fork_0 :: \langle fork_0\rangle \parallel fork_0 :: \langle ``OK''\rangle \parallel \\ phi_0 :: Philosopher_0 \parallel \\ fork_1 :: \langle fork_1\rangle \parallel fork_1 :: \langle ``OK''\rangle \parallel \\ phi_1 :: Philosopher_1 \parallel \\ fork_2 :: \langle fork_2\rangle \parallel fork_2 :: \langle ``OK''\rangle \parallel \\ \ldots \\ phi_{n-1} :: Philosopher_{n-1} \parallel \\ fork_{n-1} :: \langle fork_{n-1}\rangle \parallel fork_{n-1} :: \langle ``OK''\rangle \end{array}$$

Property satisfaction - 2

The Dining philosophers in Klaim (2[nd] attempt) specification:

- does satisfy
  - Well behavior
  - liveness
- does not satisfy
  - eating
  - nostarvation

Dealing with Open Systems

- Closed systems:
  - Complete representation of all system components (standard practice in Klaim)
- Open systems:
  - Partial knowledge of systems components (good practice in WAN)
- Context dependent systems:
  - Abstract context specification plus concrete specification of some components

A Proposal for Open Systems

- Specify known components of a system with Klaim;
- Partially specify contexts (the rest of the systems) with an ad-hoc formalism;
- Specify system properties with Klaim logics and related tools.

Context specification

- Available resources
  - The contexts has a locality named *l*: $@l$
  - tuple *et* is located at *l*: $et@l$
- A process located at *l* performs action *act*: $(l : act) \rightarrow p$
- The contexts has aa node at an unknown locality: $\lambda u.n$
- There are as many nodes as necessary satisfying n: $!n$

Contexts and OpenNets

- Contexts Syntax

$$n \ ::= \ p \ |\lambda u.n \ |n_1 \otimes n_2 \ |n_1 \oplus n_2 \ |!n$$

$$p \ ::= \ \mathbf{0} \ |(l : act) \rightarrow p \ |t@\ell \ |@\ell \ |f \ |p \otimes p \ |p \oplus p$$

- Open Nets

$$N \ ::= \ l :: P \ |N_1 \parallel N_2 \ |n\,[N]$$

Structural congruence for contexts

$$n_1\,[N_1] \parallel n_2\,[N_2] \equiv n_1 \otimes n_2\,[N_1 \parallel N_2] \qquad n_1\,[n_2\,[N]] \equiv n_1 \otimes n_2\,[N]$$

$$@l\,[N] \equiv \mathbf{0}\,[N \parallel l :: \mathbf{nil}] \qquad \mathbf{0} \otimes n\,[N] \equiv n\,[N]$$

$$!n\,[N] \equiv n \otimes !n\,[N] \qquad \langle et \rangle @l\,[N] \equiv \mathbf{0}\,[N \parallel l :: \langle et \rangle]$$

Operational semantics for contexts

$$(\textsc{Out}) \quad \frac{\mathcal{T}[\![\, t \,]\!] = et}{l_1 : \mathbf{out}(t)@l_2 \rightarrow p\,[l_2 :: Q] \succ\!\xrightarrow{\mathbf{o}(l, et, l')} p\,[l_2 :: Ql_2 :: \langle et \rangle]}$$

$$(\textsc{Eval}) \quad l_1 : \mathbf{eval}(Q)@l' \rightarrow p\,[l_2 :: P] \parallel\!\succ\!\xrightarrow{\mathbf{e}(l_1, Q, l_2)} p\,[l_2 :: P_2|Q]$$

$$(\textsc{In}) \quad \frac{match(\mathcal{T}[\![\, T \,]\!], et) = \sigma}{l_1 : \mathbf{in}(l)@T_2 \rightarrow p\,[l_2 :: \langle et \rangle] \succ\!\xrightarrow{\mathbf{i}(l_1, et, l_2)} p\,[l_2 :: \mathbf{nil}]}$$

$$(\textsc{Read}) \quad \frac{match(\mathcal{T}[\![\, T \,]\!], et) = \sigma}{l_1 :: \mathbf{read}(T)@l_2 \rightarrow p\,[l_2 :: \langle et \rangle] \succ\!\xrightarrow{\mathbf{r}(l_1, et, l_2)} p\,[l_2 :: \langle et \rangle]}$$

Localized formulae

- *Modalities localized at S*
- a set of localities (κ not in ϕ) :

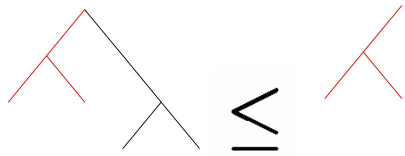
$$\langle\langle A,S\rangle\rangle\phi \triangleq \mu\kappa.\langle A\rangle\phi \vee \langle \circ - (\mathtt{Src}(S)\cup\mathtt{Trg}(S))\rangle\kappa$$

$$[[A,S]]\phi \triangleq \neg\langle\langle A,S\rangle\rangle\neg\phi = \nu\kappa.[A]\phi \wedge [\circ - (\mathtt{Src}(S)\cup\mathtt{Trg}(S))]\kappa$$

- A *formula* is *localized at S* if it contains only modalities localized at S

Behavioural relations ($\sqsubseteq^+_S$ $\simeq_S$)

- Based on a preorder induced by a sort of inclusion of computation trees:



- A *localized* version of the relations is introduced by considering only labels *over* given sets of localities

Equivalence and localized formulae

- For every formula ϕ positive and localized at S we have:

$$N_1 \sqsubseteq^+_S N_2 \text{ and } N_2 \models \phi \Longrightarrow N_1 \models \phi$$

- For every ϕ localized at S:

$$N_1 \simeq_S N_2 \text{ and } N_1 \models \phi \Longleftrightarrow N_2 \models \phi$$

Context approximation

- A net *N* is the ***concretion*** of a context $\lambda\tilde{u}.p$ within an open net $n_1[N_1]$ if and only if there exist a set of localities $\{\tilde{l}\}$ in *N* such that

$$n_1 \otimes p[\tilde{l}/\tilde{u}]\,[N_1] \sqsubseteq^+_{sites(N_1)} n_1\,[N_1 \parallel N]$$

- ***Concretion*** preserves the (un)satisfiability of *positive formulae localized at localities in* N_1

$$n_1 \otimes p[\tilde{l}/\tilde{u}][N] \models \neg\phi \Rightarrow n_1[N_1 \parallel N] \models \neg\phi$$

Context Agreement

- *N agrees* with a context $\lambda\tilde{u}.p$ within a net $n_1[N_1]$ if and only if there exist a set of localities $\{\tilde{l}\}$ in *N* such that

$$n_1 \otimes p[\tilde{l}/\tilde{u}]\,[N_1] \simeq_{sites(N_1)} n_1\,[N_1 \parallel N]$$

- *Agreement* preserves the satisfiability of *formulae localized at localities in* N_1

A pair of results

$$\frac{n_1 \wedge n\,[N_1] \models \neg\phi \qquad n_1 \wedge n\,[N_1] \sqsubseteq_{sites(N_1)} n_1\,[N_1 \parallel N]}{n_1\,[N_1 \parallel N] \models \neg\phi}$$

$$\frac{n_1 \wedge n\,[N_1] \models \phi \qquad n_1 \wedge n\,[N_1] \simeq_{sites(N_1)} n_1\,[N_1 \parallel N]}{n_1\,[N_1 \parallel N] \models \phi}$$

Context for the philosophers

- First implementation:

$$\begin{array}{lll} f & \triangleq & u_1 : \mathbf{in}(free)@lf_1 \rightarrow u_1 : \mathbf{out}(free)@lf_1 \rightarrow f \\ & \oplus & u_2 : \mathbf{in}(free)@lf_2 \rightarrow u_2 : \mathbf{out}(free)@lf_2 \rightarrow f \\ & \oplus & u_2 : \mathbf{in}(free)@lf_2 \rightarrow u_1 : \mathbf{in}(free)@lf_1 \rightarrow \\ & & \quad u_2 : \mathbf{out}(free)@lf_2 \rightarrow u_1 : \mathbf{out}(free)@lf_1 \rightarrow f \\ & \oplus & u_1 : \mathbf{in}(free)@lf_1 \rightarrow u_2 : \mathbf{in}(free)@lf_2 \rightarrow \\ & & \quad u_1 : \mathbf{out}(free)@lf_1 \rightarrow u_2 : \mathbf{out}(free)@lf_2 \rightarrow f \end{array}$$

- The *abstract* system:

$$f\,[lf_1 :: \langle free \rangle \parallel phi_1 :: Philosopher_1 \parallel lf_2 :: \langle free \rangle]$$

Refinement of the context

$$lf_0 :: \langle free \rangle \parallel phi_0 :: Philosopher_0 \parallel phi_2 :: Philosopher_2$$

agrees *f* with respect to

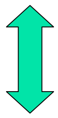
$$lf_1 :: \langle free \rangle \parallel phi_1 :: Philosopher_1 \parallel lf_2 :: \langle free \rangle$$

indeed

$$\begin{array}{l} f\,[lf_1 :: \langle free \rangle \parallel phy_1 :: Philosopher_1 \parallel lf_2 :: \langle free \rangle] \\ \simeq_{\{phi_1, lf_1, lf_2\}} \\ lf_1 :: \langle free \rangle \parallel phy_1 :: Philosopher_1 \parallel lf_2 :: \langle free \rangle \\ \parallel lf_0 :: \langle free \rangle \parallel phi_0 :: Philosopher_0 \parallel phi_2 :: Philosopher_2 \end{array}$$

Refinement of the context (2)

$$f\,[lf_1 :: \langle free \rangle \parallel phi_1 :: Philosofer_1 \parallel lf_2 :: \langle free \rangle] \models \phi$$



$$\begin{array}{l} lf_1 :: \langle free \rangle \parallel phi_1 :: Philosofer_1 \parallel lf_2 :: \langle free \rangle \\ \parallel lf_0 :: \langle free \rangle \parallel phi_0 :: Philosopher_0 \parallel phi_2 :: Philosofer_2 \end{array} \models \phi$$

Conclusions...

- Klaim, contexts and the logics are a *methodology* for programming and verifying WAN applications
- Components in the context can be progressively implemented
- Properties verified at one stage can be preserved during the *concretion* of the system

Klaim site

http://music.dsi.unifi.it

You will find

- A few papers
- Implementations
  - A prototype Model Checker
  - X-Klaim
  - Klava
  - KryptoKlava

Liveness Respecting Equivalences for Concurrent Systems

Rob van Glabbeek

INRIA

Sophia Antipolis

1.

LIVENESS + INTERLEAVING ⇒ FAIRNESS

I WILL

- Define LIVENESS, FAIRNESS, INTERLEAVING
- Substantiate controversial thesis above
- Give overview of liveness respecting interleaving semantics
  - and evaluate their suitability
- Discuss liveness respecting non-interleaving semantics
- Sketch a way for encoding non-global fairness.

Open problems and future directions

2.

LIVENESS

Safety properties

something bad won't happen

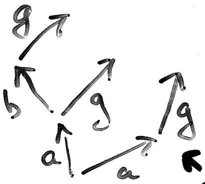
Liveness properties

something good will eventually happen

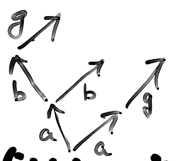
"along all possible execution paths the system will eventually do a g".

$\forall F g$ $\qquad g \in Act$

an ACTL* formula



$\models \forall F g$



$\not\models \forall F g$

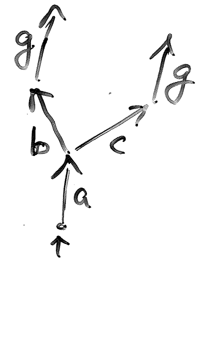
"better be" distinguished in liveness respecting semantics

3.

Progress Assumption

- actions take only a finite amount of time
- when an action is enabled within a finite amount of time some action will happen.

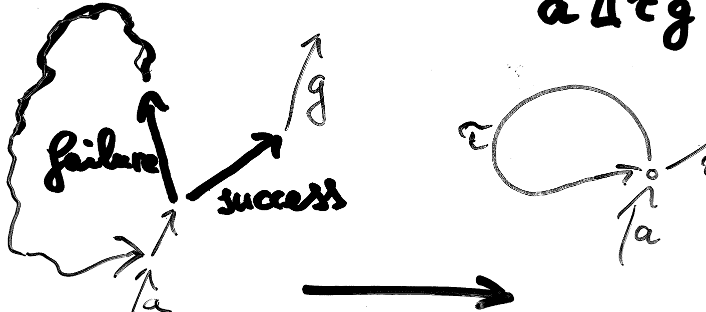
needed for validity of $\forall F g$



4.

FAIRNESS
(global)

$a\Delta\tau g$

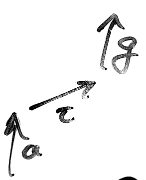


ABSTRACTION

Δ (delay) prefixes a τ-loop for a process

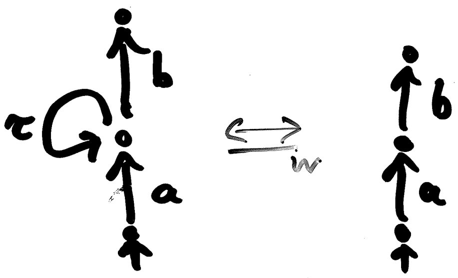
KOOMEN'S FAIR ABSTRACTION RULE (KFAR) || FAIRNESS

$$a \Delta x = a x$$



$a\tau g$

weak bisimulation equivalence $\leftrightarrows_w$ as originally devellaped by Milner satisfies KFAR

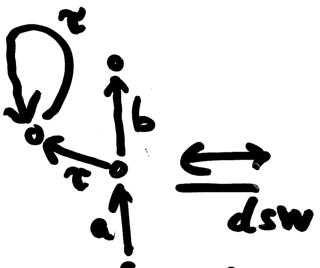


$\leftrightarrows_w$
- is a fair equivalence
- abstracts from divergence.

↑ falls short in modelling divergence precisely.

Therefore, Milner + Walker proposed divergence sensitive variant of $\leftrightarrows_w$ $\leftrightarrows_{dsw}$

qFAL qFA? qFA



UNFAIRNESS FAIRNESS

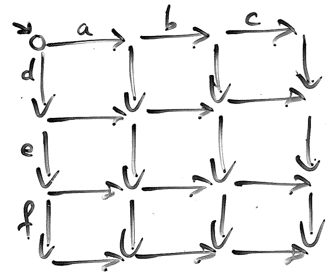
testing semantics [DH]
failures seman. [HBR]
refusal testing [PR]
etc. etc. all satisfy UNFAIRNESS
[BK] exploit fairness for protocol verification

INTERLEAVING SEMANTICS.

Interleaving operator (quasi parallel composition) // of Bekič

Two processes scheduled on one processor by arbitrary interleaving of their actions

abc//def :



$$a//b = ab+ba$$

Is this interleaving semantics ???

NO

Sequential processes

perform only one action at a time

Parallel (Concurrent) processes

perform several actions at a time

Concurrency ⇒ parallel composition

Parallel composition operator $\|$
(without communication)

Two processes operating independently.
(one here, one on the moon)

INTERLEAVING SEMANTICS
is about the semantics of the parallel composition.

CONCURRENCY → INTERLEAVING
CONCURRENCY → NON-INTERL.

No parallel composition ⇒ no interleaving semantics

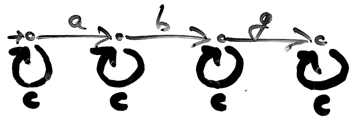
9.

Definition of interleaving semantics

$$x \| y = x /\!/ y$$

defining equation
(semantic equivalence)

$abg /\!/ c^\infty$



does not satisfy $\forall\exists g$
(unless assuming fairness)

$abg \| c^\infty \models \forall\exists g$

g will happen eventually

10.

Way out:
use fair merge

$abg /\!/_F c^\infty \models \forall\exists g$

Still

LIVENESS + INTERLEAVING ⇒ FAIRNESS

LIVENESS RESPECTING INTERLEAVING SEMANT.

- COMPOSITIONAL FOR ACP / CCS / CSP
- ALLOW EQUATION REASONING
  Guarded Recursive equations like
  $X = aX + b$
  should have unique solutions

(Recursive Specification Principle (RSP))

11.

12.

$a(\tau(x+y)+y) = a(x+y)$

BRANCHING BISIMUL. [vGW 89]

$\tau x + x = \tau x$

DELAY BIS. [Mil 81]

$a(\tau x+y) = a(\tau x+y) + ax$

WEAK BIS. [Mil 80, HM 80, Park 81]

$\tau(\tau x+y) = \tau x+y$

COUPLED SIM. [PS 92]

$ax+ay = a(\tau x+\tau y)$

CONTRASIMULATION [vG 86]

$x \sqsubseteq \tau x$

IMPOSSIBLE FUTURES [VM 00] (+vG)

$\tau(ax+z)+ay = \tau(ax+ay+z)$

FAIR TESTING [BRV 95]

differs from [DH]-must-testing only for infinite processes.

⇑ fully abstract

(but does not satisfy RSP).

13.

REPRESENTING LIVENESS WITHOUT FAIRNESS

minimum requirement:

COMPLETED & INFINITE TRACES

$\forall \exists g$ can be checked for each of those traces

COMPOSITIONALITY for CCSP/ACP +R.S.P.

⇓ something like

ST-failure trace semantics (interval refusal trace sem.)

Real-time consistency [vGV 87]

⇓

respect liveness

14.

REAL-TIME CONSISTENCY:

$p \approx q$

⇓

for each valuation of durations to actions:

real-time traces (p) = real-time tr. (q)

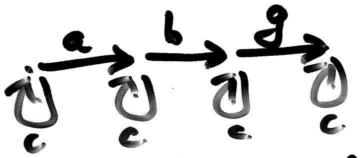
$a\ b\ g \parallel c^\infty$

↑ ↑ ↑ ↑

2 3 4 1

---

g happens at time 5 (starts)

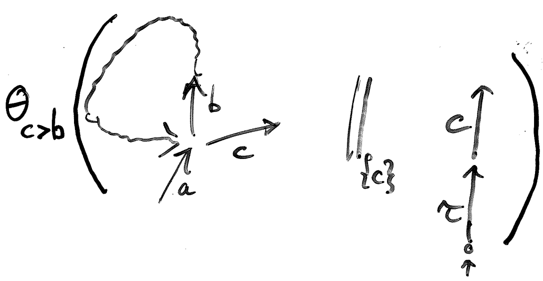


g can happen at time 5, 6, 7, ...

thus $abg \parallel c^\infty \neq abg \parallel c^\infty$

15.

Encoding non-global fairness into "justness" and priorities



16.

ST failure trace semantics
may be coarsest real-time
consistent semantics(?).

Also coarsest liveness-resp.
semantics that is compositional
and satisfies RSP. ?
... future work.

PAFAS ?
also · liveness respecting
· compositional
· RSP.

→ process algebra
also incorporating
non-global fairness

17.

technische universiteit eindhoven

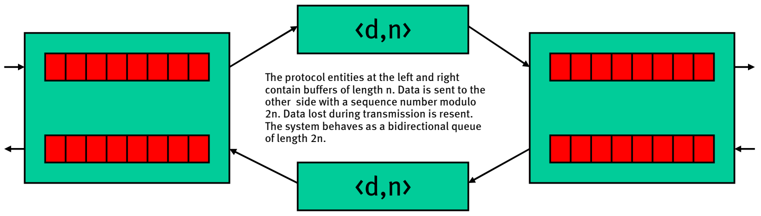
The need for proof methodologies

Jan Friso Groote

technische universiteit eindhoven

1991-2003: proof of the sliding window protocol, inspired by Tanenbaum

Contributors: Jan Friso Groote, Wan Fokkink, Jaco van de Pol, Jun Pang, Bahara Badban



technische universiteit eindhoven

Theorem about the sliding window protocol:

$$\tau_I(\delta_H(sendlr(n)||K||L||sendrl(n)))=BidirectQueue(2n)$$

Previous work:

- Middeldorp
- Groeneveld
- Brunekreef
- Schoone
- Snepscheut
- Bezem-Groote
- CSP guys
- Chkliaev-Hooman
- de Backer-Hoogerwoord
- Roeckl-Esparza

technische universiteit eindhoven

Lessons learned:

1. The sliding window protocol by Tanenbaum contained a livelock. Supports 100% rule.
2. Nice external behaviour is an important design consideration. Tanenbaum's SWP was unusable without extra buffering.
3. It is impossible to give a proof of this SWP in 'classical' process algebra nor directly in any related semantical realm.

New proof methodologies are required.

Finding new proof methodologies has been one of my concerns, leading to the techniques on the next pages.

technische universiteit eindhoven

Processes must have data:

μCRL=ACP+eq.data+four extensions below

Data in processes — Conditionals

$$S(n{:}Nat) = read(n).S(n)\lhd n>0 \rhd \delta + \Sigma_{m:Nat}\, write(m).S(m)$$

Sum operator — Data in actions

technische universiteit eindhoven

Note: all mCRL processes can be translated to linear processes
Ph.D thesis Usenko 2003

Linear Process Equation/Operator:

State — Action — Effect

$$X(d{:}D)= \Sigma_{i\in I} \Sigma_{e_i:E_i}\, a(f_i(d,e_i))\, X(g_i(d,e_i)) \lhd c_i(d,e_i) \rhd \delta$$

Condition

Linear Process Equations form the core of the μCRL toolset

$$\Psi=\lambda X \lambda\, d{:}D. \Sigma_{i\in I} \Sigma_{e_i:E_i}\, a(f_i(d,e_i))\, X(g_i(d,e_i)) \quad c_i(d,e_i) \quad \delta$$

Concrete Invariant Corollary:

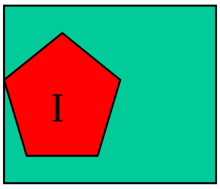
Invariant: $I(d) \wedge c(d,e_i) \Rightarrow I(g_i(d,e_i))$

$$I(d) \Rightarrow p(d)=\psi pd$$

$$I(d) \Rightarrow q(d)=\psi qd$$

$$I(d) \Rightarrow p(d)=q(d)$$

Structure of an implementation

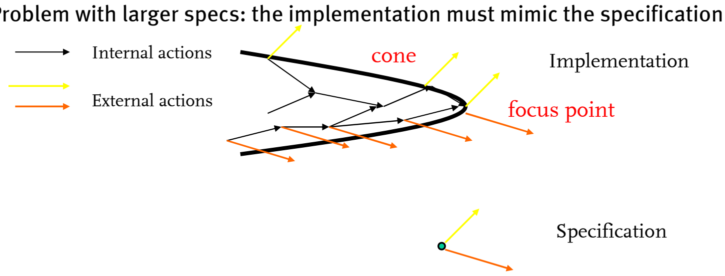


Cones and Foci theorem

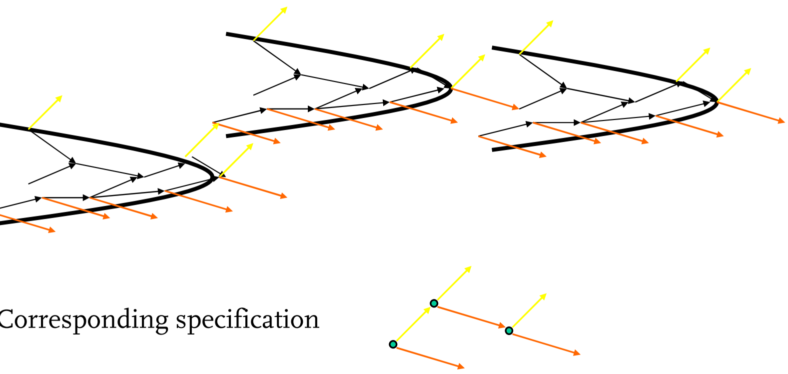
Implementation: $X(d{:}D)=\Sigma_{i\in I}\,\Sigma_{e_i:E_i}\, a(f_i(d,e_i))\, X(g_i(d,e_i)) \lhd c_i(d,e_i) \rhd \delta$

Specification: $X(d{:}D)=\Sigma_{i\in I'}\,\Sigma_{e_i:E_i'}\, a(f_i'(d,e_i))\, X(g_i(d,e_i)) \lhd c_i(d,e_i) \rhd \delta$

Problem with larger specs: the implementation must mimic the specification:

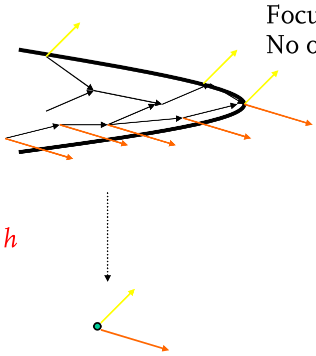


‘The implementation is a set of cones’



Corresponding specification

Focus point:
No outgoing τ’s possible



State mapping: *h*

In order to prove implementation equal to the specification matching criteria must be proven (possibly using an invariant).

This formula gives an impression of how the matching criteria look like as formulas.

1. There are no infinite τ-paths (τ-convergence, original paper had a relaxing theorem).
2. τ’s preserve state mapping: $c(d,e_i) \Rightarrow h(d)=h(g_i(d,e_i))$.
3. Visible actions in the implementation must be mimicked in the specification.
4. Visible actions in the specification must be mimicked in the focus point of the implementation!!!!!
5. Parameters of actions must match.
6. Corresponding end points must match.

All 6 matching criteria can straightforwardly be expressed in terms of LPO’s

Using the cones and foci technique most protocols can be shown correct:

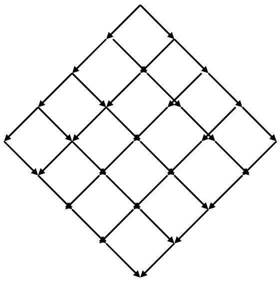
1. One bit sliding window.
2. Distributed summing.
3. Leader election in firewire network.
4. Slip protocol
5. ABP, CABP, Queues, etc.
6. Leader election in a ring (Dolev, Klawe and Rodeh)

Subsequent work:

1. Vaandrager/Griffioen: normed bisimulations
2. van der Zwaag: cones and foci for time.
3. Fokkink/Pang: relaxing convergence requirements.

technische universiteit eindhoven

Use confluence in verifications: a common pattern in behaviour is:

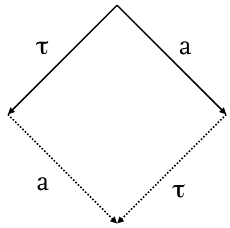


technische universiteit eindhoven

A transition system is called τ-confluent iff for all states and actions:

if:

then:

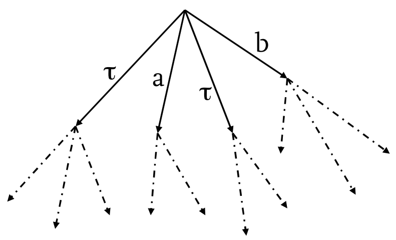


τ-confluence and τ-convergence are easy to prove on LPO's. The μCRL toolset contains tools that can prove τ-confluence automatically.

Recall τ-convergent means no infinite τ-paths.

technische universiteit eindhoven

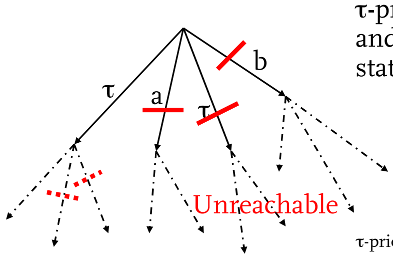
τ-confluence and τ-convergence allows τ-prioritisation:



Is branching bisimilar to:

technische universiteit eindhoven

τ-confluence and τ-convergence allows τ-prioritisation:

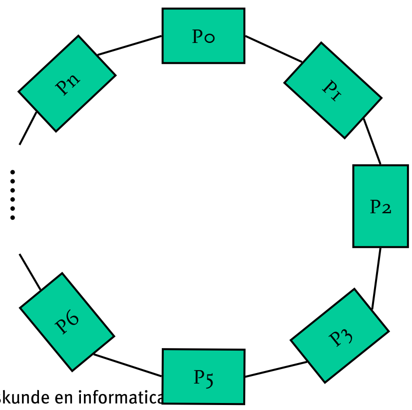


It is easy to apply τ-priorisation on LPOs and to use it when generating state spaces (Blom and van de Pol)

τ-priorisation has not been used for larger manual verifications, although the Dolev, Klawe and Rodeh leader election was the inspiration and it could have been applied there.

technische universiteit eindhoven

n+1 parallel processes



technische universiteit eindhoven

Single process:

$$P(k{:}Nat,d{:}D)=\sum_{i\in I}\sum_{e_i:E_i} a_i(f_i(k,d,e_i))\ P(k,g_i(k,d,e_i)) \lhd c_i(k,d,e_i) \rhd \delta$$

System of n+1 parallel processes:

$$Sys(n{:}Nat,dt{:}Nat\to D)=$$
$$P(0,dt[0]) \lhd n=0 \rhd P(n,dt[n]) \| Sys(n-1,dt)$$

System of n+1 parallel processes as a linear process equation:

$$Y(n{:}Nat,dt{:}Nat\to D)=$$
$$\sum_{i\in I}\sum_{k:Nat}\sum_{e_i:E_i} a_i(f_i(k,dt[k],e_i))\ Y(n,dt[k]:=g_i(k,dt[k],e_i))$$
$$\lhd c_i(k,dt[k],e_i) \wedge k\leq n \rhd \delta +$$

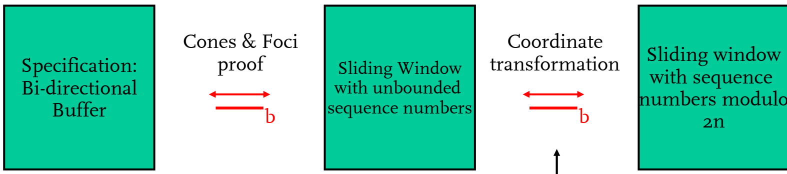
similar summand for handshaking communication.

Is this sufficient for the sliding window protocol?

- No, because lack of mental capacities to provide the proof.
- Yes, meta-theorems say that after having found the invariants and cones and foci should have been sufficient (Jaco van de Pol)

I hardly believe this: we weren't that stupid

Coordinate transformations (already applied by Anneke Schoone):



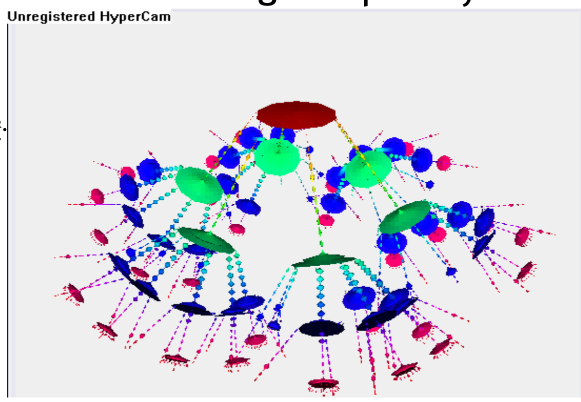
Actually a simple application of CL-RSP

Future work and open problems:

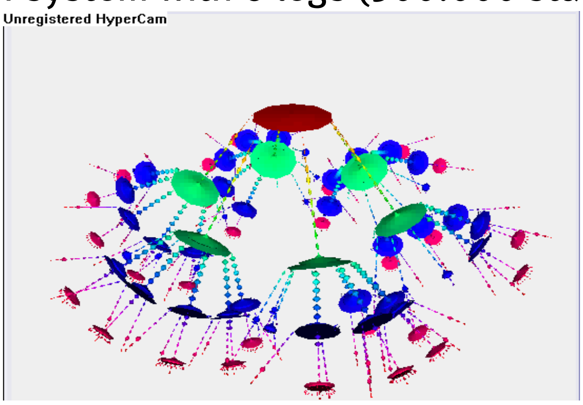
- Make these techniques suitable for distributed systems and protocol design. Design new algorithms proven correct with these techniques.
- Extension to weaker equivalences: Bloom's register, prophecy variables.
- Handle data types using 'common sense', instead of via abstract data types. Develop meta knowledge on data.

And now something completely different?

If time allows????

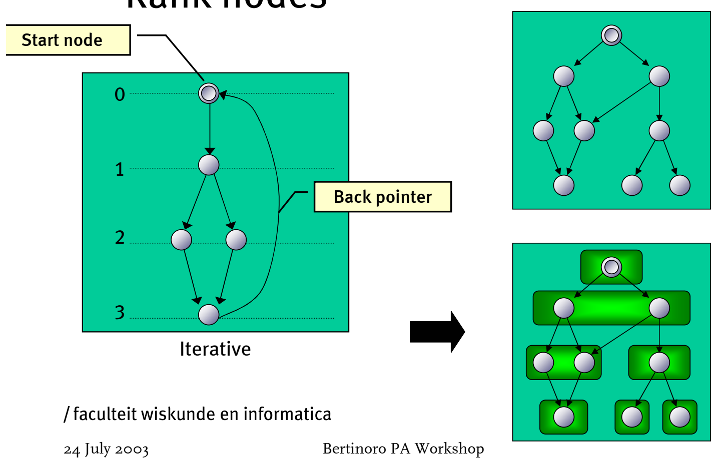


Hef system with 6 legs (500.000 states)



Rank nodes

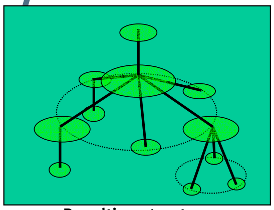
Cluster nodes

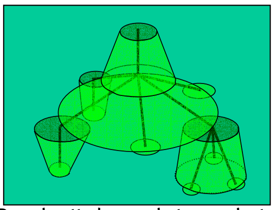


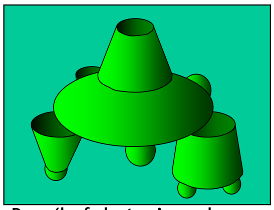
Phd of Frank van Ham

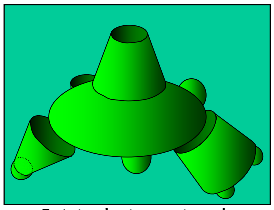
TU/e technische universiteit eindhoven

Draw tree as 3D object


Resulting structure


Draw knotted cones between clusters


Draw 'leaf-clusters' as spheres.

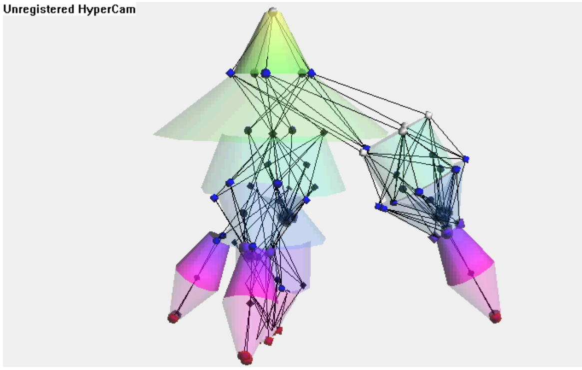

Rotate clusters outward.

/ faculteit wiskunde en informatica

24 July 2003 Bertinoro PA Workshop 25

TU/e technische universiteit eindhoven

Display without backpointers

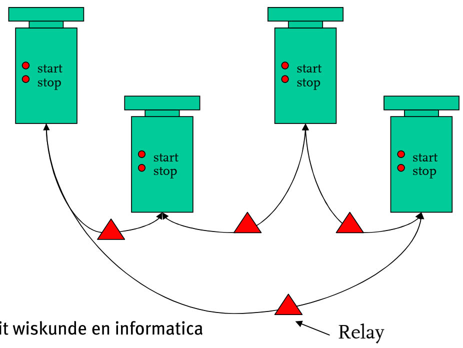


/ faculteit wiskunde en informatica

24 July 2003 Bertinoro PA Workshop 26

TU/e technische universiteit eindhoven

A larger example: a modular hef system

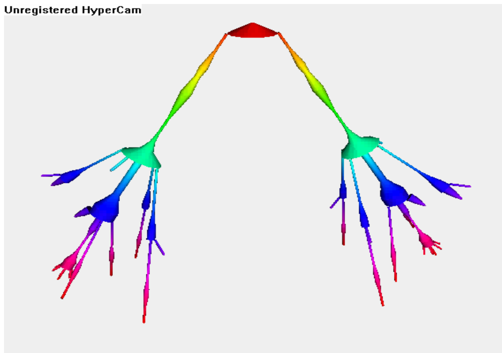


/ faculteit wiskunde en informatica

24 July 2003 Bertinoro PA Workshop 27

TU/e technische universiteit eindhoven

The hef system with 2 legs

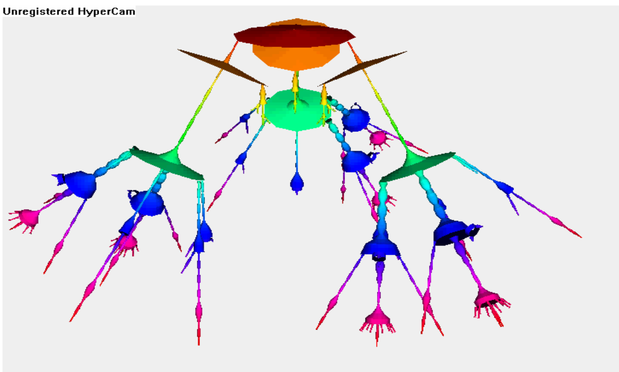


/ faculteit wiskunde en informatica

24 July 2003 Bertinoro PA Workshop 28

TU/e technische universiteit eindhoven

Lift system with three legs

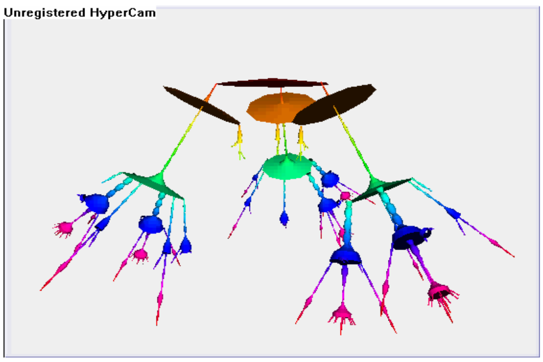


/ faculteit wiskunde en informatica

24 July 2003 Bertinoro PA Workshop 29

TU/e technische universiteit eindhoven

Zoom into the deadlock

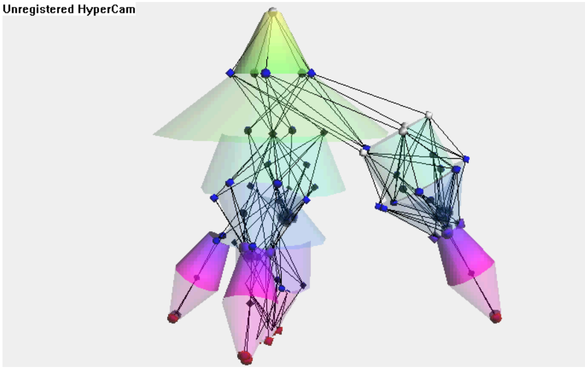


/ faculteit wiskunde en informatica

24 July 2003 Bertinoro PA Workshop 30

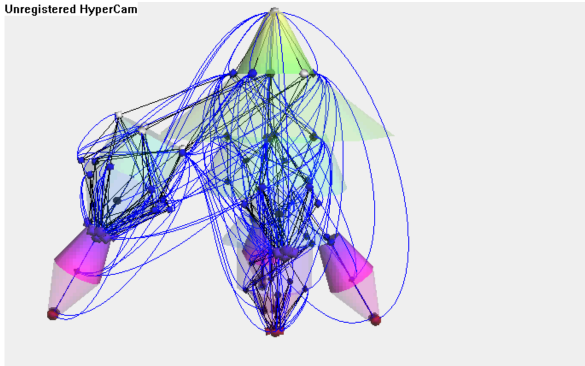
technische universiteit eindhoven

Display without backpointers

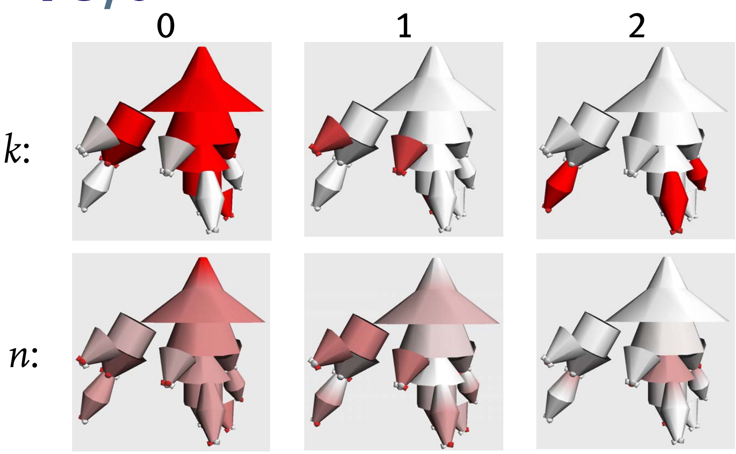


technische universiteit eindhoven

Display with back pointers

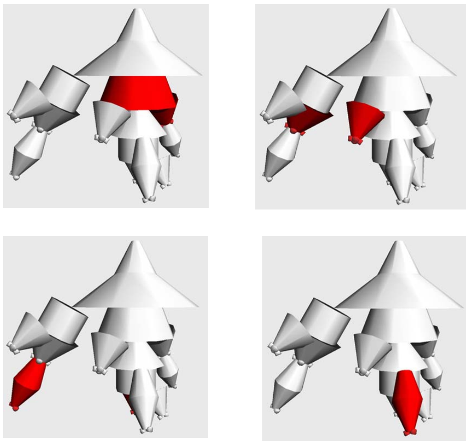


technische universiteit eindhoven



Color on the values of variables

technische universiteit eindhoven



Color based on inform(*n*), *n*=0,1,2,3,4

Axiomatising Divergence

Holger Hermanns
Saarland University, D
&
University of Twente, NL

joint work with
Markus Lohrey (Uni Stuttgart, D)
Pedro R. D'Argenio (Uni Cordoba, ARG)

PA-Any problems? July 23 2003

Thanks for inviting me to the Amsterdam day of this workshop

2

Overview

- Setting the stage
- The divergence sensitive spectrum
- The axiomatisation result we can offer
- A bit of the proof strategy
- The open problems
- Lunch

3

Setting the stage

- Assume you have two polynoms

$P_1(x)=x^4+3x^2-(2x-1)x$ and $P_2(x)=(x-1)^3-3x^3+x^2(x^2-1)$

and you need to know whether they are the same.

- How do you proceed?

I guess you use equational reasoning based on axioms in the ring of polynoms to rewrite P_1 and P_2 into a normal form.

$P_1(x)= x^4-x^2-x$ and $P_2(x)= x^4-x^2-3x^3+(x-1)(x^2-2x+1)$
$= x^4-4x^2-2x^3-x-1$

- How does Maple/Mathematica etc. proceed?

Well, take $P_1(x)-P_2(x)$, and uses an efficient test whether this is the zero-polynom, by just throwing in 5 different values for x.

4

The moral

- I am naïve.
- I do believe in axiomatisations, because axioms provide insight in calculi in a very elegant and high level manner.
- I do not think axiomatisations by nature have a practical (i.e. algorithmical) impact.
- Maybe sometimes they have.

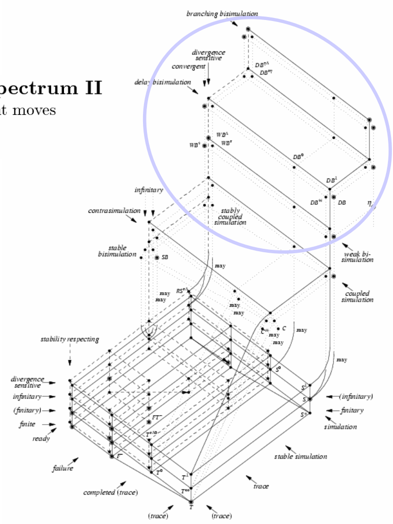
5

So, let's look out for some elegant axioms

Our starting point:

The Linear Time – Branching Time Spectrum II
The semantics of sequential systems with silent moves
Preliminary version
R.J. van Glabbeek*

- We aim to provide sound and complete axiomatisations for the divergence sensitive spectrum of weak bisimulations.
- We think this will help to provide a profound understanding of divergence in process algebra



How come I am interested in this issue

- There was a paper in Concur'98

Priority and Maximal Progress are completely axiomatisable (extended abstract)

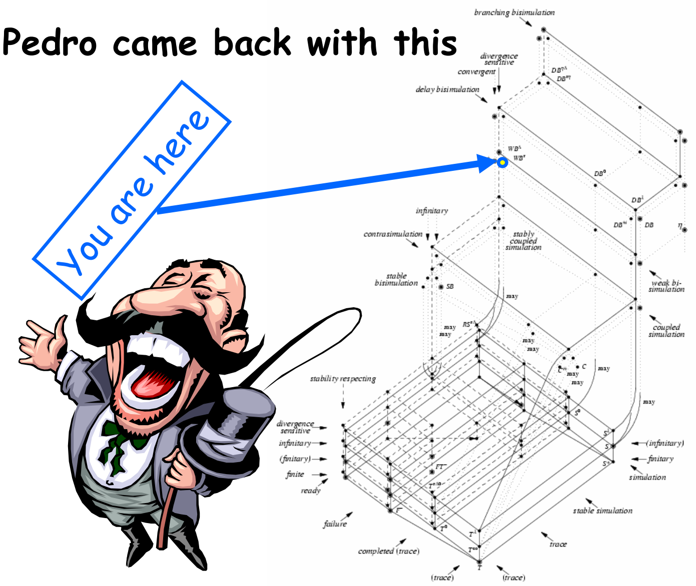
Holger Hermanns[1] and Markus Lohrey[2]

which was discussed on Tuesday by Mario Bravetti.

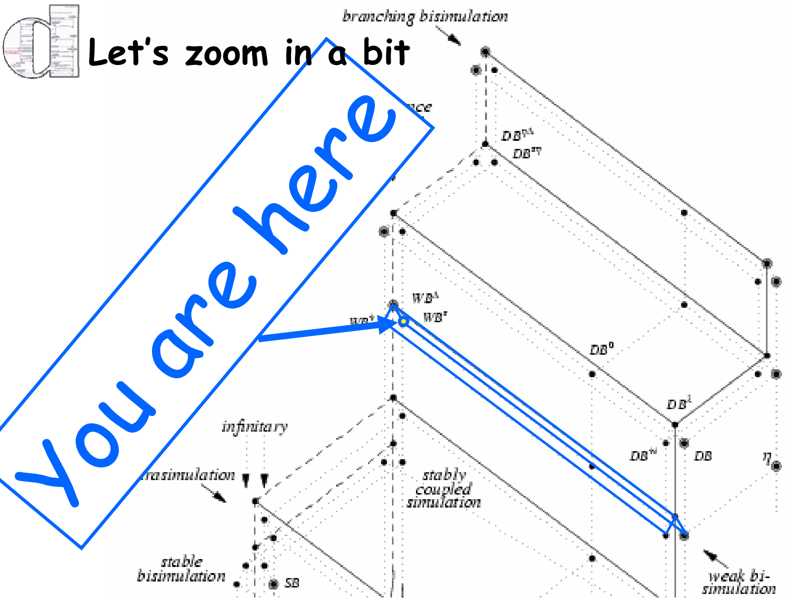
- In that paper we give an axiomatisation of a non-standard weak bisimulation with a specific treatment of divergence.
- A little later I discussed with Pedro R. D'Argenio, who at some point ran off to grap the spectrum-II paper.

7

Pedro came back with this



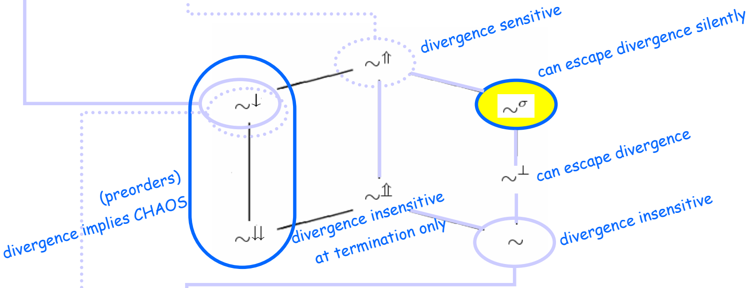
Let's zoom in a bit



What we are going to address today

D.J. Walker. Bisimulation and divergence. *Inf. Comp.*, 85:202–241, 1990.

J.A. Bergstra, J.W. Klop, and E.-R. Olderog. Failure semantics with fair abstraction. Report CS-R8609, CWI, Amsterdam, 1986.



R. Milner. A Complete Axiomatisation for Observational Congruence of Finite-State Behaviours. *Inf. Comp.*, 91(227-247), 1989.

L. Aceto and M. Hennessy. Termination, deadlock, and divergence. *J. ACM*, 39(1):147–187, 1992.

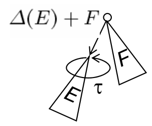
10

Overview

- Setting the stage
- The divergence sensitive spectrum
- The axiomatisation result we can offer
- A bit of the proof strategy
- The open problems
- Lunch

11

Calculus and semantics

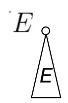


- We consider agents generated by the following grammar

$$\mathcal{E} \quad ::= \quad a.\mathcal{E} \quad | \quad \mathcal{E}+\mathcal{E} \quad | \quad recX.\mathcal{E} \quad | \quad X \quad | \quad \Delta(\mathcal{E})$$

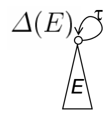
where $\Delta(E)$ exhibits all the behaviour of E, or may diverge.

- The semantic rules:



$$\frac{}{a.E \xrightarrow{a} E} \qquad \frac{E \xrightarrow{a} E'}{E+F \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E'}{F+E \xrightarrow{a} E'}$$

$$\frac{E\{recX.E/X\} \xrightarrow{a} E'}{recX.E \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E'}{\Delta(E) \xrightarrow{a} E'} \qquad \frac{}{\Delta(E) \xrightarrow{\tau} \Delta(E)}$$



12

The behavioural predicates

– $E\,\sigma$ (E is stable) if
$E \stackrel{\tau}{\longrightarrow}$ does not hold.

– $E\bot$ (E is inactive) if
$E \longrightarrow$ does not hold.

– $E \Uparrow$ (E may diverge) if
there are expressions E_i for $i \in \mathbb{N}$ such that
$E = E_0 \stackrel{\tau}{\longrightarrow} E_1 \stackrel{\tau}{\longrightarrow} E_2 \cdots$.

– $E\Uparrow\!\!\bot$ (E may diverge or silently turn inactive) if
$E \Uparrow$ or there exists F with $E \Longrightarrow F\bot$.

– Write $E \stackrel{a}{\Longrightarrow} F$ if $E \Longrightarrow\stackrel{a}{\longrightarrow}\Longrightarrow F$
– Write $E \stackrel{\hat{a}}{\Longrightarrow} F$ if ($a \neq \tau$ and $E \stackrel{a}{\Longrightarrow} F$) or ($a = \tau$ and $E \Longrightarrow F$).

The bisimulations

A symmetric relation $\mathcal{R}$ is a weak bisimulation (WB) if
for all P, Q, P' and $a \in \mathbb{A}$ the following holds:

if $(P,Q) \in \mathcal{R}$ and $P \stackrel{a}{\longrightarrow} P'$, then there exist Q' such that

$$Q \stackrel{\hat{a}}{\Longrightarrow} Q' \quad \text{and } (P',Q') \in \mathcal{R}$$

We say that $\mathcal{R}$ preserves a property ϕ if
for all P, Q, P' the following holds:

if $(P,Q) \in \mathcal{R}$ and $P \Longrightarrow P'\phi$, then there exists Q' such that $Q \Longrightarrow Q'\phi$

The bisimulations

A symmetric relation $\mathcal{R}$ is a weak bisimulation (WB) if
for all P, Q, P' and $a \in \mathbb{A}$ the following holds:

if $(P,Q) \in \mathcal{R}$ and $P \stackrel{a}{\longrightarrow} P'$, then there exist Q' such that

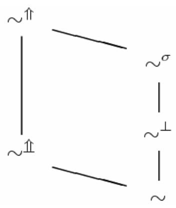
$$Q \stackrel{\hat{a}}{\Longrightarrow} Q' \quad \text{and } (P',Q') \in \mathcal{R}$$

We say that $\mathcal{R}$ preserves a property ϕ if
for all P, Q, P' the following holds:

if $(P,Q) \in \mathcal{R}$ and $P \Longrightarrow P'\phi$, then there exists Q' such that $Q \Longrightarrow Q'\phi$

A WB $\mathcal{R}$ is a WB$^{\phi}$ if it preserves ϕ. The relation $\sim^{\phi}$ is defined as the union of all WB$^{\phi}$. It is a WB$^{\phi}$, and is an equivalence relation.

These relations form a lattice



– WB$^{\sigma}$ is called *stable* WB
– WB$^{\bot}$ is called *completed* WB
– WB$^{\Uparrow}$ is called *divergent* WB
– WB$^{\Uparrow\!\bot}$ is called *divergent stable* WB
– and WB$^{\epsilon}$ just stands for WB

– $E\,\sigma$ (E is stable) if
$E \stackrel{\tau}{\longrightarrow}$ does not hold.

– $E\bot$ (E is inactive) if
$E \longrightarrow$ does not hold.

– $E \Uparrow$ (E may diverge) if
there are expressions E_i for $i \in \mathbb{N}$ such that
$E = E_0 \stackrel{\tau}{\longrightarrow} E_1 \stackrel{\tau}{\longrightarrow} E_2 \cdots$.

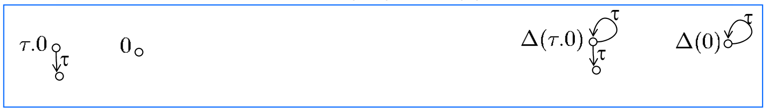
– $E\Uparrow\!\!\bot$ (E may diverge or silently turn inactive) if
$E \Uparrow$ or there exists F with $E \Longrightarrow F\bot$.

None of them is a congruence for +

- This is no surprise, note however:

For $\phi \in \{\Uparrow, \Uparrow\!\bot, \sigma, \bot\}$, $\sim^{\phi}$ is not a congruence with respect to the Δ-operator.

For instance $\tau.0 \sim^{\Uparrow} 0$, but $\Delta(\tau.0) \not\sim^{\Uparrow} \Delta(0)$.

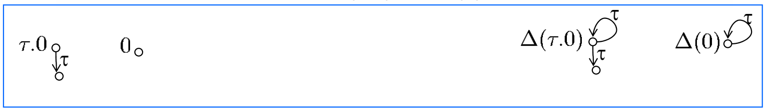


None of them is a congruence for +

- This is no surprise, note however:

For $\phi \in \{\Uparrow, \Uparrow\!\bot, \sigma, \bot\}$, $\sim^{\phi}$ is not a congruence with respect to the Δ-operator.

For instance $\tau.0 \sim^{\Uparrow} 0$, but $\Delta(\tau.0) \not\sim^{\Uparrow} \Delta(0)$.



The coarsest congruence is obtained with the usual root condition.

$\simeq^{\phi}$ is the relation containing exactly the pairs (P,Q) satisfying:

– if $P \stackrel{a}{\longrightarrow} P'$, then $Q \stackrel{a}{\Longrightarrow} Q'$ and $P' \sim^{\phi} Q'$ for some Q'
– if $Q \stackrel{a}{\longrightarrow} Q'$, then $P \stackrel{a}{\Longrightarrow} P'$ and $P' \sim^{\phi} Q'$ for some P'

Lifting to open expressions is as usual.

So, we are going to axiomatise these coarsest congruences

- The core axioms:

$(S1)\ E + F = F + E$	$(\tau 1)\ a.\tau.E = a.E$
$(S2)\ E + (F + G) = (E + F) + G$	$(\tau 2)\ \tau.E + E = \tau.E$
$(S3)\ E + E = E$	$(\tau 3)\ a.(E + \tau.F) = a.(E + \tau.F) + a.F$
$(S4)\ E + 0 = E$	

α-conversion $(rec1)$ if Y is not free in $recX.E$ then $recX.E = recY.(E\{Y/X\})$

unwinding $(rec2)$ $recX.E = E\{recX.E/X\}$

unique solution $(rec3)$ if X is guarded in E and $F = E\{F/X\}$ then $F = recX.E$

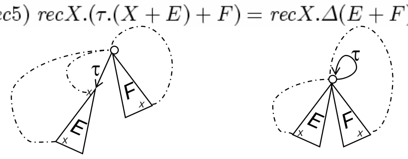
unguardedness $(rec4)$ $recX.(X + E) = recX.E$

$(rec5)$ $recX.(\tau.(X + E) + F) = recX.\Delta(E + F)$?

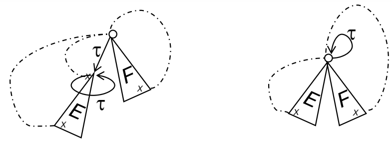
$(rec6)$ $recX.(\Delta(X + E) + F) = recX.\Delta(E + F)$?

The non-standard - but core - axioms

$(rec5)$ $recX.(\tau.(X + E) + F) = recX.\Delta(E + F)$

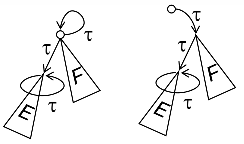


$(rec6)$ $recX.(\Delta(X + E) + F) = recX.\Delta(E + F)$

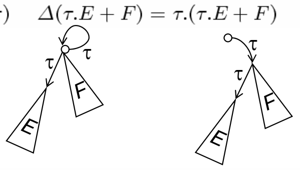


The distinguishing axioms

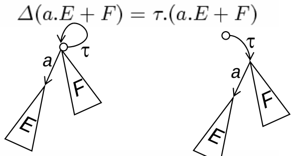
$(\Uparrow)$ $\Delta(\Delta(E) + F) = \tau.(\Delta(E) + F)$



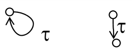
(σ) $\Delta(\tau.E + F) = \tau.(\tau.E + F)$



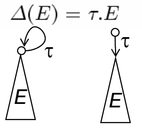
$(\perp)$ $\Delta(a.E + F) = \tau.(a.E + F)$



$(\Uparrow\!\!\Uparrow)$ $\Delta(0) = \tau.0$

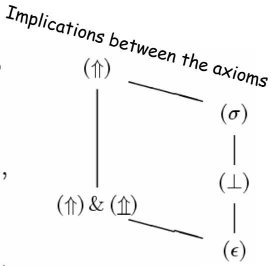


(ϵ) $\Delta(E) = \tau.E$



The equational theories

- Axioms for $\simeq^{\Uparrow}$: core axioms plus axiom $(\Uparrow)$,
- Axioms for $\simeq^{\sigma}$: core axioms plus axiom (σ),
- Axioms for $\simeq^{\perp}$: core axioms plus axiom $(\perp)$,
- Axioms for $\simeq^{\epsilon}$: core axioms plus axiom (ϵ),
- Axioms for $\simeq^{\Uparrow\!\!\Uparrow}$: core axioms plus $(\Uparrow)$ and $(\Uparrow\!\!\Uparrow)$.



We write $E =^{\phi} F$ if $E = F$ can be derived by application of the axioms for $\simeq^{\phi}$.

Soundness

Theorem *If $E, F \in \mathbb{E}$ and $E =^{\phi} F$ then $E \simeq^{\phi} F$.*

The following laws can be derived:

$(\Uparrow_1)$ $\Delta(E) =^{\Uparrow} \Delta(E) + E$

$(\Uparrow_2)$ $\Delta(E) =^{\Uparrow} \tau.\Delta(E) + E$

$(\Uparrow_3)$ $\Delta(E) =^{\Uparrow} \tau.\Delta(E)$

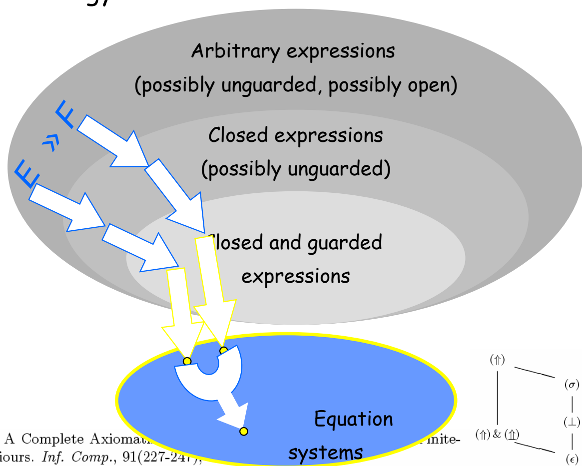
$(rec7)$ $recX.(\tau.(X + E) + F) =^{\Uparrow} recX.(\tau.X + E + F)$

Milner's axiom (rec5)

$(\Uparrow)$	$\Delta(\Delta(E) + F) = \tau.(\Delta(E) + F)$
(σ)	$\Delta(\tau.E + F) = \tau.(\tau.E + F)$
$(\perp)$	$\Delta(a.E + F) = \tau.(a.E + F)$
(ϵ)	$\Delta(E) = \tau.E$
$(\Uparrow\!\!\Uparrow)$	$\Delta(0) = \tau.0$

Completeness

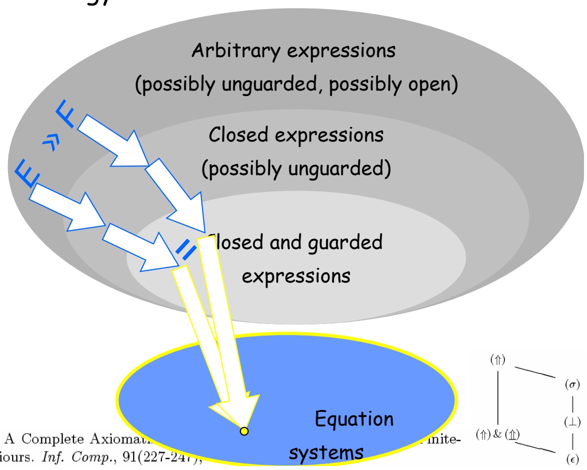
- Milner's proof strategy:



R. Milner. A Complete Axiomat... -nite-State Behaviours. *Inf. Comp.*, 91(227-24...)

Completeness

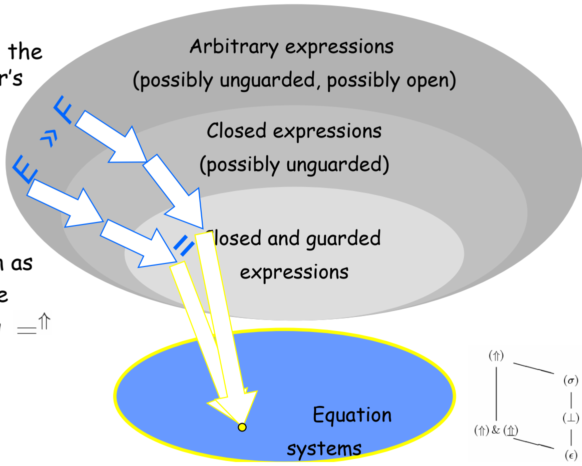
- Milner's proof strategy:



R. Milner. A Complete Axiomati... finite-State Behaviours. *Inf. Comp.*, 91(227-247),

Completeness

- Our strategy:
  - Proceed along the lines of Milner's proof
  - Take care of divergence
  - Work as much as possible in the finest setting $=^{\Uparrow}$



Overview

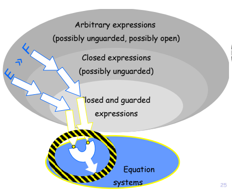
- Setting the stage
- The divergence sensitive spectrum
- The axiomatisation result we can offer
- A bit of the proof strategy
- The open problems
- Lunch

A few bits of the proof

27

Equation systems

$$
\begin{aligned}
X_1 &= \sum a_j.X_j + \sum Y\\
X_2 &= \sum a_j.X_j + \sum Y\\
&\vdots\\
X_n &= \sum a_j.X_j + \sum Y
\end{aligned}
$$



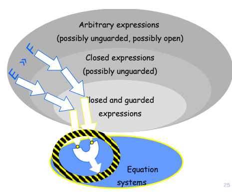
An *equation system over the free variables* V *and the formal variables* $\boldsymbol{X}$ is a set of equations $\mathcal{E} = \{X_i = E_i \mid 1 \le i \le n\}$ such that $E_i \in \mathbb{E}$ and $\mathbb{V}(E_i) \subseteq \{X_1, \ldots, X_n\} \cup V$ for $1 \le i \le n$.

29

Equation systems

$$
\Omega^{\Sigma}\left\{
\begin{aligned}
X_1 &= \sum a_j.X_j + \sum Y\\
X_2 &= \sum a_j.X_j + \sum Y\\
&\vdots\\
X_n &= \sum a_j.X_j + \sum Y
\end{aligned}\right.
$$

$$
\Omega^{\Delta}\left\{
\begin{aligned}
X_n+1 &= \Delta(\sum a_j.X_j + \sum Y)\\
X_n+2 &= \Delta(\sum a_j.X_j + \sum Y)\\
&\vdots\\
X_n+m &= \Delta(\sum a_j.X_j + \sum Y)
\end{aligned}\right.
$$

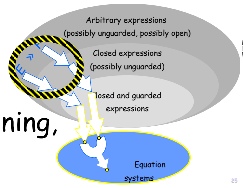


The remainder of the proof
(that the SESs of any two ϕ-equivalent expressions
can be merged into a single SES)
requires careful bookkeeping and a few lemmas.

If $E \Uparrow$*, then there are* G, H *with* $E =^{\Uparrow} \Delta(G) + H$.

30

Completeness for open expressions



- Our proof relies on purely syntactic reasoning, but this fails for ?.

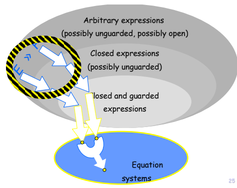
Lemma 20. *Let $\phi \neq \bot$ and $E, F \in \mathbb{E}$. If $a \in \mathbb{A}\setminus\{\tau\}$ does neither occur in E nor in F, then $E\{a.0/X\} =^{\phi} F\{a.0/X\}$ implies $E =^{\phi} F$.*

- So we have completeness also for open expressions.

This is false for $\phi = \bot$.

We have $\tau.a.0 =^{\bot} \Delta(a.0)$ but $\tau.X \neq^{\bot} \Delta(X)$, since $\tau.0 \not\simeq^{\bot} \Delta(0)$.

Completeness for open expressions for =?



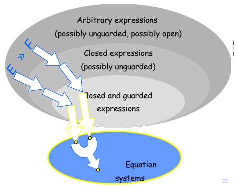
- Well, we add a brute-force axiom:

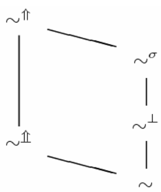
($\bot'$) If $E\{0/X\} = F\{0/X\}$ and $E\{a.0/X\} = F\{a.0/X\}$ where $a \in \mathbb{A}\setminus\{\tau\}$ does neither occur in E nor in F, then $E = F$.

This new axiom is indeed sound for $\simeq^{\bot}$.

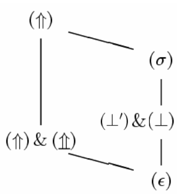
- And this gives us completeness for =?.

Again the result



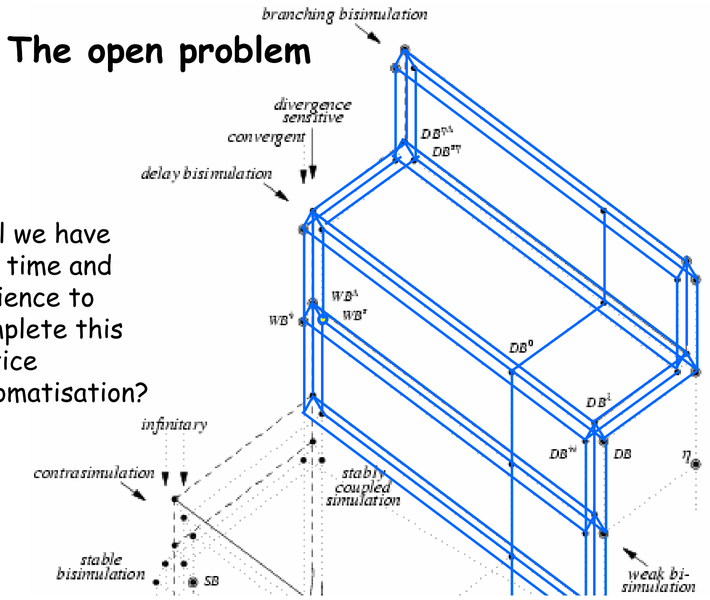


$$
\begin{array}{ll}
(\Uparrow) & \Delta(\Delta(E)+F) = \tau.(\Delta(E)+F) \\
(\sigma) & \Delta(\tau.E+F) = \tau.(\tau.E+F) \\
(\bot) & \Delta(a.E+F) = \tau.(a.E+F) \\
(\epsilon) & \Delta(E) = \tau.E \\
(\Uparrow) & \Delta(0) = \tau.0
\end{array}
$$



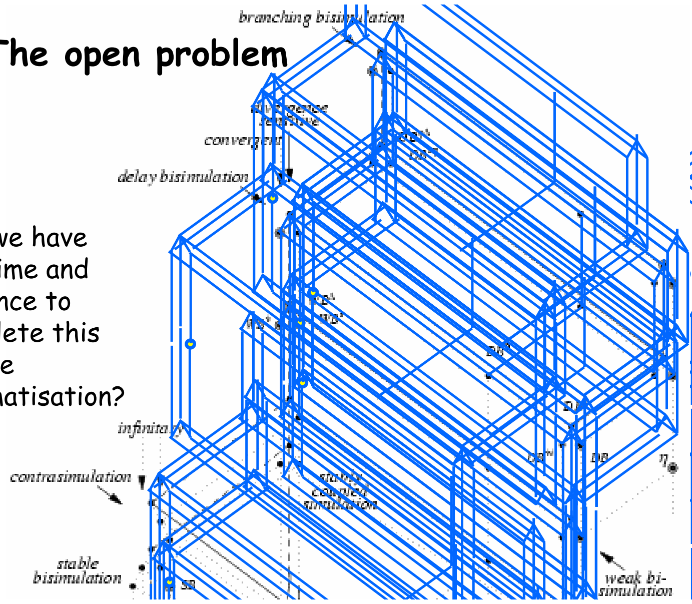
The open problem

- Will we have the time and patience to complete this lattice axiomatisation?



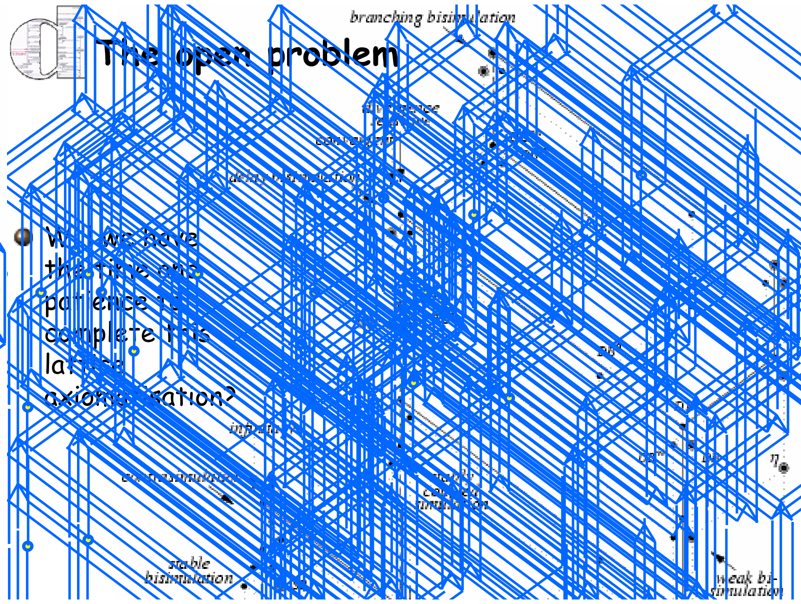
The open problem

- Will we have the time and patience to complete this lattice axiomatisation?



The open problem

- Will we have the time and patience to complete this lattice axiomatisation?



Language and Tool Design of MoDeST

a Process Algebra aimed for Embedded Systems

Joost-Pieter Katoen

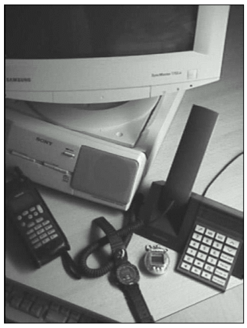
Department of Electrical Engineering, Mathematics and Computer Science
University of Twente

joint work with: Pedro D'Argenio, Henrik Bohnenkamp, Holger Hermanns and Ric Klaren

Workshop on Open Problems and Future Perspectives of Process Algebra, July 21, 2003

What's special about embedded software?

- It is embedded
  - ⇒ high requirements on performability
  - ⇒ need to react promptly
  - ⇒ integrity of generated outputs is vital
- Interacts permanently
  - ⇒ non-terminating behaviour
  - ⇒ concurrency
- It does not run on computers
  - ⇒ requires efficient usage of resources
  - ⇒ its performance is essential
  - ⇒ over-dimensioning is unacceptable



prevailing abstractions of software forget about these "non-functional" aspects

Relevant properties of embedded software

- Is the system correct? — verification
- Are two models the "same"? — equivalence checking
- Does the implementation conform to the model? — testing
- Is it timely? — real-time verification
- What is its performance? — performance evaluation
- What is its availability? — dependability analysis

Diversity of tools and models

- Verification: model checkers — SPIN and SMV
- Equivalence checking: bisimulation checkers — μCRL toolset and CADP
- Conformance testing: test generation tools — ToRX and TGV
- Real-time verification: model checkers — UppAal
- Performance evaluation: simulators and analyzers — Marca and Möbius
- Dependability evaluation — Möbius

Our philosophy

- How to ensure consistency between the different models used?
- Who wants to specify all these models anyway?
- Our view:

  A single-specification, multi-solution approach

- One (core) formalism, with one semantics
- Extract models for particular property of interest
- Provide mappings from other modeling languages (UML statechart)

What is MoDeST?

- A formal specification language for

  MOdelling and DEscription of Stochastic and Timed systems

- Inspired by:
  - process algebra FSP [Kramer & McGee]
  - stochastic process algebra (with nondeterminism) [Bravetti, D'Argenio & Katoen, Hermanns]
  - timed automata [Alur & Dill, Bornot & Sifakis]
  - (simple) probabilistic automata [Segala]
  - Promela [Holzmann]
  - . . . a bit of Java
- Exploiting results from concurrency theory for probabilistic systems

Design rationales

- Compositionality
- Orthogonality of language concepts
- Usability: aimed to be modern and "light-weight" language
  - syntax (and data types) close to C and Promela
  - control structures like loops and exception handling
  - possibilities to link user-defined libraries
  - (limited) control of the atomicity of assignments
- Expressiveness for verification and stochastic systems
  - nondeterminism of actions and timing
  - probabilistic behaviour of actions and timing
- Clean and formal semantics

Pure nondeterminism

```
process BitChoice(bool b) {
  alt{                          // nondeterministically
    :: when(tt) b = 0           // choose 0
    :: when(tt) b = 1           // or choose 1
  }
}
```

Important for:

- incomplete information: under-specification
- implementation freedom: only describes *what* a system must do, not how
- concurrency freedom: no assumption about relative speed of components
- external environment: no stipulation of behavior environment

Probabilistic choice

```
action a, b;                    // declare actions a and b
int x = 1, y = 10;              // declare integer variables x and y

a palt {                        // offer a, then ....
  :49:                          // in 98% of the cases
      ⟨x+= 1, y+= 1⟩; b         // increment x and y and offer b
  : 1:                          // in 2% of the cases
      ⟨x = y, y = x⟩; stop      // swap x and y and halt
}
```

action-guardedness is required to avoid asynchronous parallel composition

Mixed probability and nondeterminism

```
int s;                                  // stake of roulette player

s = 1;                                  // start with 1$
do{ :: choose palt {                    // iteratively choose red or black
          :1: s = 2 * s                 // lost; double the stake
          :1: alt{
                 :: s = N; break        // risk all in last game
                 :: s = 1 }             // repeat strategy
          }
  };
choose palt { :1: happy :1: sad }       // last game
```


Nondeterministic delay

```
action a, b;          // declare actions a and b
clock c;              // declare clock c

c = 0;                // reset clock c
urgent(c > 20)        // ultimately when c = 20
when(c > 10)          // when c exceeds 10
    a                 // enable action a
  ; b                 // offer action b
```

action a will be offered in the interval $(10, 20]$ since the start
action b will be offered at some time point arbitrarily later

Random delay

```
action a;             // declare action a
float x;              // declare float x
clock c;              // declare clock c

x = exp(2.1);         // sample an exp. distr. with rate 2.1
c = 0;                // reset clock c
urgent(c > x)         // ultimately when c equals x
when(c ⩾ x)           // when c is at least x
  a                   // offer action a
```

Syntax of simple processes

act	action
stop	deadlock
error	unhandled error
break	break process
throw$(excp)$	throw exception
process *ProcName*$(t_1\ x_1, \ldots, t_k\ x_k)$ $\{dcl\ P\}$	process declaration
ProcName$(e_1, \ldots, e_k)$	process instantiation
extend $\{act_1, \ldots, act_k\}$ P	alphabet extension
relabel $\{act_1, \ldots, act_k\}$ by $\{act'_1, \ldots, act'_k\}$ P	relabeling

process bodies need to be guarded

Syntax of composed processes

when(b) P	enabling
urgent(b) P	urgency
par$\{::\ P_1\ \ldots\ ::\ P_k\}$	parallel
$P_1;\ P_2$	sequential
do$\{::\ P_1\ \ldots\ ::\ P_k\}$	loop
alt$\{::\ P_1\ \ldots\ ::\ P_k\}$	alternative
act palt $\{:w_1:\ asgn_1\ ;\ P_1\ \ldots\ :w_k:\ asgn_k\ ;\ P_k\}$	probabilistic choice
try$\{P\}$ catch $excp_1$ $\{P_1\}$ $\ldots$ catch $excp_k$ $\{P_k\}$	exception handling

data types include `bool`, `int`, `float`, `clock` and `array` and structured combinations (`record`) of this

Some handy abbreviations

alt$\{::$ when(b_1) P_1 $\ldots$ $::$ when(b_k) P_k $::$ else $Q\}$

$\stackrel{\text{def}}{=}$ alt$\{::$ when(b_1) P_1 $\ldots$ $::$ when(b_k) P_k $::$ when$(\neg \bigvee_{i=1}^{k} b_i)$ $Q\}$

$\langle x_1 = e_1, \ldots, x_k = e_k \rangle$ $\stackrel{\text{def}}{=}$ urgent(tt) τ palt $\{:1:$ $\langle x_1 = e_1, \ldots, x_k = e_k$

hide $\{act_1, \ldots, act_k\}$ P $\stackrel{\text{def}}{=}$ relabel $\{act_1, \ldots, act_k\}$ by $\{\underbrace{\tau, \ldots, \tau}_{k \text{ times}}\}$ P

invariant(b) P $\stackrel{\text{def}}{=}$ urgent$(\neg b)$ when(b) P

while$(b)\{P\}$ $\stackrel{\text{def}}{=}$ do$\{::$ when(b) P $::$ else break$\}$

Two layers of semantics

MoDeST Language

↓ Structural Operational Semantics

Discrete State Model: Stochastic Timed Automata

↓ Layered Semantics

Continuous State Model: Timed Probabilistic Transition Systems

Stochastic timed automata

- S, a set of locations
- Act, a set of actions
- $\longrightarrow$, a transition relation such that $s \xrightarrow{a,g,d} \mathcal{P}$ where
  - s is the current location
  - action $a \in$ Act is offered
  - g is a guard (boolean expression)
  - d is a deadline (boolean expression)
  - $\mathcal{P}$ is a (discrete) probability spaces over pairs (assignments, location)

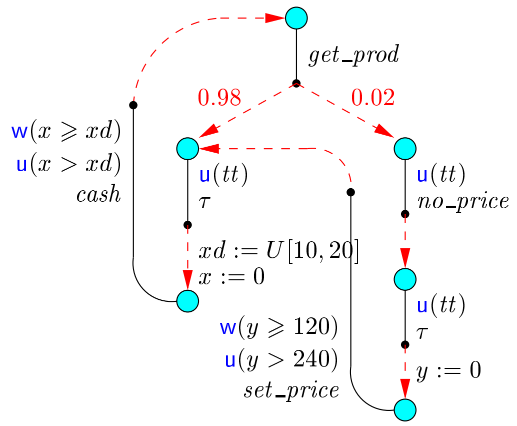
mixture of timed automata with deadlines [Bornot & Sifakis], stochastic automata [D'Argenio *et al.*], and simple probabilistic automata [Segala]

Interpretation of stochastic timed automata

- A state records the current location and valuation of all variables
- If $s \xrightarrow{a,g,d} \mathcal{P}$ and the current valuation satisfies g, then:
  - with probability $\mathbf{P}(A, s')$ where $\mathbf{P}$ being the probability measure of $\mathcal{P}$
  - the valuation is changed according to the sequence of assignments A, and
  - the next location is s'
- Time is advanced with $t > 0$ if in the current location:
  - no deadline is met for any $t' \leqslant t$
  - increasing all clock variables with t
  - letting all other variables unchanged

continuous space model: timed probabilistic transition system

An example stochastic timed automaton



Submodels of stochastic timed automata

	LTS	PTS	TA	PTA	MC	GSMP	IMC	SA
probabilistic branching	-	+	-	+	+	+	+	-
clocks	-	-	+	+	R	+	R	+
random variables	-	-	-	-	R	+	R	+
delay nondeterminism	-	-	+	+	-	-	-	-
action nondeterminism	+	+	+	+	-	-	+	+

Operational semantics: simple processes

$a \xrightarrow{a,tt,ff} \mathcal{P}$ with $\mathbf{P}(\varnothing, \surd) = 1$

$\mathsf{error} \xrightarrow{\bot,tt,ff} \mathcal{P}$ with $\mathbf{P}(\varnothing, \mathsf{error}) = 1$

$\mathsf{break} \xrightarrow{\mathsf{break},tt,ff} \mathcal{P}$ with $\mathbf{P}(\varnothing, \surd) = 1$

$\mathsf{throw}(excp) \xrightarrow{excp,tt,ff} \mathcal{P}$ with $\mathbf{P}(\varnothing, \mathsf{error}) = 1$

$$\frac{P[x_1/e_1, \ldots, x_k/e_k] \xrightarrow{a,g,d} \mathcal{P}}{Proc(e_1, \ldots, e_k) \xrightarrow{a,g,d} \mathcal{P}} \quad \text{provided } \mathsf{process}\ Proc(x_1, \ldots, x_k)\{P\}$$

$$\frac{P \xrightarrow{a,g,d} \mathcal{P} \qquad f = [a_1/a_1', \ldots, a_k/a_k']}{\mathsf{relabel}\ \{a_1, \ldots, a_k\}\ \mathsf{by}\ \{a_1', \ldots, a_k'\}\ P \xrightarrow{f(a),g,d} \mathcal{R}}$$

where $\mathbf{R}(A, \surd) = \mathbf{P}(A, \surd)$ and $\mathbf{R}(A, \mathsf{relabel}\ f\ \mathsf{by}\ P') = \mathbf{P}(A, P')$

Operational semantics: composed processes

$$\frac{P \xrightarrow{a,g,d} \mathcal{P}}{\mathsf{when}(b)\ P \xrightarrow{a,b \wedge g,d} \mathcal{P}}$$

$$\frac{P \xrightarrow{a,g,d} \mathcal{P}}{\mathsf{urgent}(b)\ P \xrightarrow{a,g,b \vee d} \mathcal{P}}$$

$$\frac{P_i \xrightarrow{a,g,d} \mathcal{P}_i \qquad (1 \leqslant i \leqslant k)}{\mathsf{alt}\{::\ P_1\ \ldots\ ::\ P_k\} \xrightarrow{a,g,d} \mathcal{P}_i}$$

$$a\ \mathsf{palt}\ \{:w_i:\ A_i\ ;\ P_i\}_{i \in I} \xrightarrow{a,tt,ff} \mathcal{P} \text{ with } \mathbf{P}(A_i, P_i) = \frac{w_i}{\sum_{j \in I} w_j}$$

Iteration

$$\mathsf{do}\{::\ P_i\}_{i \in I} \stackrel{\text{def}}{=} \mathsf{auxdo}\{\mathsf{alt}\{::\ P_i\}_{i \in I}\}\{\mathsf{alt}\{::\ P_i\}_{i \in I}\}$$

$$\frac{P \xrightarrow{a,g,d} \mathcal{P} \qquad (a \neq \mathsf{break})}{\mathsf{auxdo}\{P\}\{Q\} \xrightarrow{a,g,d} \mathcal{R}}$$

where $\mathbf{R}(A, \mathsf{auxdo}\{Q\}\{Q\}) = \mathbf{P}(A, \surd)$ and $\mathbf{R}(A, \mathsf{auxdo}\{P'\}\{Q\}) = \mathbf{P}(A, P')$

$$\frac{P \xrightarrow{\mathsf{break},g,d} \mathcal{P}}{\mathsf{auxdo}\{P\}\{Q\} \xrightarrow{\tau,g,d} \mathcal{R}} \text{ where } \mathbf{R}(A, \surd) = 1$$

Exception handling

raising an exception $\mathsf{throw}(excp) \xrightarrow{excp,tt,ff} \mathcal{P}$ with $\mathbf{P}(\varnothing, \mathsf{error}) = 1$

$$\frac{P \xrightarrow{a,g,d} \mathcal{P} \qquad \forall i \in I.\ a \neq excp_i}{\mathsf{try}\{P\}\{\mathsf{catch}\ excp_i\ \{P_i\}\}_{i \in I} \xrightarrow{a,g,d} \mathcal{R}}$$

where $\mathbf{R}(A, \mathsf{try}\{P'\}\{\mathsf{catch}\ excp_i\ \{P_i\}\}_{i \in I}) = \mathbf{P}(A, P')$ and $\mathbf{R}(A, \surd) = \mathbf{P}(A, \surd)$

$$\frac{P \xrightarrow{excp_i,g,d} \mathcal{P} \qquad i \in I}{\mathsf{try}\{P\}\{\mathsf{catch}\ excp_i\ \{P_i\}\}_{i \in I} \xrightarrow{\tau,g,d} \mathcal{R}} \text{ with } \mathbf{R}(\varnothing, P_i) = 1$$

Parallel composition

note that: $\text{par}\{:: P_1 \ \ldots \ :: P_k\} \stackrel{\text{def}}{=} (\ldots((P_1 \parallel_{B_1} P_2)\ldots)) \parallel_{B_{k-1}} P_k$

with $\parallel_B$ is CSP parallel composition and $B_j = \left(\bigcup_{i=1}^{j} \alpha(P_i)\right) \cap \alpha(P_{j+1})$

performing autonomous actions

$$\frac{P \xrightarrow{a,g,d} \mathcal{P} \qquad a \notin B}{P \parallel_B Q \xrightarrow{a,g,d} \mathcal{R}} \quad \text{with} \quad \begin{aligned} \mathbf{R}(A, P' \parallel_B Q) &= \mathbf{P}(A, P') \\ \mathbf{R}(A, Q \backslash B) &= \mathbf{P}(A, \surd) \end{aligned}$$

performing exceptions (or $\perp$)

$$\frac{P \xrightarrow{excp,g,d} \mathcal{P}}{P \parallel_B Q \xrightarrow{excp,g,d} \mathcal{R}} \quad \text{with } \mathbf{R}(A, \text{error}) = 1$$

Synchronization

$$\frac{P \xrightarrow{a,g,d} \mathcal{P} \qquad Q \xrightarrow{a,g',d'} \mathcal{Q} \qquad a \in B}{P \parallel_B Q \xrightarrow{a,g \wedge g',d \wedge d'} \mathcal{R}} \quad \text{if } a \text{ is patient}$$

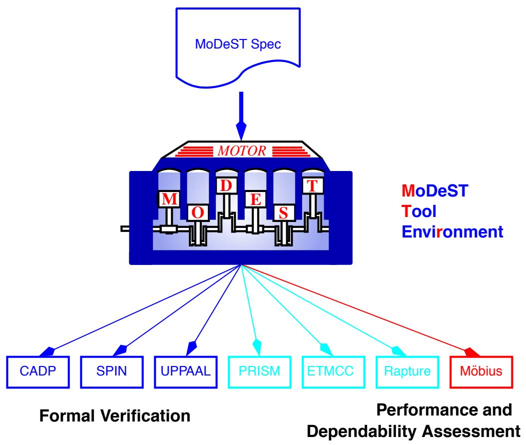
$$\frac{P \xrightarrow{a,g,d} \mathcal{P} \qquad Q \xrightarrow{a,g',d'} \mathcal{Q} \qquad a \in B}{P \parallel_B Q \xrightarrow{a,g \wedge g',d \vee d'} \mathcal{R}} \quad \text{if } a \text{ is impatient}$$

where

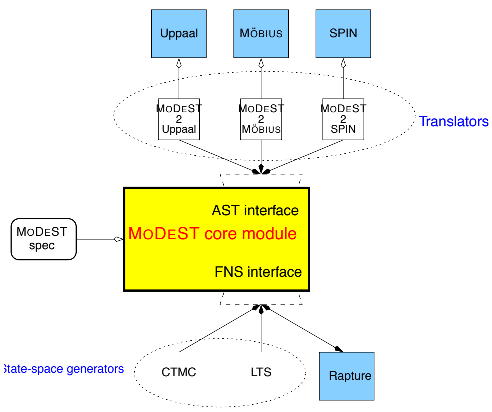
$$\begin{aligned} \mathbf{R}(A \cup A', P' \parallel_B Q') &= \mathbf{P}(A, P') \cdot \mathbf{Q}(A', Q') \\ \mathbf{R}(A \cup A', \surd) &= \mathbf{P}(A, \surd) \cdot \mathbf{Q}(A', \surd) \\ \mathbf{R}(A \cup A', Q' \backslash B) &= \mathbf{P}(A, \surd) \cdot \mathbf{Q}(A', Q') \text{ and symmetric} \end{aligned}$$

if $A \cup A'$ is not a proper assignment, a predefined exception is raised

Aim of the MOTOR tool environment

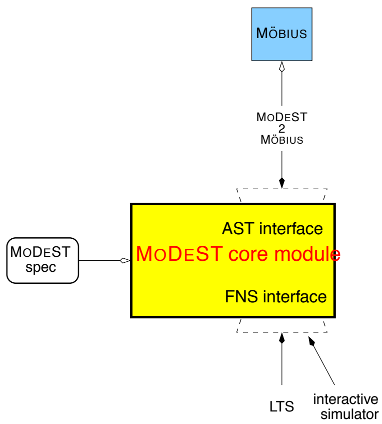


The current MOTOR tool architecture



- Core module
  - parser (using ANTLR)
  - operational semantics
- First/Next-State API
  - access global state space
  - current "state"
  - enabled transitions
- AST API
  - access abstract syntax tree
  - useful for tree walkers
  - used by FNS API
- Satellite modules
  - adapting to other tools
  - "native" analysis means

Current state of implementation



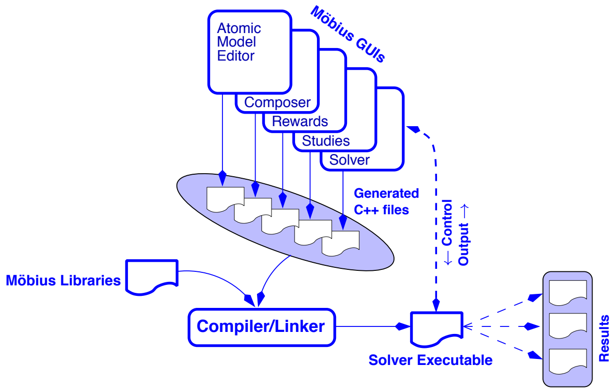
Open FNS interface

- `State* getInitialState()`
  - returns a pointer to the initial global state
- `State* getCurrentState()`
  - returns a pointer to the current global state
- `transition_list& getTransitions(State* state)`
  - returns a list of transitions outgoing from `state`
- `Guard* getGuard(), Guard* getDeadline()`
  - returns guard and deadline of transition
- `bool isNeverEnabled()`
  - returns true if and only if guard transition is constantly false
- `void Fire(Branch& branch, double delay)`
  - executes transition by taking `branch` and advance time by `delay`

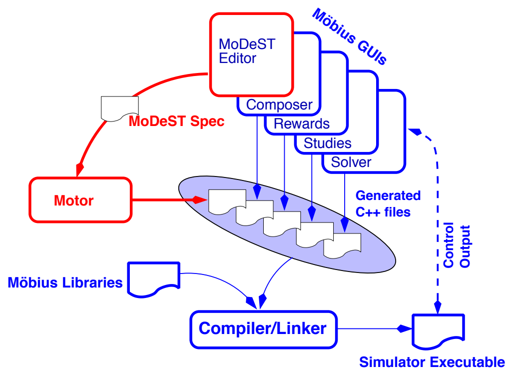
Some implementation details

- Simple interactive simulator
  - textual user interface to the FNS API
- State-space generator
  - can output `.dot`, `.aut` files of transition system
  - and compact `.bcg` format for the CADP toolset
  - useful for verification and visualization facilities of CADP
  - and for on-the-fly test generation (using ToRX)
- 23,000 lines of C++-code (compiled with `g++`)
- Command-line interface, input: plain ascii MoDeST model

Möbius tool architecture



Integrating MOTOR into the Möbius framework



Epilogue

- Summarizing:
  - MoDeST is aimed at a single-specification multi-solution approach
  - a single, coherent specification used for various analysis phases
  - main ingredients: compositionality and formal semantics
  - first prototype realised + linking to Möbius solvers
- Future work:
  - extend the integration with Möbius
  - conduct industrial case studies (integration in UML design flow)
  - link to model checking
  - extend the MOTOR tool environment (Uppaal, ToRX, ETMCC)
  - safe (compositional) abstractions to simpler model classes
  - incorporate hybrid aspects?

PROCESS ALGEBRAS and ARITHMETICS

Anna Labella

joint work with
Rocco De Nicola

Process Algebra Workshop
Bertinoro

LANGUAGES and TREES

- Language locally accepted by an automaton
- Tree as local behaviour of a nondeterministic process

$$X(Y + Z) \neq XY + Y Z$$

2

The category of trees: Tree(A)

vs

The algebra of languages P(A*)

A tree

$t = (X, e, d)$:

i. a set of runs: X

ii. an extent map $e: X \to A^*$,

ii. an agreement map $d : X \times X \to A^*$

s.t. for any x, y, z in X,

- $d(x, x) = e(x)$
- $d(x, y) \leq e(x)$ □ $e(y)$
- $d(x, y)$ □ $d(y, z) \leq d(x, z)$
- $d(x, y) = d(y, x)$

4

A morphism of trees

$f: t_1 \to t_2$

is a map $f : X_1 \to X_2$ satisfying

i. $e_2(f(x)) = e_1(x)$

ii. $d_1(x, y) \leq d_2(f(x), f(y))$

5

The cartesian category Tree(A)

- has finite limits: terminal object is A*
- has sums
- is distributive
- has subobject classifier A*A*
- Actually is a pretopos and its objects are presentations of the objects of sh(A*)

6

The monoidal category Tree(A)

Concatenation of two trees:

$t_1 \otimes t_2 = \langle X, e, d \rangle$, where

- $X = X_1 \times X_2$
- $e(\langle x_1, x_2 \rangle) = e_1(x_1) \cdot e_2(x_2)$
- $d(\langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle) = \begin{cases} d_1(x_1, y_1).d_2(x_2, y_2) & \text{if } x_1 = y_1 \\ d_1(x_1, y_1) & \text{otherwise} \end{cases}$

- $1 = \langle \{x\}, e(x) = \varepsilon, d(x, x) = \varepsilon \rangle$ is the unit
- Tree(A) is monoidal right closed
- and right distributive

7

Iteration

If $t = (X, e, d)$,

$t^* = (X\infty, e\,\infty, d\,\infty)$:

- $X\infty = \{ \langle x_1, x_2, \ldots, x_n \rangle \mid n \in N \text{ and } x_i \in X \}$
- $e\infty \langle x_1, x_2, \ldots, x_n \rangle = e(x_1).\, e(x_2) \ldots e(x_n)$
- $d\infty(\langle x_1, x_2, \ldots, x_n \rangle, \langle y_1, y_2, \ldots, y_m \rangle) =$

$$\begin{cases} e(x_1).\, e(x_2) \ldots e(x_k)\, d(x_{k+1}, y_{k+1}) & x_i = y_i,\ 0 \le i \le k;\ x_{k+1} \neq y_{k+1} \\ e(x_1).\, e(x_2) \ldots e(x_k) & x_k = y_k,\ 0 \le i \le k;\ n = k \text{ or } m = k \end{cases}$$

8

Iteration

- t^* is the colimit of the chain of its approximants.

- Tree(A) has initial solution for recursive equations of the form

$$x = Ux + V$$

and it actually is

$$U^*V$$

9

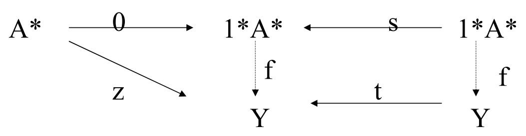
NNO

$1^* A^*$ is initial solution of the equation

$$x = x + A^*$$

Indeed, $1^*A^* = 11^*A^* + A^*$

Or, equivalently, P-L axiom holds



$$\begin{cases} f\,0 = z \\ f\,s = t\,f \end{cases}$$

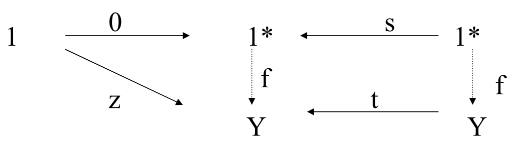
10

LNNO

1^* is initial solution of the equation

$$x = x + 1$$

equivalently, the left monoidal form of P-L axiom holds



$$\begin{cases} f\,0 = z \\ f\,s = t\,f \end{cases}$$

11

NNO and LNNO

The parametric form for LNNO is

$x = x + K$ i.e. $x = 1 \otimes x + 1 \otimes K$

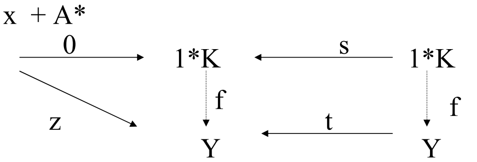
and coincides with the parametric form for NNO

$$x = A^* \times x + A^* \times K$$

In particular

$x = 1x + A^*$ has the same initial solution as

$x = A^* \times x + A^*$



$$\begin{cases} f\,0 = z \\ f\,s = t\,f \end{cases}$$

12

A general theorem

- Given a distributive category C, which is also right distributive as a monoidal category, if there exists I*, then:
  - There exists a LNNO LN

    LN = I*
  - There exists also a NNO N and

    N = LN ⊗ **1** = I* ⊗ **1**

N and LN induce the same kind of recursion.

13

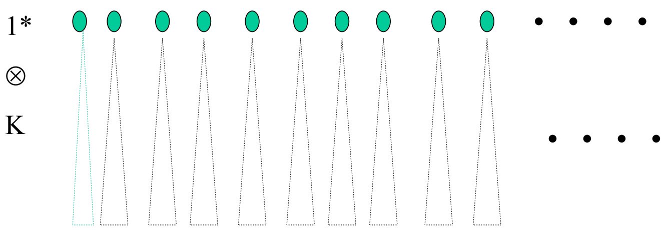
Tree(A) and arithmetics

- Within Tree(A) we can reconstruct arithmetics

- Tree(A) is an arithmetical universe

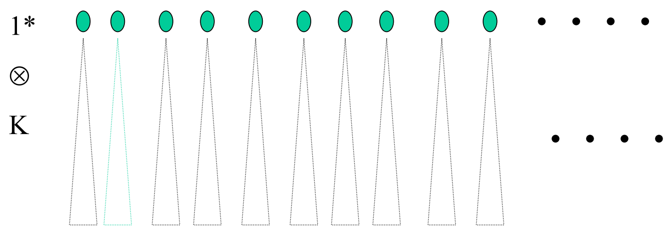
14

Horizontal recursion



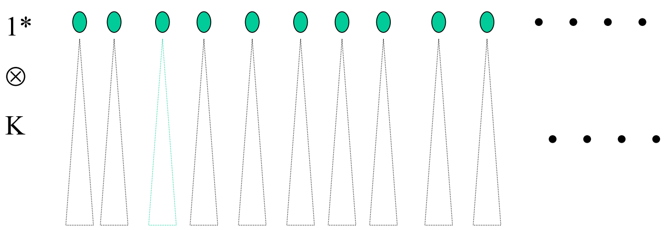
15

Horizontal recursion



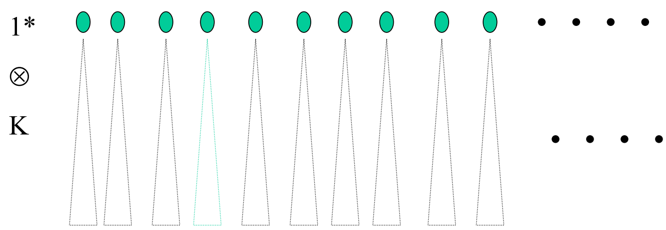
16

Horizontal recursion



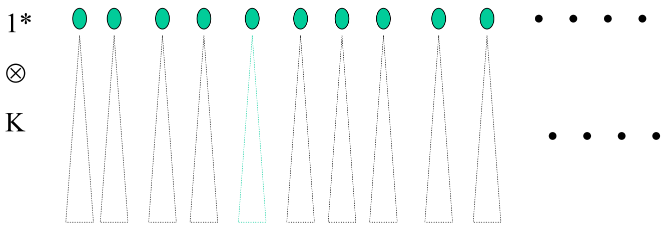
17

Horizontal recursion



18

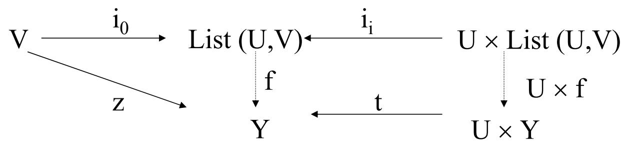
Horizontal recursion



Tree(A) and lists

- Tree(A) can represent lists in the sense of Cockett because it has initial solution for equations

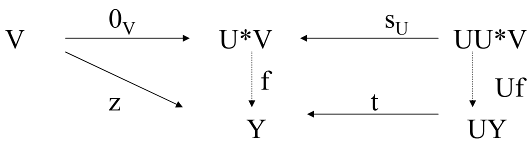
$$x = U \times x + V$$



Vertical recursion

- The following recursion is analogous to Cockett diagram for lists, and we can consider it as a parametric form of a U-arithmetics in presence of a parameter V

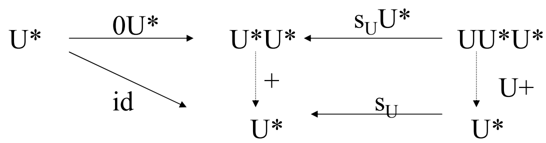
$$x = U \otimes x + V$$



$$\begin{cases} f\,0_V = z \\ f\,s_U = t\,Uf \end{cases}$$

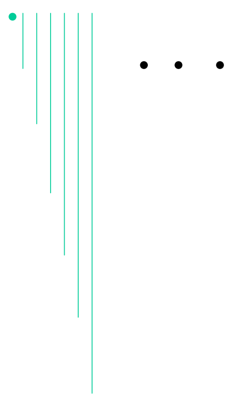
Vertical recursion

- Our arithmetical objects are U* (not U*V)
- we can develop arithmetics for every U (with sum only, because we do not have projections)

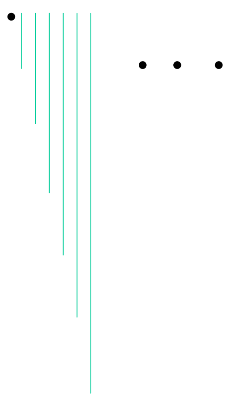


$$\begin{cases} +\,0 = id \\ +\,s_U U^* = s_U\,U+ \end{cases}$$

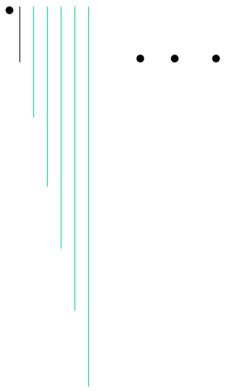
Example: a*



Example: a*

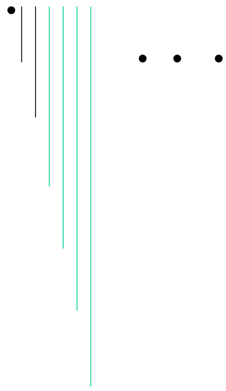


Example: a*



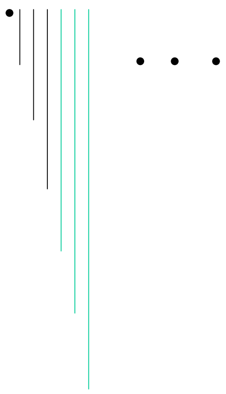
25

Example: a*



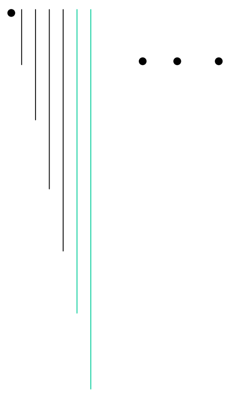
26

Example: a*



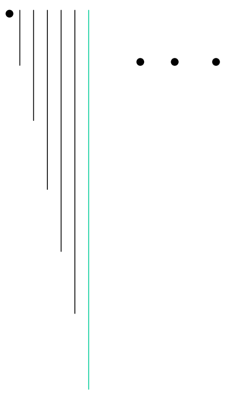
27

Example: a*



28

Example: a*



29

Horizontal and Vertical Recursion

- Horizontal recursion counts adjacent copies of a given shape K in a behaviour
- Vertical U-recursion counts loops of the same shape U in a behaviour

30

Considerations

- Distributivity plays a crucial role
- Strong idempotency of sum $1^* = 1$ would destroy everything.
- P(A*) – the algebra of languages - does not enjoys the presented properties
- All properties are concerned with *regular objects

31

The category Tree*

- Objects: right linear hierarchical systems (rlhs) of equations that correspond to regular expressions
- Morphisms: path preserving correspondences between variables in rlhs

32

Unfolding to compose

$x = Qx + P$

has the same initial solution as

$$\begin{cases} x = Qx' + P \\ x' = Qx' + P \end{cases}$$

33

Examples

A= {a}

a* : $u = a\,u + 1$

N=1*a* : $\begin{cases} n = n + u \\ u = a\,u + 1 \end{cases}$ or $\begin{cases} n = n' + u \\ n' = n' + u \\ u = a\,u + 1 \end{cases}$

0: $a^* \to 1^*a^*$ $(u, u) = a\,(u,u) + 1$

s: $1^*a^* \to 1^*a^*$ $\begin{cases} (n, n') = (n, n') + (u, u) \\ (u, u) = a\,(u, u) + 1 \end{cases}$

34

Tree*

- Is a pretopos
- Is monoidal (not closed anymore)
- Has parametrical NNO and LNNO
- Has vertical NNO's

35

What does it mean?

- Functions defined through primitive recursion are finitely presented in Tree*
- ????

36

References

- Bénabou, J., manuscript
- Cockett, J.R.B.: List-arithmetic distributive categories: locoi. JPAA 66 (1990) 1-29.
- Corradini,F., De Nicola,R., Labella,A.: Models of Nondeterministic Regular Expressions. JCSS 59 (1999) 412 - 449.
- De Nicola,R., Labella,A.:Nondeterministic regular expressions as solutions of equational systems, TCS 302, (2003) 179-189.
- Kasangian,S., Labella,A.: Enriched Categorical Semantics for Distributed Calculi. J PAA 83, (1992) 295-321.
- Labella,A.: Categories with sums and right distributive tensor product. J PAA, (2002)
- Maietti. M.E.: The typed calculus of arithmetic universes, manuscript
- Paré, R., Romàn, L. : Monoidal Categories with Natural Numbers Object. Studia Logica, 48(3) (1989) 361-376.
- Romàn, L. Cartesian Categories with Natural Number Object. JPPA 58 (1989) 267-278.

37

Unique Parallel Decomposition

Bas Luttik

Vrije Universiteit Amsterdam

July 23, 2003

Current affiliation: *Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands*

E-mail address: *luttik@win.tue.nl*

Terminology

process theory: mathematical theory of behaviour

algebraic process theory: abstract from certain special process theoretic considerations (e.g., abstract from what a process is) and place them into a general algebraic context

Number Theory

Fundamental Theorem of Arithmetic:

Every positive natural number can be expressed as a product of prime numbers in a way that is unique up to a permutation of the primes.

Unique Decomposition

Let $(M, \otimes, \iota)$ be an arbitrary commutative monoid.

$p \in M$ is **prime** if $p \neq \iota$ and $p = x \otimes y$ implies $x = \iota$ or $y = \iota$.

If $p_1, \ldots, p_n$ are prime elements of M such that $x = p_1 \otimes \cdots \otimes p_n$, then the expression $p_1 \otimes \cdots \otimes p_n$ is a **decomposition** of x.

Decompositions $p_1 \otimes \cdots \otimes p_m$ and $q_1 \otimes \cdots \otimes q_n$ of x are **equivalent** if there is a bijection $\sigma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ s.t. $p_i = q_{\sigma(i)}$.

M has **unique decomposition** if every $x \in M$ has a decomposition and every two decompositions of x are equivalent.

M has **cancellation** if $x \otimes z = y \otimes z$ implies $x = y$.

... in Process Theory

Consider the commutative monoid $\mathcal{P}/\approx$ of process expressions modulo your favourite process equivalence with parallel composition $\parallel$ as binary operation and the empty process ε as identity.

It has **unique parallel decomposition** if every process P can be expressed as a parallel composition

$$P \approx P_1 \parallel \cdots \parallel P_n$$

of *parallel primes* $P_1, \ldots, P_n$ in a way that is unique up to $\approx$ and up to a permutation of the parallel primes.

Why?

A unique parallel decomposition theorem is an indispensable tool when the parallel operator cannot be eliminated from process expressions.

Typical examples:

- decidability of bisimulation for normed BPP/PA;
- completeness of axiom system for a theory with action refinement;
- non-existence of finite axiomatisation of PA *without left-merge*;
- existence of a finite basis for PA.

Unique Parallel Decomposition

Two proof techniques:

1. **Via Cancellation**:
   - elegant
   - limited applicability;
2. **Milner's Technique**:
   - complicated
   - more powerful.

Unique Decomposition via Milner's Technique

Moller (1989):

(i) finite BCCS + *free merge* modulo strong bisimulation;

ii) finite BCCS + *composition* modulo strong bisimulation;

ii) finite BCCS + *composition* modulo weak bisimulation.

Christensen (1993):

(i) weakly normed BPP modulo strong bisimulation;

ii) weakly normed BPP$_\tau$ modulo *strong* bisimulation.

Overview

1. introduction
2. ACP$_\varepsilon$ and bisimulation
3. inventarise complications in ACP$_\varepsilon$ w.r.t. unique decomposition
4. identify subset of ACP$_\varepsilon$-expressions modulo bisimulation for which Milner's Technique might apply
5. 'axiomatise' Milner's Technique and show that it can be applied
6. open problems/future directions

ACP$_\varepsilon$ — syntax

Let $\mathcal{A}$ be a set of **actions**.

Let $\gamma : \mathcal{A} \times \mathcal{A} \rightharpoonup \mathcal{A}$ be a **communication function**.

Let $\mathcal{V}$ be a set of **process variables**.

Suppose that $a \in \mathcal{A}$, $\mathcal{H} \subseteq \mathcal{A}$, $X \in \mathcal{V}$.

The set of **process expressions** $\mathcal{P}$ is generated by

$$P ::= \varepsilon \mid \delta \mid a \mid X \mid P \cdot P \mid P + P \mid \partial_{\mathcal{H}}(P) \mid P \mathbin{\|\!\_} P \mid P \mid P \mid P \parallel P.$$

We presuppose a **guarded recursive specification** $\mathcal{S}$.

ACP$_\varepsilon$ — operational semantics

$$\frac{}{\varepsilon\downarrow} \quad \frac{P\downarrow,\ Q\downarrow}{(P\cdot Q)\downarrow} \quad \frac{P\downarrow}{(P+Q)\downarrow,\ (Q+P)\downarrow} \quad \frac{P\downarrow,\ Q\downarrow}{(P\parallel Q)\downarrow,\ (Q\parallel P)\downarrow} \quad \frac{P\downarrow,\ Q\downarrow}{(P\mid Q)\downarrow} \quad \frac{P\downarrow}{\partial_{\mathcal{H}}(P)\downarrow}$$

$$\frac{}{a \xrightarrow{a} \varepsilon} \quad \frac{P \xrightarrow{a} P'}{P\cdot Q \xrightarrow{a} P'\cdot Q} \quad \frac{P\downarrow,\ Q \xrightarrow{a} Q'}{P\cdot Q \xrightarrow{a} Q'} \quad \frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P',\ Q+P \xrightarrow{a} P'}$$

$$\frac{P \xrightarrow{a} P'}{P\parallel Q \xrightarrow{a} P'\parallel Q,\ Q\parallel P \xrightarrow{a} Q\parallel P'} \quad \frac{P \xrightarrow{b} P',\ Q \xrightarrow{c} Q',\ a=\gamma(b,c)}{P\parallel Q \xrightarrow{a} P'\parallel Q'}$$

$$\frac{P \xrightarrow{a} P'}{P \mathbin{\|\!\_} Q \xrightarrow{a} P'\parallel Q} \quad \frac{P \xrightarrow{b} P',\ Q \xrightarrow{c} Q',\ a=\gamma(b,c)}{P\mid Q \xrightarrow{a} P'\parallel Q'}$$

$$\frac{P \xrightarrow{a} P',\ [X \stackrel{\text{def}}{=} P] \in \mathcal{S}}{X \xrightarrow{a} P'} \quad \frac{P \xrightarrow{a} P',\ a \notin \mathcal{H}}{\partial_{\mathcal{H}}(P) \xrightarrow{a} \partial_{\mathcal{H}}(P')}$$

ACP$_\varepsilon$ — bisimulation

A **bisimulation** is a *symmetric* binary relation $\mathcal{R}$ on $\mathcal{P}$ such that $P\ \mathcal{R}\ Q$ implies

(i) if $P\downarrow$, then $Q\downarrow$; and

ii) if $P \xrightarrow{a} P'$, then there exists Q' such that $Q \xrightarrow{a} Q'$ and $P'\ \mathcal{R}\ Q'$.

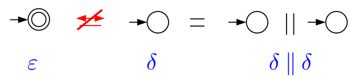
P and Q are said to be **bisimilar** (notation: $P \leftrightarrow Q$) if there exists a bisimulation $\mathcal{R}$ such that $P\ \mathcal{R}\ Q$.

Question

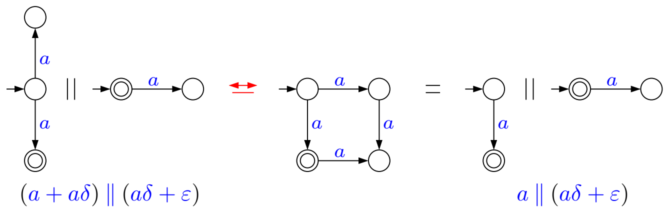
Does Milner's Technique generalise to ACP$_\varepsilon$?

Problem 1: the distinction between ε and δ

There are finite processes without a decomposition:



There are finite processes with two distinct decompositions:



Weakly normed ACP$_\varepsilon$

If $w \in \mathcal{A}^*$, say $w = a_1, \ldots, a_n$, write $P \xrightarrow{w} Q$ for $P \xrightarrow{a_1} \cdots \xrightarrow{a_n} Q$.

P is **weakly normed** iff

$P \xrightarrow{w} Q \leftrightarrow \varepsilon$ for some Q and $w \in \mathcal{A}^*$.

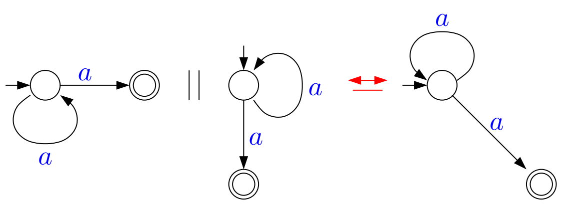
(e.g., a, $a + a\delta$ and $a + \varepsilon$ are weakly normed, but $a\delta + \varepsilon$ is not)

Does Milner's Technique generalise to weakly normed ACP$_\varepsilon$?

Problem 2: liberal communication mechanism

Suppose that $\gamma(a, a) = a$ and $X \stackrel{\text{def}}{=} aX + a$.
Note that X is weakly normed.

$X \not\leftrightarrow \varepsilon$ and X is not prime for we have $X \leftrightarrow X \parallel X$:



So X cannot have a unique decomposition!

ACP$_\varepsilon$ with bounded communication

If there exists $\ell : \mathcal{A} \to \mathbf{N} - \{0\}$ such that for all a, b and c

$$a = \gamma(b, c) \implies \ell(a) = \ell(b) + \ell(c),$$

then we say that γ **bounded**.

(Note: **handshaking** is the special case where $\ell : \mathcal{A} \to \{1, 2\}$)

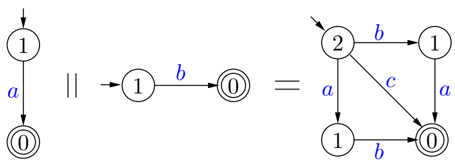
Extend ℓ to $\mathcal{A}^*$ as follows:

$$\ell(\lambda) = 0;$$
$$\ell(wa) = \ell(w) + \ell(a).$$

Define $|P| = \min\{\ell(w) \mid \exists Q.\ P \xrightarrow{w} Q \leftrightarrow \varepsilon\}$.

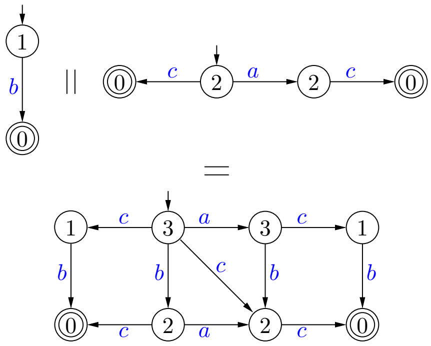
Example

Let $\gamma(a, b) = c$, $\ell(a) = \ell(b) = 1$ and $\ell(c) = 2$ in



Example

Let $\gamma(a, b) = c$, $\ell(a) = \ell(b) = 1$ and $\ell(c) = 2$ in



Claim

Milner's Technique generalises to weakly normed ACP_ε
provided that the communication function γ is bounded!

Positively ordered monoids (1)

A **positively ordered monoid** (**p.o. monoid**) is a nonempty set M together with

(i) an associative binary operation $\otimes$ on M;

(ii) a two-sided identity element ι for $\otimes$;

(iii) a partial order $\preccurlyeq$ on M that is *compatible* with $\otimes$, i.e.,

if $x \preccurlyeq y$, then $x \otimes z \preccurlyeq y \otimes z$ and $z \otimes x \preccurlyeq z \otimes y$,

and for which ι is the *least element*, i.e., $\iota \preccurlyeq x$ for all x.

M is **commutative** if $x \otimes y = y \otimes x$.

Examples

Example 1: the set $\mathbf{N}$ of natural numbers with $+$, 0 and $\leq$ is a commutative p.o. monoid.

Example 2: the set $\mathbf{N} - \{0\}$ of positive natural numbers with $\cdot$, 1 and $|$ is a commutative p.o. monoid.

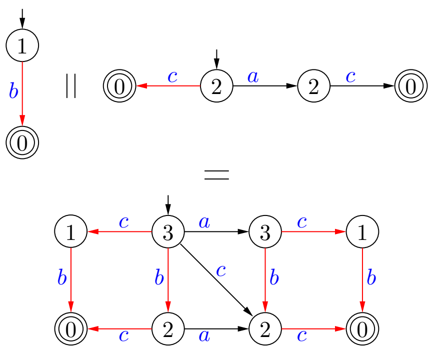
Example 3: let $\mathcal{P}^\varepsilon$ be the set of weakly normed ACP_ε-expressions; define on $\mathcal{P}^\varepsilon$ a binary relation $\rightarrowtail$ by

$$P \rightarrowtail Q \iff \exists a \in \mathcal{A}.\ P \xrightarrow{a} Q \ \&\ |P| = |Q| + \ell(a);$$

$\mathcal{P}^\varepsilon/\leftrightarrow$ is a commutative p.o. monoid with the partial order induced on it by $\rightarrowtail^*$.

Example

Let $\gamma(a, b) = c$, $\ell(a) = \ell(b) = 1$ and $\ell(c) = 2$ in



Stratification

A **stratification** of a p.o. monoid M is a *strict homomorphism*

$$|\_| : M \to \mathbf{N}.$$

So: $|x \otimes y| = |x| + |y|$, $|\iota| = 0$, and if $x \prec y$, then $|x| < |y|$.

A **stratified** p.o. monoid is a p.o. monoid M together with a stratification $|\_| : M \to \mathbf{N}$.

The natural number $|x|$ assigned to $x \in M$ is called the **norm** of x.

Examples

Example 1: $\mathrm{id}_{\mathbf{N}}$ is a stratification of $\mathbf{N}$ with $+$, 0 and $\leq$.

Example 2: for $\mathbf{N}-\{0\}$ with $\cdot$, 1 and $|$, define $|\_| : \mathbf{N}-\{0\} \to \mathbf{N}$ by

$$|k| = \max\{n \geq 0 : \exists k_0 < \cdots < k_n(1 = k_0 \mid k_1 \mid \cdots \mid k_n = k)\}.$$

Example 3: the mapping

$$|\_| : \mathcal{P}^{\varepsilon} \to \mathbf{N}$$

induces a stratification on $\mathcal{P}^{\varepsilon}/\underline{\leftrightarrow}$.

Precompositionality

A p.o. monoid M is **precompositional** if

$$x \preccurlyeq y \otimes z \Rightarrow \text{there exist } y' \preccurlyeq y \text{ and } z' \preccurlyeq z \text{ such that } x = y' \otimes z'.$$

Example 1: it is easy to see that $\mathbf{N}$ is precompositional.

Example 2: to prove that $\mathbf{N}-\{0\}$ is precompositional use that $p \mid k \cdot l$ implies $p \mid k$ or $p \mid l$ for every prime number p.

Example 3: that $\mathcal{P}^{\varepsilon}/\underline{\leftrightarrow}$ is precompositional follows since $P \parallel Q \rightarrowtail^{*} R$ implies that there exist P' and Q' such that

$$P \rightarrowtail^{*} P',\ Q \rightarrowtail^{*} Q' \text{ and } R = P' \parallel Q'.$$

Main Results

Theorem: In a stratified and precompositional commutative p.o. monoid every element has a unique decomposition.

Proof: abstract version of Milner's Technique.

Corollaries: In each of the monoids $\mathbf{N}$, $\mathbf{N}-\{0\}$ and $\mathcal{P}^{\varepsilon}/\underline{\leftrightarrow}$ every element has a unique decomposition.

Open problems/future directions (1)

Test applicability of result, e.g., for

1. branching bisimulation semantics
   a natural candidate for the order seems to be $(\Longrightarrow \rightarrowtail \Longrightarrow)^{*}$ (where $\rightarrowtail$ may only refer to a τ-step if it is not inert)
   hard part: establishing that if $P \xrightarrow{\tau} P'$ is not inert then $P \parallel Q \xrightarrow{\tau} P' \parallel Q$ is not inert either
2. other equivalences in the linear time/branching time spectrum (there are counterexamples for all equivalences below possible worlds semantics)

Ad 1. (added September 1, 2003): unique decomposition fails for the commutative monoid of weakly normed ACP$^{\varepsilon}$-expressions modulo branching bisimulation: if $X \stackrel{\text{def}}{=} aX + \tau$, then $X \,\underline{\leftrightarrow}_b\, X \parallel X$.

Open problems/future directions (2)

Analyse the equational theory of ACP with handshaking.

Conjecture: If γ is 'handshaking', then the equational theory of the algebra of ACP-expressions modulo bisimulation is not finitely based.

We want to prove this by showing that for every finite set of equations E that are sound with respect to bisimulation there exists $n \geq 0$ such that the equation

$$((\cdots(((x_1 \mid x_2) \cdot y_1 \mathbin{|\!\!\lfloor} z_1) \cdot y_2 \mathbin{|\!\!\lfloor} z_2) \cdots) \cdot y_n \mathbin{|\!\!\lfloor} z_n) \mid x_3 = \delta$$

is not equationally derivable from E.

Preamble: Foundational Considerations

Dale Miller
INRIA/Futurs and École polytechnique

The study of process calculi depends a great deal on algebra and operational semantics (SOS, bisimulation, modal logics).

Certain new features to process calculi are a challenge for traditional algebraic and operational semantic approaches since these are based on formalisms (such as first-order quantification) that lack abstractions.

- higher-order process calculi
- mobility; name and key restrictions
- spacial logics

The algebraic approach often turns to category theory for more power:

- an active approach to semantics these days
- often seems to be "heavy lifting" with naturalness overwhelmed by technical devices.

Proof Search

The *sequent calculus* is a rich framework for representing and reasoning about many logics. Linear logic has recently helped expand the usefulness of sequent calculus for computation.

"Simple" (goal-directed and cut-free) sequent calculus proofs can be used to capture computation traces: we think of *searching for proofs* in a bottom-up fashion. This approach to computation is often called **proof search**, in contrast to **proof reduction**, a foundations of functional programming.

If *cut-elimination* holds (that is, lemmas can be in-lined), the various elements of expressiveness fit together modularly (no ugly feature interactions).

The treatment of various abstractions are standard and well understood.

Implementation issues are generally clear and prototype implementations are often easy using existing tools (Isabelle, λProlog, Twelf, ...).

A logic might have a "good" proof system with many structural properties and still lack simple, transparent model theoretic semantics (eg, higher-order linear logic).

Two approaches to process calculus in proof search

First: Processes-as-formula.

- Generally requires a "resource sensitive logic" like linear logic.
- Map process combinators to logical connectives: for example, parallel composition $|$ is mapped to the $\parr$ or $\otimes$ in linear logic.
- There are not many examples known, but it's too exciting not to explore. (The rest of this talk includes such an example.)
- *May* behavior is typically all that is captured: $\vdash A$ means that "there exists a proof of A" and this encodes "there is a computation trace of A".
- *Must* behavior needs to be looked at more closely in proof theory.
- Little choice of equivalences to consider: the logical equivalence of formulas is the finest equivalence that applies to processes.

Two approaches to process calculus in proof search

Second: Processes-as-terms (within a logic). The general and common setting.

- Choose your predicates (relations over processes, actions, etc) and the logic to embed it into (Horn clauses, ...).
- SOS: Main predicate encodes one-step transitions.

$$P|Q \xrightarrow{\tau} P'|Q' \;\subset\; \exists A[P \xrightarrow{A} P' \wedge Q \xrightarrow{\bar{A}} Q'].$$

- Modal Logic/HML: Add also the satisfies relation:

$$P \models \langle a \rangle A \;\subset\; \exists P'[P \xrightarrow{a} P' \wedge P' \models A].$$

Side-conditions should not be captured with additional predicates but with expressive elements of the surrounding logic.

Linear logic is valuable for making SOS more expressive and declarative.

Miller & Tui [LICS 03] introduced the ∇ quantifier as a more expressive element of logic: as a result, π-calculus can be specified with no side conditions.

Encryption as an Abstract Datatype:

observations about relating security protocols and proof search in linear logic

Dale Miller
INRIA/Futurs and École polytechnique

Outline

1. Security protocols specified using multisets rewriting.
2. Eigenvariables for nonces and session keys.
3. Encrypted data as an abstract datatype.
4. Protocols as linear logic theories.
5. Tests, traces, and interpolants.

A Typical Protocol Specification

The following is a presentation of the Needham-Schroeder Shared Key Protocol. Alice and Bob make use of a trusted server to help them establish their own private channel for communications.

Message 1	$A \longrightarrow S$: A, B, n_A
Message 2	$S \longrightarrow A$: $\{n_A, B, k_{AB}, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$
Message 3	$A \longrightarrow B$: $\{k_{AB}, A\}_{k_{BS}}$
Message 4	$B \longrightarrow A$: $\{n_B\}_{k_{AB}}$
Message 5	$A \longrightarrow B$: $\{n_B - 1\}_{k_{AB}}$

Here, A, B, and S are agents (Alice, Bob, server), and the k's are encryption keys, and the n's are nonces.

One of our goals is to replace this specific syntax with one that is based on a direct use of logic. We will then investigate if logic's meta-theory can help in reasoning about security.

Motivating a more declarative specification

The notation $A \longrightarrow B\colon M$ seems to indicate a "three-way synchronization," but communication here is asynchronous: Alice put a message on in a network and Bob picks it up from the network. An intruder might read/delete/modify the message.

A better syntax might be:

$$\begin{array}{rcl} A & \longrightarrow & A' \mid \mathsf{N}(M) \\ B \mid \mathsf{N}(M) & \longrightarrow & B' \\ & \vdots & \\ E \mid \mathsf{N}(M) & \longrightarrow & E' \mid \mathsf{N}(M) \end{array}$$

More generally,

$$(A\ \mathit{Memory}) \mid \mathsf{N}(M_1) \mid \cdots \mid \mathsf{N}(M_p) \longrightarrow (A'\ \mathit{Memory}') \mid \mathsf{N}(P_1) \mid \cdots \mid \mathsf{N}(P_q)$$

where $p, q \geq 0$. The agent can be missing from the left (agent creation) or can be missing from the right (agent deletion).

This is essentially a specification of *multiset rewriting* of atomics formulas.

Dynamic creation of new symbols

New symbols representing nonces (used to help guarantee "freshness") and new keys for encryption and session management are needed also in protocols. We could introduced syntax such as:

$$a_1\ S \longrightarrow \mathit{new}\ k.\ a_2\ \langle k, S\rangle \mid \mathsf{N}(\{M\}_k)$$

This *new* operator looks a bit like a quantifier: it should support α-conversion and seems to be a bit like reasoning generically. The scope of *new* is over the body of this rule.

Static distribution of keys

Consider a protocol containing the following messages.

$$\begin{array}{ll} \vdots & \\ \text{Message } i & A \longrightarrow S\colon \{M\}_k \\ \text{Message } j & S \longrightarrow A\colon \{P\}_k \\ \vdots & \end{array}$$

How can we declare that a key, such as k, is only built into two specific agents. This static declaration is critical for modularity and for establishing correctness later. A **local** declaration can be used (borrowed from λProlog).

$$\begin{array}{l} \vdots \\ \mathit{local}\ k.\ \left\{\begin{array}{rl} A \longrightarrow & A' \mid\ \mathsf{N}(\{M\}_k) \\ S \mid\ \mathsf{N}(\{P\}_k) \longrightarrow & S' \end{array}\right\} \\ \vdots \end{array}$$

This declarations also appears to be similar to a quantifier.

Are these specifications logical expressions?

Can we view the symbols we have introduced as logical connectives?

	$\mid$	$\longrightarrow$	*new*	*local*	*empty*
disjunctive (Forum)	$\parr$	$\multimapinv$	$\forall$	$\exists$	$\bot$
conjunctive (MSR)	$\otimes$	$\multimap$	$\exists$	$\forall$	$\mathbf{1}$

The disjunctive approach allows protocols to be seen as **abstract logic programs**: that is, it fits into the "logic programming as goal-directed search" paradigm.

Note: Logic is not used here to form judgments *about* protocol. Rather, elements of logic are elements of the protocol.

MSR: Cervesato, Durgin, Lincoln, Mitchell, Scedrov. "A Meta-Notation for Protocol Analysis," Proceedings of the 12th IEEE Computer Security Foundations Workshop IEEE Computer Society Press, 1999.

Encrypted data as an abstract data type

Encryption keys are encoded as symbolic functions on data of type $\mathit{data} \rightarrow \mathit{data}$.

Replace $\{M\}_k$ with $(k\ M)$.

By providing scope to such keys, encrypted data forms an abstract datatype.

To insert an encryption key into data, we will use the postfix coercion constructor $(\cdot)^\circ$ of type $(\mathit{data} \rightarrow \mathit{data}) \rightarrow \mathit{data}$.

The use of higher-order types means that we will also use the equations of $\alpha\beta\eta$-conversion (a well studied extension to logic programming with robust implementations).

$$\exists\ k.\ \left[\begin{array}{rl} a_1\ S \multimapinv & \forall n.\ a_2\ \langle k^\circ, S\rangle \parr \mathsf{N}(k\ n) \\ a_2\ \langle k^\circ, S\rangle \parr \mathsf{N}(k\ M) \multimapinv & \ldots \end{array}\right]$$

A Linear Logic Specification of Needham-Schroeder

$$\begin{array}{llll} \exists k_{as} \exists k_{bs} \{ & & & \\ \quad a\ S & \multimapinv & \forall na. & a_1\ \langle na, S\rangle \parr \mathsf{N}(\langle a, b, na\rangle). \\ \quad a_1\ \langle N, S\rangle \parr \mathsf{N}(k_{as}\langle N, b, K, En\rangle) & \multimapinv & & a_2\ \langle N, K, S\rangle \parr \mathsf{N}(En). \\ \quad a_2\ \langle Na, Key^\circ, S\rangle \parr \mathsf{N}(Key\ Nb) & \multimapinv & & a_3\ \langle\rangle \parr \mathsf{N}(Key\ \langle Nb, S\rangle). \\ \quad b\ \langle\rangle \parr \mathsf{N}(k_{bs}\ \langle Key^\circ, a\rangle) & \multimapinv & \forall nb. & b_1\ \langle nb, Key^\circ\rangle \parr \mathsf{N}(Key\ nb). \\ \quad b_1\ \langle Nb, Key\rangle \parr \mathsf{N}(Key\langle Nb, S\rangle) & \multimapinv & & b_2\ S. \\ \quad s\ \langle\rangle \parr \mathsf{N}(\langle a, b, N\rangle) & \multimapinv & \forall k. & s\ \langle\rangle \parr \mathsf{N}(k_{as}\langle N, b, k^\circ, k_{bs}\langle k^\circ, a\rangle\rangle). \\ \} & & & \end{array}$$

Outermost universal quantifiers around individual clauses have not been written but are assumed for variables (tokens starting with a capital letter).

Relating implementation and specification

A property of NS should be that Alice can communicate to Bob a secret with the help of a server. That is, the clause

$$\forall x\ (a\ \langle x\rangle \parr b\ \langle\rangle \parr s\ \langle\rangle \multimapinv a_3\ \langle\rangle \parr b_2\ \langle x\rangle \parr s\ \langle\rangle)$$

can be seen as part of the specification of this protocol.

If we call the above clause *SPEC* and the formula for Needham-Schroeder *NS*, then it is a simple calculation to prove that $NS \vdash SPEC$ in linear logic.

Of course, a kind of converse is more interesting and harder. At least a trivial thing is proved trivially.

Should not logical entailment be a center piece of logical specifications?

Automation of proof search

Automation of proof search must not involve "invention". The subformula property is a good guide.

- Lemmas should not be automated: i.e., consider only cut-free proofs.
- Higher-order predicates substitutions must be "tame"; no automation of *invariants*.

Cuts can be avoided since linear logics satisfies the *cut elimination property*.

Higher-order substitutions have traditionally been avoided by restricting to first-order. But this is too draconian! It makes impossible admitting rich forms of abstractions (e.g., higher-order programming, abstract datatypes, HOAS, etc).

Not all higher-order quantification is hard to automate and even the most simple forms can be a great asset when *reasoning about* logic programs.

Quantification rules

There are two ways $\forall$ is used in a proof: To prove a $\forall$, prove a generic instance of it. To use a $\forall$ assumption, make any instance of it.

$$\frac{\Sigma \vdash t : \tau \qquad \Sigma : \Delta, B[t/x] \longrightarrow \Gamma}{\Sigma : \Delta, \forall_\tau x.B \longrightarrow \Gamma}\ \forall\mathcal{L} \qquad \frac{y : \tau, \Sigma : \Delta \longrightarrow B[y/x], \Gamma}{\Sigma : \Delta \longrightarrow \forall_\tau x.B, \Gamma}\ \forall\mathcal{R}$$

If a $\forall$ appears on the right (positively), then replace it with a new constant. Even in the higher-order setting, this is a trivial operation.

If a $\forall$ appears on the left (negatively), then replace with some substitution term. The choice of substitution term is generally determined using unification.

Dual statements can be made for the $\exists$ quantifier.

Scheme for reasoning about logic programs

Higher-order quantification in this talk will be featured in two ways.

- During computation (proof search) higher-order quantification will be "easy": e.g., generate a new symbol at higher-order type.
- When reasoning about computations, the dual operation of instantiating new symbols with clever substitutions might be necessary.

One approach to reasoning about logic programs is the following:

$$\frac{P \vdash G \quad \text{proof search (cut-free, automated)} \qquad P' \vdash P \quad \text{reasoning about programs: involves rich substitutions and lemmas}}{P' \vdash G \quad \text{after cut-elimination (lemma removal), we have a computation again}}$$

Notice that P has appeared both positively and negatively in these examples. What corresponded to "generate a new predicate" dualizes to "find a logical expression to substitution" when the polarity is shifted.

Can't we compile away higher-order quantification?

If we simply execute security protocols, then the expressions $\{M\}_k$ and $(k\ M)$ can be compiled as first-order expressions such as

$$(apply\ k\ M), \quad \text{or more appropriately, as} \quad (encrypt\ k\ M).$$

In order to reason about such a protocol, we need to explain the meaning of this new non-logical constant. This complicates the reasoning process somewhat.

Lesson: Do not leave the paradise of Church too soon.

A simple logical equivalence

Consider the following two clauses:

$$a \multimapinv \forall k.\mathsf{N}(k\ m) \quad \text{and} \quad a \multimapinv \forall k.\mathsf{N}(k\ m').$$

These two clauses show that Alice can take a step that generates a new encryption key and then outputs either the message m or m' in encrypted form. These two clauses seem "observationally similar".

More surprisingly

$$a \multimapinv \forall k.\mathsf{N}(k\ m) \dashv\vdash a \multimapinv \forall k.\mathsf{N}(k\ m').$$

That is, they are logically equivalent! In particular, the sequent

$$\forall k.\mathsf{N}(k\ m) \longrightarrow \forall k.\mathsf{N}(k\ m')$$

is proved by using the eigenvariable c on the right and the term $\lambda w.(c\ m')$ on the left.

More logical equivalences

If we allow local ($\exists$) abstractions of predicates, then other more interesting logical equivalences are possible.

For example, 3-way synchronization can be implemented using 2-way synchronization with a hidden intermediary.

$$\exists\, x. \left\{ \begin{array}{l} a \parr b \mathrel{\circ\!-} x \\ x \parr c \mathrel{\circ\!-} d \parr e \end{array} \right\} \quad \dashv\vdash \quad a \parr b \parr c \mathrel{\circ\!-} d \parr e$$

Intermediate states of an agent can be taken out entirely.

$$\exists\, a_2, a_3. \left\{ \begin{array}{l} a_1 \parr \mathsf{N}(m_0) \mathrel{\circ\!-} a_2 \parr \mathsf{N}(m_1) \\ a_2 \parr \mathsf{N}(m_2) \mathrel{\circ\!-} a_3 \parr \mathsf{N}(m_3) \\ a_3 \parr \mathsf{N}(m_4) \mathrel{\circ\!-} a_4 \parr \mathsf{N}(m_5) \end{array} \right\} \quad \dashv\vdash$$

$$a_1 \parr \mathsf{N}(m_0) \mathrel{\circ\!-} (\mathsf{N}(m_1) \mathrel{\circ\!-} (\mathsf{N}(m_2) \mathrel{\circ\!-} (\mathsf{N}(m_3) \mathrel{\circ\!-} (\mathsf{N}(m_4) \mathrel{\circ\!-} (\mathsf{N}(m_5) \parr a_4)))))$$

This suggests an alternative syntax for agents.

Needham-Schroeder revisited

$\exists k_{as} \exists k_{bs}.[$

(Out) $\forall na.\mathsf{N}(\langle alice,\ bob,\ na\rangle) \mathrel{\circ\!-}$

(In) $(\forall Kab \forall En.\mathsf{N}(kas\langle na,\ bob,\ Kab^\circ,\ En\rangle) \mathrel{\circ\!-}$

(Out) $(\mathsf{N}(En) \mathrel{\circ\!-}$

(In) $(\forall Nb.\mathsf{N}(Kab\ Nb) \mathrel{\circ\!-}$

(Out) $\mathsf{N}(Kab(Nb,\ secret))))).$

(Out) $\bot \mathrel{\circ\!-}$

(In) $(\forall Kab.\mathsf{N}(kbs(Kab^\circ,\ alice)) \mathrel{\circ\!-}$

(Out) $(\forall nb.\mathsf{N}(Kab\ nb) \mathrel{\circ\!-}$

(In) $(\forall S.\mathsf{N}(Kab(nb,\ S)) \mathrel{\circ\!-}$

(Cont) $b\ S))).$

(Out) $\bot \mathrel{\circ\!-}$

(In) $(\forall N.\mathsf{N}(\langle alice,\ bob,\ N\rangle) \mathrel{\circ\!-}$

(Out) $(\forall key.\mathsf{N}(kas\langle N,\ bob,\ key^\circ,\ kbs(key^\circ,\ alice)\rangle))).$

$]$

Two classes of connectives

The logical connectives of linear logic can be classified as

asynchronous $\bot$, $\parr$, $\forall$, ... The right introduction rules for these are invertible. These rules yield structural equivalences (internal reorganizations).

synchronous $\mathbf{1}$, $\otimes$, $\exists$, ... The right introduction rules for these are not invertible. These rules yield interaction with the environment.

These connectives are de Morgan duals of each other. For example, if an asynchronous connectives appears on the left of the sequent arrow, it acts synchronously.

We shall only write asynchronous connectives but write them on both sides of the sequent arrow (yielding both behaviors). We also use implications:

$$B \multimap C \equiv B^\perp \parr C \qquad \text{and} \qquad B \Rightarrow C \equiv\, !B \multimap C$$

Alternation of synchronous and asynchronous connectives

A *bipolar* formula is a formula in which no asynchronous connectives is in the scope of a synchronous connective. That is, there is an outer layer of asynchronous connectives followed by an inner layer of synchronous connectives.

The multiset rewriting clauses are bipolars, for example,

$$a \parr b \mathrel{\circ\!-} c \parr d \equiv a \parr b \parr (c^\perp \otimes d^\perp).$$

Andreoli showed how to compile arbitrary alternation of syn/asyn connectives into bipolars by introducing new predicate symbols. He also argued for only using bipolars for proof search.

Avoiding bipolars has some advantages

Only one predicate is need, namely, $\mathsf{N}(\cdot)$. The other predicates (used as "line numbers" in a protocol) are not needed.

The scope of variables within a formula encodes an agent's memory.

Agents now look much more like process calculus expressions with input and output prefixes. The formula $a \mathrel{\circ\!-} (b \mathrel{\circ\!-} (c \mathrel{\circ\!-} (d \mathrel{\circ\!-} k)))$ can denote either

$$\bar{a} \,||\, (b.\ (\bar{c} \,||\, (d.\ \ldots))) \quad \text{or} \quad a.\ (\bar{b} \,||\, (c.\ (\bar{d} \,||\, \ldots)))$$

depending on if it appears on the right or the left of the sequent arrow. Writing it and its negation without linear implications:

$$a \parr (b^\perp \otimes (c \parr (d^\perp \otimes \ldots))) \quad \text{resp,} \quad a^\perp \otimes (b \parr (c^\perp \otimes (d \parr \ldots)))$$

Value passing, name generation, and scope extrusion (ie, dynamic distribution of nonces and keys) are modelled by using quantifiers.

There is a strict alternation of input and output phases. If an agent skips a phase, the adjacent phases can be merged:

$$a \mathrel{\circ\!-} (\bot \mathrel{\circ\!-} (b \mathrel{\circ\!-} k)) \equiv (a \parr b) \mathrel{\circ\!-} k.$$

The general setting for specifying agents

Let A denote atomic formulas. Consider

$$H = A \mid \bot \mid H \parr H \mid \forall x.\ H \qquad \text{(heads)}$$

$$K = H \mid H \mathrel{\circ\!-} K \mid \forall x.\ K \qquad \text{(agents)}$$

Let $\mathcal{A}$ denote a multiset of atoms (ie, network messages). Let Γ and Δ be a multiset of "agents" (K-formulas), such that those in Γ are in output mode and those in Δ are in output mode.

The sequent $\Delta \longrightarrow \Gamma, \mathcal{A}$ captures the relationship between these three elements (nework messages are degenerated output processes).

The rules for implication introduction provide the basic dynamics:

$$\frac{H \longrightarrow \mathcal{A}_1 \qquad \Delta \longrightarrow K, \mathcal{A}_2}{\Delta, H \mathrel{\circ\!-} K \longrightarrow \mathcal{A}_1, \mathcal{A}_2} \qquad \frac{\Delta, K \longrightarrow \Gamma, H, \mathcal{A}}{\Delta \longrightarrow H \mathrel{\circ\!-} K, \Gamma, \mathcal{A}}$$

Left-introduction can be limited to sequents with atomic left-hand sides.

If in the definition of K-formulas above we write $H \mathrel{\circ\!-} H$ instead of $H \mathrel{\circ\!-} K$, we are restricting ourselves to MSR (bipolars) again.

Intruders, Testing, and Interpolants

One approach to characterizer *intruders*, called *tests* here, is to say that they are essentially the same things as principles, except that they can halt a computation, using with the $\top$ logical connective:

$$W ::= \top \mid H \mid H \multimap W \mid \forall x.W.$$

P and Q are *testing equivalent* if for every multiset Γ of testers, $\vdash P, \Gamma$ iff $\vdash Q, \Gamma$.

Interpolants can be used to monitor communications across a boundary.

Classically, if $A \vdash B$ then an interpolant is a formula R such that $A \vdash R$ and $R \vdash B$ and all the non-logical constants in R occur in both A *and* B.

Interpolation Theorem. Let Γ be a set of role formulas (principals) and let Δ be a set of tests (intruders) such that $\vdash \Gamma, \Delta$. There is a formula R (the interpolant) such that the non-logical constants in R occur in both Γ and in Δ and is such that $\vdash \Gamma, R$ and $R \vdash \Delta$.

Tracing Communications

The interpolants needed in this theorem have the following structure:

$$M ::= \bot \mid A \mid M \parr M$$

$$R^+ ::= \top \mid \forall x.R^+ \mid M \multimap R^- \qquad R^- ::= M \multimap R^+ \mid \forall x.R^-.$$

Formulas in the R^+ syntactic category are called *traces*. Clearly, traces are in fact simple tests.

Two processes are *trace equivalent* if for every trace R, $\vdash P, R$ iff $\vdash Q, R$.

Theorem. Trace equivalent and testing equivalent coincide.

Proof. Since every trace is a test, the forward implication is immediate. Conversely, assume that P and Q are trace equivalent and that Γ is a set of testers such that $\vdash P, \Gamma$. By the Interpolation Theorem, we know that there is a trace R such that $\vdash P, R$ and $R \vdash \Gamma$. Since P and Q are trace equivalent, we know that $\vdash Q, R$ and by cut-elimination, we know that $\vdash Q, \Gamma$.

This theorem states that "one intruder is enough."

Conclusions

1. Linear logic can be used to specify the *execution* of security protocols.
2. Seeing encryption as an abstract datatype seems a powerful logical device to help reason about hiding information.
3. Abstraction of "continuation predicates" can transform bipolar (MSR) expressions into non-bipolar (process calculus expression) expressions.
4. Proof theoretical techniques have a use in reasoning about protocol correctness.
   (a) Cut-elimination is a basic tool.
   (b) Higher-type quantification makes protocols more declarative and offer new avenues for reasoning about protocols.
   (c) Interpolants can be used to characterize the interaction between agents and environments.
5. Related work: Sumii & Pierce, Logical relations for encryption, CSFW 2001.
6. Also: *strand spaces* seem to be simple graph-like structures definable via cut-free proofs.

Tile Systems for Process Algebras

Ugo Montanari
Dipartimento di Informatica
Università di Pisa

in collaboration with

GianLuigi Ferrari, Andrea Corradini, Fabio Gadducci, Roberto Bruni
and
David de Frutos Escrig, Reiko Heckel, Narciso Marti-Oliet, Jose Meseguer, Francesca Rossi, Vladimiro Sassone

Outline

Motivations

- Compositional: in space and time
- Concurrent: synchronous/asynchronous
- Open: instantiation/contextualization
- Distributed: graphs, rewriting, no global names
- Mobile: name generation and passing
- Abstract semantics: contexts, bisimilarity
- Typed: Curry-Howard, tiles as terms
- Higher-order: double cartesian closed
- Executable: via translation into rewriting logic

Tiles, Informally

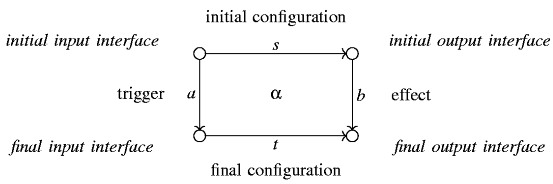


Fig. 3. A generic tile.

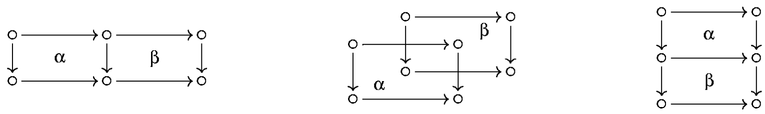


Fig. 4. Horizontal, parallel and vertical tile compositions.

Horizontal & Vertical Structure, I

The simplest case:

- Horizontal arrows: term contexts on some signature
- Vertical arrows: actions

In general:

- Horizontal arrows: suitable classes of graphs
- Vertical arrows: partial orders
- Both axiomatized via extended GS monoidal theories

Tiles as Sequents

Basic Sequents. *Generators and Identities:*

$$(gen)\ \frac{r : s \xrightarrow[b]{a} t \in R}{s \xrightarrow[b]{a} t}$$

$$(v\text{-}ref)\ \frac{a : \underline{n} \to \underline{k} \in G_{E_v}(\Sigma_v)}{id_{\underline{n}} \xrightarrow[a]{a} id_{\underline{k}}} \qquad (h\text{-}ref)\ \frac{s : \underline{n} \to \underline{m} \in \mathbf{A}_{E_h}(\Sigma_h)}{s \xrightarrow[id_{\underline{m}}]{id_{\underline{n}}} s}$$

Composed Sequents. *Parallel, Horizontal and Vertical compositions:*

$$(vert)\ \frac{s_1 \xrightarrow[b_1]{a_1} t,\ t \xrightarrow[b_2]{a_2} t_1}{s_1 \xrightarrow[b_1;b_2]{a_1;a_2} t_1}$$

$$(par)\ \frac{s_1 \xrightarrow[b_1]{a_1} t_1,\ s_2 \xrightarrow[b_2]{b_1} t_2}{s_1 \otimes s_2 \xrightarrow[b_1 \otimes b_2]{a_1 \otimes a_2} t_1 \otimes t_2} \qquad (hor)\ \frac{s_1 \xrightarrow[b]{a_1} t_1,\ s_2 \xrightarrow[b_1]{b} t_2}{s_1; s_2 \xrightarrow[b_1]{a_1} t_1; t_2}$$

GS Monoidal Theories, I

$$\frac{u \in S^*_\Sigma}{id_u : u \to u}, \qquad \frac{u, v \in S^*_\Sigma}{\rho_{u,v} : uv \to vu}, \qquad \frac{t : u \to v,\ t' : u' \to v'}{t \otimes t' : uu' \to vv'},$$

$$\frac{u \in S^*_\Sigma}{!_u : u \to \lambda}, \qquad \frac{u \in S^*_\Sigma}{\nabla_u : u \to uu}, \qquad \frac{f \in F_{u,w}}{f_\Sigma : u \to w}$$

7

GS Monoidal Theories, II

Functoriality:

$$id_{uv} = id_u \otimes id_v$$
$$(t; t_1) \otimes (t_2; t_3) = (t \otimes t_2); (t_1 \otimes t_3)$$

Monoidality:

$$t \otimes id_\lambda = t = id_\lambda \otimes t$$
$$(t \otimes t_1) \otimes t_2 = t \otimes (t_1 \otimes t_2)$$

Naturality:

$$\rho_{u_1,u}; (t \otimes t_1) = (t_1 \otimes t); \rho_{v_1,v}$$

Simmetry:

$$\rho_{\lambda,\lambda} = id_\lambda$$
$$\rho_{u,v}; \rho_{v,u} = id_{uv}$$
$$\rho_{uv.w} = (id_u \otimes \rho_{v.w}); (\rho_{u.w} \otimes id_v)$$

8

GS Monoidal Theories, III

Iterative duplication:

$$\nabla_a; (id_a \otimes \nabla_a) = \nabla_a; (\nabla_a \otimes id_a)$$

Exchange of copies:

$$\nabla_a; \rho_{a,a} = \nabla_a$$

Vacuous duplication:

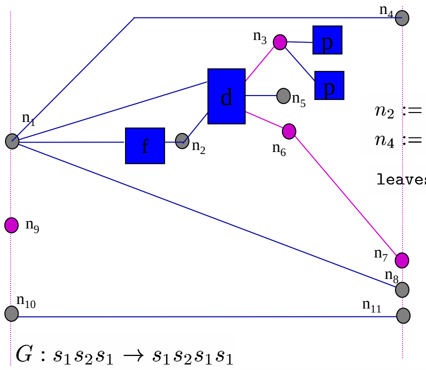
$$\nabla_a; (id_a \otimes !_a) = id_a$$

Monoidality and coherence:

$$!_{a\otimes b} = !_a \otimes !_b$$
$$\nabla_{a\otimes b} = (\nabla_a \otimes \nabla_b); (id_a \otimes \rho_{a,b} \otimes id_b)$$
$$\nabla_e = !_e = id_e$$

9

An Example



$n_2 := f(n_1),\ n_3n_5n_6 := d(n_1, n_2),\ p(n_3),\ p(n_3)$
$n_4 := n_1,\ n_7 := n_6,\ n_8 := n_1,\ n_{11} := n_{10},$

$\texttt{leaves} = (n_1, n_9, n_{10})$ $\texttt{roots} = (n_4, n_7, n_8, n_{11})$

$G : s_1 s_2 s_1 \to s_1 s_2 s_1 s_1$

$G = (\nabla_{s_1}; \nabla_{s_1} \otimes \nabla_{s_1}; id_{s_1} \otimes (id_{s_1} \otimes f; d; (\nabla_{s_2}; p \otimes p) \otimes !_{s_1} \otimes id_{s_2}) \otimes id_{s_1}) \otimes !_{s_2} \otimes id_{s_1}$

10

Horizontal & Vertical Structure, II

Additional axioms:

$$s; !_{\underline{m}} = !_{\underline{n}} \qquad s; \nabla_{\underline{m}} = \nabla_{\underline{n}}; (s \otimes s)$$

With these axioms GS monoidal theories
become algebraic theories,
i.e. they represent term contexts

11

Algebras of Connectors, I

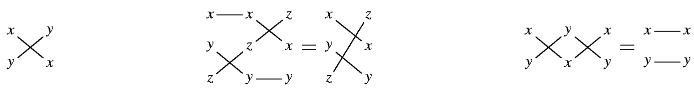


Fig. 5. Symmetries: the wire and box notation.

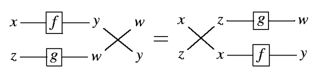


Fig. 6. Wire and box representation of the naturality axiom for symmetries.

Fig. 7. Four connectors in the wire and box notation: ∇_x, $!_x$, Δ_x and $¡_x$.

12

Algebras of Connectors, II

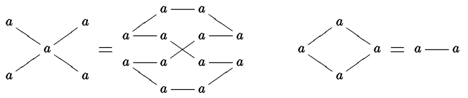


Fig. 12. The axioms $\Delta_a; \nabla_a = \nabla_{a \otimes a}; (\Delta_a \otimes \Delta_a)$ and $\nabla_a; \Delta_a = a$.

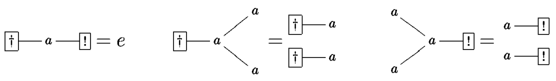


Fig. 13. The axioms $\dagger_a; !_a = e$, $\dagger_a; \nabla_a = \dagger_a \otimes \dagger_a$ and $\Delta_a; !_a = !_a \otimes !_a$.

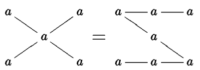


Fig. 14. The axiom $\Delta_a; \nabla_a = (\nabla_a \otimes a); (a \otimes \Delta_a)$.

Algebras of connectors, III

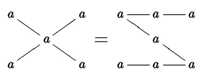
Table 1
A taxonomy of connectors.

	Sh	GS	coGS	RM	MS	NB	PM	DG	TR
∇	+	+	-	+	+	-	+	+	+
Δ	-	-	+	+	+	-	+	+	+
!	-	+	-	+	-	+	+	+	-
¡	-	-	+	+	-	+	+	+	+
⋄ = –	-	-	-	+	+	-	+	+	+
X = ⧖	-	-	-	+	+	-	+	+	+
X = Z	-	-	-	-	+	-	+	+	-
⊢⊣ = e	-	-	-	+	-	+	+	-	-
⊢< = ⊨	-	-	-	+	-	-	-	-	+
>⊣ = ⫤	-	-	-	+	-	-	-	-	-

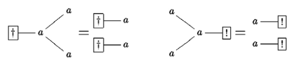
Algebras of Connectors, IV

The arrows of the following theories are:

- Symmetric monoidal theories (GS without duplicators, dischargers):
  - nonsequential processes of P/T nets
- GS monoidal theories:
  - With constructors n->1: term graphs
  - With constructors n->0: (open) graphs
- RM theories: GS, co-GS with all axioms except
  - With empty signature: relations
  - With constructors 1->1 and identities in parallel: partial orders



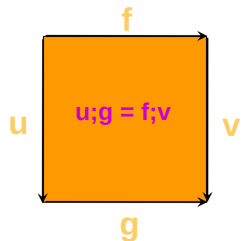
- PM theories: GS, co-GS with all axioms except
  - with empty signature: partitions



- PM monoidal with Conway axioms: iteration theories/recursion

Auxiliary Tiles and Naturality Axioms

- Auxiliary tiles
  - f, g, u and v are wires
  - Auxiliary tiles are like two-dimensional wires
  - They consist of "empty" commuting squares
  - Sometimes ALL commuting squares, other times SOME commuting squares



- Cell naturality axioms
  - Extend the one-dimensional naturality axioms

Abstract semantics

Definition 2. *Let $\mathcal{R} = (\mathcal{H}, \mathcal{V}, N, R)$ be a tile system. A symmetric relation $\sim_t$ on configurations is called* tile bisimulation *if whenever $s \sim_t t$ and $\mathcal{R} \vdash s \xrightarrow[b]{a} s'$, then there exists t' such that $\mathcal{R} \vdash t \xrightarrow[b]{a} t'$ and $s' \sim_t t'$. The maximal tile bisimulation is called* tile bisimilarity *and denoted by $\simeq_t$.*

Definition 3. *A tile system $\mathcal{R} = (\mathcal{H}, \mathcal{V}, N, R)$ enjoys the* decomposition property *if for all arrows $s \in \mathcal{H}$ and for all sequents $s \xrightarrow[b]{a} t$ entailed by $\mathcal{R}$, then: (1) if $s = s_1; s_2$ then $\exists c \in \mathcal{V}$, $t_1, t_2 \in \mathcal{H}$ such that $\mathcal{R} \vdash s_1 \xrightarrow[c]{a} t_1$, $\mathcal{R} \vdash s_2 \xrightarrow[b]{c} t_2$ and $t = t_1; t_2$; (2) if $s = s_1 \otimes s_2$ then $\exists a_1, a_2, b_1, b_2 \in \mathcal{V}$, $t_1, t_2 \in \mathcal{H}$ such that $\mathcal{R} \vdash s_1 \xrightarrow[b_1]{a_1} t_1$, $\mathcal{R} \vdash s_2 \xrightarrow[b_2]{a_2} t_2$, $a = a_1 \otimes a_2$, $b = b_1 \otimes b_2$ and $t = t_1 \otimes t_2$.*

Proposition 1 (cf. [11]). *If $\mathcal{R}$ enjoys decomposition, then $\simeq_t$ is a congruence.*

Tile Systems as Double Monoidal Categories

- Objects, horizontal arrows, vertical arrows and cells
- Horizontal 1-category: objects and horizontal arrows
- Vertical 1-category: objects and vertical arrows
- Horizontal D-category: vertical arrows and cells
- Vertical D-category: horizontal arrows and cells
- Monoidal operation on objects, horizontal arrows, vertical arrows and cells
- Any two operations of vertical, horizontal and monoidal structure commute
  - vs(hs(A) = hs(vs(A))
  - (A $;_h$ B) $;_v$ (C $;_h$ D) = (A $;_v$ C) $;_h$ (B $;_v$ D) exchange law
  - (A $;_h$ B) x (C $;_h$ D) = (A x C) $;_h$ (B x D)

Tile Systems as Coalgebras

- Coalgebras represent transition systems and bisimulations
- The kernel of the unique morphism to the final coalgebra is bisimilarity
- Bialgebras have the structure of both algebras and coalgebras
- The kernels of their arrows are both bisimulations and congruences
- Tile systems with the decomposition property can be seen as bialgebras

CCS, I

$$\frac{}{\mu.P \xrightarrow{\mu} P} \qquad \frac{P \xrightarrow{\mu} Q \;\; \mu \notin \{\alpha, \overline{\alpha}\}}{P\backslash\alpha \xrightarrow{\mu} Q\backslash\alpha}$$

$$\frac{P \xrightarrow{\mu} Q}{P+R \xrightarrow{\mu} Q} \qquad \frac{P \xrightarrow{\mu} Q}{R+P \xrightarrow{\mu} Q}$$

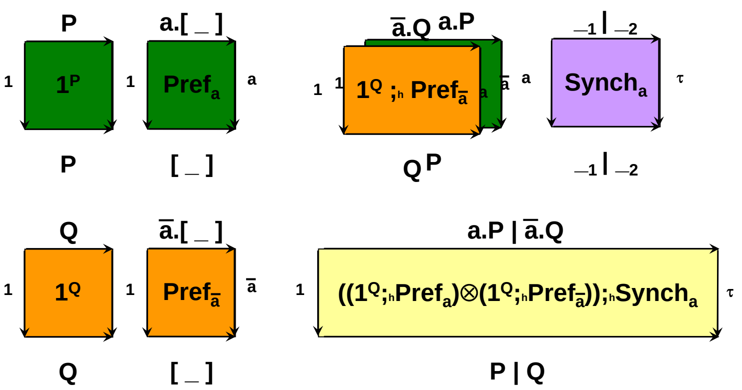
$$\frac{P \xrightarrow{\mu} Q}{P|R \xrightarrow{\mu} Q|R} \qquad \frac{P \xrightarrow{\alpha} Q,\; P' \xrightarrow{\overline{\alpha}} Q'}{P|P' \xrightarrow{\tau} Q|Q'} \qquad \frac{P \xrightarrow{\mu} Q}{R|P \xrightarrow{\mu} R|Q}.$$

$$\mathtt{Pref}_\mu : \; \mu \xrightarrow[\underline{\mu}]{\underline{1}} \underline{1} \qquad \mathtt{Res}_\mu : \; \backslash\alpha \xrightarrow[\underline{\mu}]{\underline{\mu}} \backslash\alpha \qquad \text{for} \quad \mu \notin \{\alpha, \overline{\alpha}\}$$

$$\mathtt{Suml}_\mu : \; + \xrightarrow[\underline{\mu}]{\underline{\mu}\otimes\underline{1}} \underline{1}\otimes ! \qquad \mathtt{Sumr}_\mu : \; + \xrightarrow[\underline{\mu}]{\underline{1}\otimes\underline{\mu}} ! \otimes \underline{1}$$

$$\mathtt{Compl}_\mu : | \xrightarrow[\underline{\mu}]{\underline{\mu}\otimes\underline{1}} | \qquad \mathtt{Compr}_\mu : | \xrightarrow[\underline{\mu}]{\underline{1}\otimes\underline{\mu}} | \qquad \mathtt{Synch}_\lambda : | \xrightarrow[\underline{\tau}]{\underline{\lambda}\otimes\overline{\underline{\lambda}}} | \,.$$

CCS, II



Open Process Algebras

Ordinary formats do not guarantee that bisimilarity is a congruence

C[x] = x\a|x\a and

D[x] = (x|x)\a are bisimilar (as "coordinators")

But with

p = a.nil + ā.nil

C[p] = (a.nil + ā.nil)\a | (a.nil + ā.nil)\a and D[p] = (a.nil + ā.nil | a.nil + ā.nil)\a

are not bisimilar

	$\mathcal{H}$	$\mathcal{V}$	auxiliary tiles
monoidal tile format	$\mathbf{M}[\Sigma]$	$\mathbf{M}[\Lambda]$	none
gs-monoidal tile format	$\mathbf{GS}[\Sigma]$	$\mathbf{M}[\Lambda]$	$\gamma_{a,b}$, ∇_a, $!_a$
algebraic format	$\mathbf{Th}[\Sigma]$	$\mathbf{M}[\Lambda]$	$\gamma_{a,b}$, ∇_a, $!_a$
term tile format	$\mathbf{Th}[\Sigma]$	$\mathbf{Th}[\Lambda]$	$\gamma_{a,b}$, ∇_a, $!_a$, $\gamma_{s,t}$, ∇_t, $!_t$, $\sigma_{\underline{1},\underline{1}}$, $\tau_{\underline{1}}$, $\pi_{\underline{1}}$, …

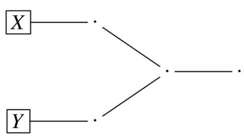
Concurrent Systems, BPP(**1**,**P**)

$$t ::= \varepsilon \mid X \mid t \parallel t$$

$$[\![\varepsilon]\!] = ¡_1$$

$$[\![t_1 \parallel t_2]\!] = ([\![t_1]\!] \otimes [\![t_2]\!]);\Delta_1$$

$[\![X \parallel Y]\!]$ =



Algebras of connectors, III

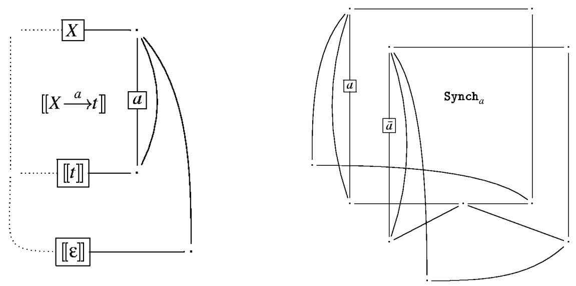
horizontal *vertical*

Table 1
A taxonomy of connectors.

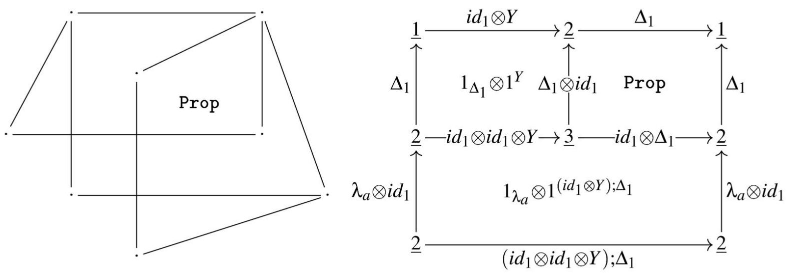
	Sh	GS	coGS	RM	MS	NB	PM	DG	TR
∇	+	+	-	+	+	-	+	+	+
Δ	-	-	+	+	+	-	+	+	+
!	-	+	-	+	-	+	+	+	-
¡	-	-	+	+	-	+	+	+	+
◇ = —	-	-	-	+	+	-	+	+	+
X = ⧖	-	-	-	+	+	-	+	+	+
X = Z	-	-	-	-	+	-	+	+	-
⊢⊣ = e	-	-	-	+	-	+	+	-	-
⊢< = ⊨	-	-	-	+	-	-	-	-	+
>⊣ = ⫤	-	-	-	+	-	-	-	-	-

Concurrent Systems, Transitions

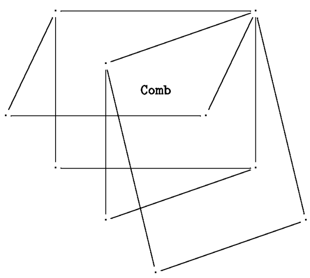
$$\frac{X \xrightarrow{a} t \in \Omega}{X \xrightarrow{a} t \in T^{\mathrm{ws}}_{\Omega}} \qquad \frac{t_1 \xrightarrow{a} t_1' \in T^{\mathrm{ws}}_{\Omega}, \;\; t_2 \xrightarrow{\bar{a}} t_2' \in T^{\mathrm{ws}}_{\Omega}}{t_1 \parallel t_2 \xrightarrow{\tau} t_1' \parallel t_2' \in T^{\mathrm{ws}}_{\Omega}} \qquad \frac{t_1 \xrightarrow{\mu} t_1' \in T^{\mathrm{ws}}_{\Omega}}{t_1 \parallel t_2 \xrightarrow{\mu} t_1' \parallel t_2 \in T^{\mathrm{ws}}_{\Omega}}$$

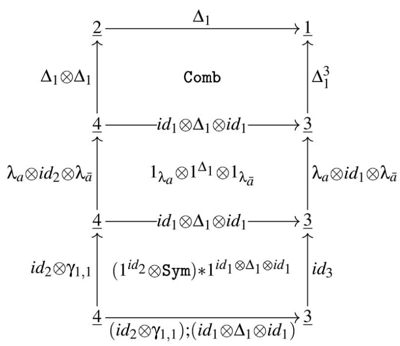


Concurrent Systems, Auxiliary Tiles, I

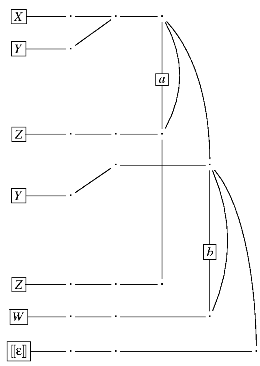


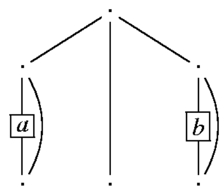
Concurrent Systems, Auxiliary Tiles, II



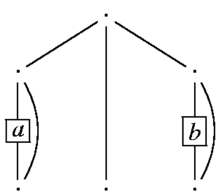


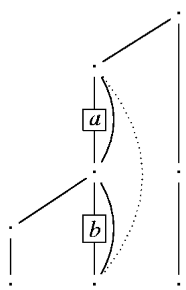
Concurrent Systems, Example, I





Concurrent Systems, Example, II





Concurrent Systems, Result

tile equivalence

Theorem 52 *Given two ordinary ws-*BPP$(\mathbf{1},\mathbf{P})$ *processes* t *and* t', *then* $[\![t]\!] \simeq_{\mathrm{t}} [\![t']\!]$ *if and only if* $t \approx_{\mathrm{w}} t'$.

Degano-Darondeau causal equivalence

Synchronized Hyperedge Replacement

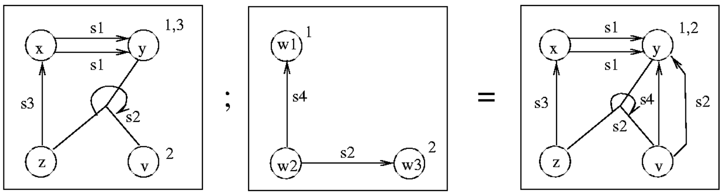


Figure 3. Sequential composition of open graphs.

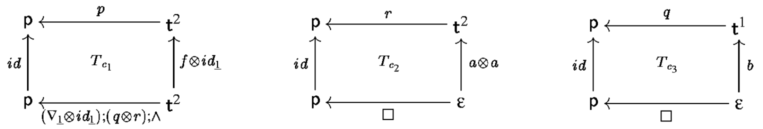
$$(dupl)\ \frac{a : \underline{n} \rightarrow \underline{m} \in \mathbf{M}(\Sigma_v)}{\nabla_a = \nabla_{\underline{n}} \xrightarrow[a\otimes a]{a} \nabla_{\underline{m}} \in \mathbf{R}}$$

Logic Programming, I

$c_1 \equiv$ `p(f(X1),X2) :- q(X1), r(X1,X2).`

$c_2 \equiv$ `r(a,a).`

$c_3 \equiv$ `q(b).`



Recent Bibliography on Tile Logic & PA

[1] Bruni, R., de Frutos-Escrig, D., Marti-Oliet, N. and Montanari, U., Bisimilarity Congruences for Open Terms and Term Graphs via Tile Logic, in: Catuscia Palamidessi, Ed., CONCUR 2000, Springer LNCS 1877, pp. 259-274.
[2] Bruni, R., Gadducci, F. and Montanari, U., Normal Forms for Algebras of Connections, TCS, vol 286 (2002) pp 247-292.
[3] Bruni, R. and Montanari, U., Dynamic Connectors for Concurrency, TCS, 281(1-2):131–176, 2002.
[4] Ferrari, G. and Montanari, U., Tile Formats for Located and Mobile Systems, Information and Computation, Vol. 156, No. 1/2, pp. 173-235, January 2000.
[5] Gadducci, F. and Montanari, U., Comparing Logics for Rewriting: Rewriting Logic, Action Calculi and Tile Logic, TCS, Vol. 285, Issue 2, 28 August 2002, Pages 319-358.

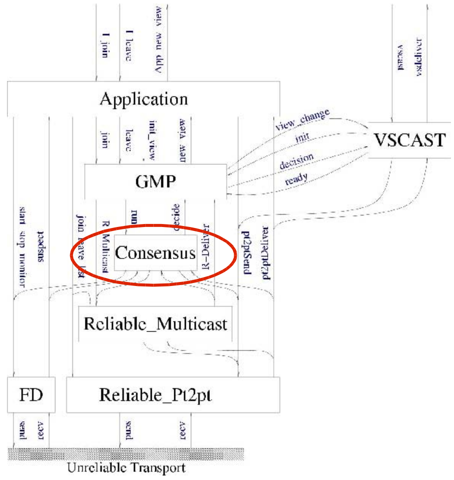
Modeling of and Reasoning about Fault-Tolerant Distributed Algorithms with Process Calculi / Operational Semantics

Uwe Nestmann
Rachele Fuzzati
Massimo Merro*

Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer & Communication Sciences (I&C)

**University of Verona*

Background: Group Communication



A Quote ...

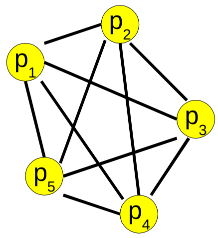
"**Most papers in Computer Science**

describe how their author learned

what someone else already knew."

(Peter Landin, around 1967?)

Computation Model



- ***fixed number of finite-state automata***
- ***asynchronous message-passing***
  (unordered message buffer / ether)
- ***crashes, but no recovery***
  (known problem: slow or crashed?)

P={p_1,...,p_n} **:** set of process ids
T : set/sequence of discrete time values
F : T → 2^P failure patterns in time

Runs (T,F,I,S):
- starting in some initial state I
- schedule S : sequence of steps of the form "*receive ; change-state ; emit*" generated by the individual automata in atomic fashion

Consensus (I)

Goal is the consensus among all processes about one of a number of proposed values.

t →

1	prop(v1)	dec(v'1)
2	prop(v2)	X
...	...	...
n	prop(vn)	dec(v'n)

In any run, the following properties must hold:

Validity
The decided value must be one of the proposed values.
Agreement ...
Termination
Every correct process eventually decides.

Consensus (II)

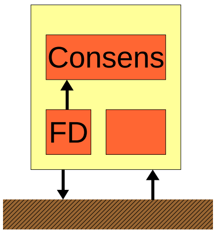
[Fischer, Lynch, Patterson 83/85]

Consensus is **not solvable** in **asynchronous** systems, if even only a **single** process **may crash**.

BUT:
With "enough" synchrony Consensus becomes solvable !

Failure Detectors (FD):
"Modules" that pronounce suspicions about which processes they currently believe to have crashed.

Failure Detectors [Chandra,Toueg]



FDs provide timely failure information about the other processes.

FDs are extremely unreliable !
- they may be wrong
- they may change their mind
- they may disagree

Search for minimal requirements on the reliablity of FDs !

in every possible run !

Completeness:
guaranteed suspicion of crashed processes.

Accuracy:
guaranteed non-suspicion of correct processes.

Runs with Failure Detectors

P={p_1,...,p_n} **:** process ids
T : set/sequence of time values
F : T $\rightarrow 2^P$ failure patterns in time

H : **P** $\times$ T $\rightarrow 2^P$: ***failure detector histories***

e.g.: H(p_2,26)={p_1,p_3} means that
"*The FD of process* p_2 *at time* 26
suspects processes p_1 *and* p_3 *to have crashed*."

A FD-run is a 5-tuple (T,F,H,I,S),
subject to a "compatibility condition" between F/H and S.

Failure Detector ◆S

[the weakest FD solving Consensus]

(strong) completeness:
eventually ***every crashed*** process
is permanently suspected
by ***every correct*** process (i.e., its FD).

$\exists\, t \in T: \forall\, p \in crash(F): \forall\, q \in correct(F): \forall\, t' \geq t: p \in H(q,t')$

(eventual weak) accuracy:
there is a time after which ***some correct*** process
is never again suspected
by ***any correct*** process (i.e., its FD).

$\exists\, t \in T: \exists\, p \in correct(F): \forall\, q \in correct(F): \forall\, t' \geq t: p \notin H(q,t')$

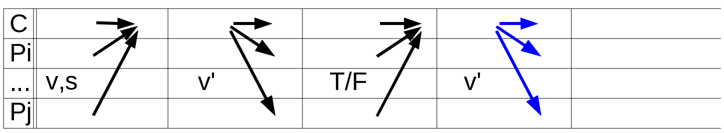
The Algorithm (I)

"***rotating coordinator paradigm***"
- in each **round** there is precisely one **coordinator** (C)
- every other process (P: "**participant**") knows it
- after every round, the ***next one*** becomes coordinator
- in every round,
  - the **participants** report their current beliefs on the decision value to the coordinator of this round
  - based on this information, the **coordinator** then proposes a new belief that it then tries to establish with the participants.

Note that rounds are completely asynchronous!

The Algorithm (II)

every round proceeds in 4 phases:
- P's send their current beliefs (v,s) to C
- C chooses one of the most recent (w.r.t. s), ... as soon as it has received "sufficiently" many.
- P's wait for the new coordinator proposal v' ... or suspect C (if possible! FD !!)
- C waits for "enough" acknowledgements (pos/neg) ... and decides, is a majority is in favor.

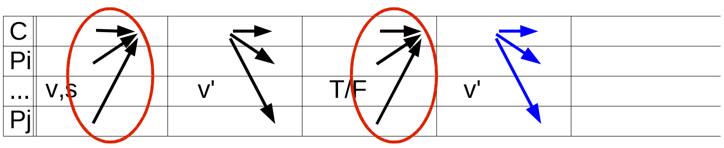


"Reliable Broadcast"

The Algorithm (III)

What happens, if participants crash?

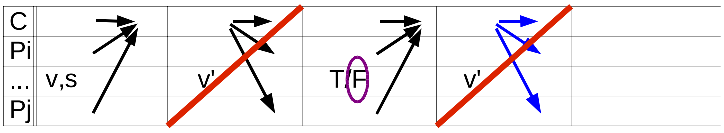
No problem, if a **majority** ⌈(n-1/2)⌉ survives !
(to prevent coordinators from blocking)



The Algorithm (IV)

What happens, if a coordinator crashes ?

No problem, if the participants
will eventually be allowed to suspect it !
(to prevent their own blocking)
... **completeness !** ...

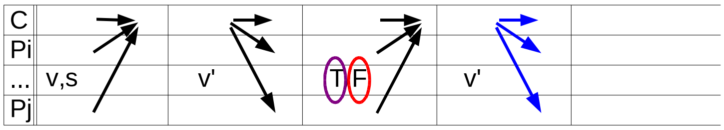


The Algorithm (V)

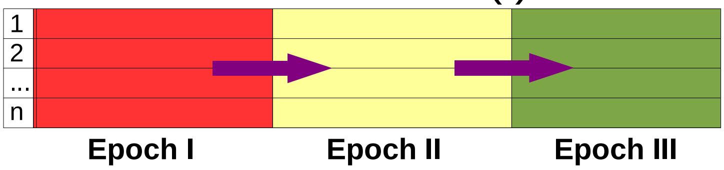
What happens, if a coordinator
is mistakenly suspected ?

(Restart with next round ...)

No problem, if only a single coordinator
will eventually no longer be mistakenly suspected!
... **accuracy !** ...



Correctness (I)



I **anything goes**, *every value has the same chances*

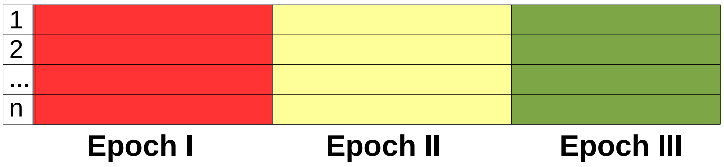
as soon as <u>in some round</u>
enough participants send positive acknowledgments

II one ***value becomes locked***, *but ...*
... no decisions have yet been taken

as soon as <u>in some round</u>
the coordinator of this round decides

III ***at least one correct coordinator decides***,
*all **correct** others **follow** (Reliable Broadcast ...)*

Correctness (II)



<u>*In every run, we require:*</u>

Validity
(quite trivial)

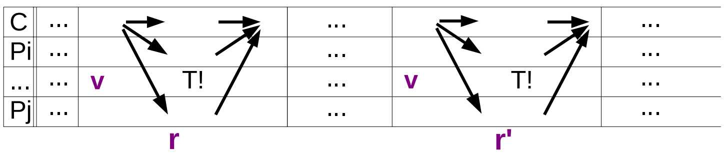
Agreement
(holds due to the locking of a value in Epoch II)

Termination
(holds due to completeness and accuracy)
(only then we are guaranteed to reach II and III !)

Proof: Agreement

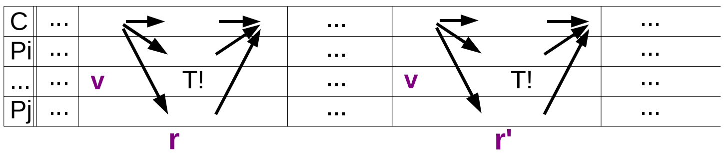
Theorem: If, in a run, two (correct) processes decide,
then their decision values must be the same.

- if decision, then ***enough positive acknowledgments*** must have been ***sent***, in some round.
- take ***"the first such round"*** **r**
- remember the value (**v**) that got locked in this round
- ***<u>Induction over all later rounds</u>*** **r'**, in which again enough positive acknowledgments were sent:
- In each such the new proposal of the coordinator is **v**



Induction Over Rounds (I)

The critical information is the ***access per round*** to all those ***messages*** that were sent in the ***past*** before reaching the current state.



... roughly corresponds to a ***matrix*** (process $\times$ round)

Induction Over Rounds (II)

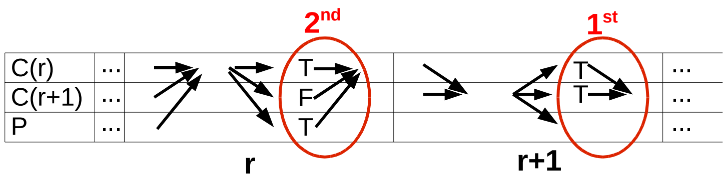
Rounds are asynchronous !!

The transition from round to round
only **indirectly** relates to *runs of algorithmus* ...

There are reachable states in which
- all processes are in completely different rounds
- all processes are coordinator "at the same time", althouhg in different rounds ...

Induction Over Rounds (III)

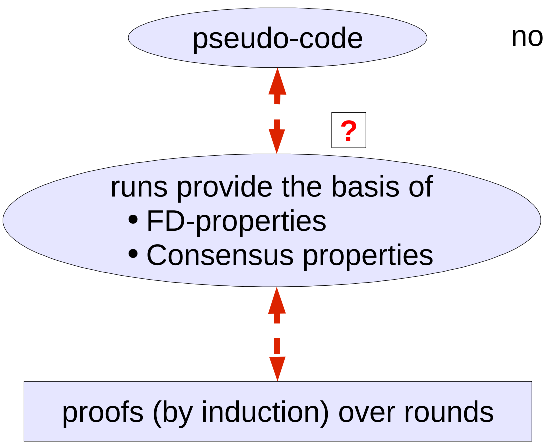
The induction may "back-fire" ...



The timeline of runs and the appearance of rounds that satisfy properties are not necessarily compatible.

What then have we proved on runs of algorithms when we perform induction over rounds?

Chandra & Toueg

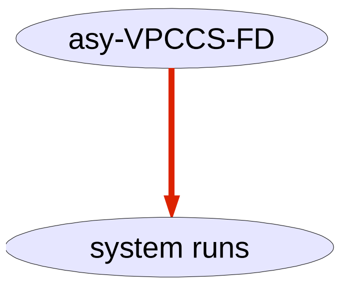


no semantics

informal

justified?

Process Calculus (I)



Consensus as
a term in the calculus
(explicitly distributed processes)

Consensus runs as
generated by an O.S.
(mathematical structure)

No-Name Process Calculus

N ::= i[P] | N | N | M
sites

M ::= a!v
messages

v ::= x | t | f | i | f(v) | ...
variables, booleans, ids, complex values, ...

P ::= M | G | Y⟨v⟩ | if P then P else P
(local) processes, with recursion & conditionals

G ::= α.P | G+G | **0**
guarded processes

α ::= a?x | $susp_j$ | a! | a?
input, suspicion, synchronous (local) signals

no name-passing ...
no restriction ...

Some Semantics Rules ...

Sites distribute over parallel (for syntactic convenience):

$i[\, P_1 \mid P_2 \,] \equiv i[\, P_1 \,] \mid i[P_2]$

Communication between sites takes two steps:

$i[\, a!v \,] \xrightarrow{\tau@i} a!v$

$a!v \mid i[\, a?x.P + G \,] \xrightarrow{\tau@i} i[\, P\{v/x\} \,]$

Local synchronization takes only one:

$i[\, a!.P_1 + G_1 \mid a?.P_2 + G_2 \,] \xrightarrow{\tau@i} i[\, P_1 \mid P_2 \,]$

Suspicions generate the only non-tau label:

$i[\, susp_j.P + G \,] \xrightarrow{susp_j@i} i[\, P \,]$

$$\mathit{Consensus}_{(v_1 \ldots, v_n)} \stackrel{\text{def}}{=} \prod_{i=1}^{n} i\left[\, \mathsf{P1}_i^{1,v_i,0,\emptyset} \mid \mathsf{D}_i \,\right]$$

$$\mathsf{P1}_i^{r,v,s,L} \stackrel{\text{def}}{=} \overline{c1_{\mathrm{crd}(r)}}\langle i, r, v, s\rangle \mid \text{if } i = \mathrm{crd}(r) \text{ then } \mathsf{P2}_i^{r,v,s,L} \text{ else } \mathsf{P3}_i^{r,v,s,L}$$

$$\begin{aligned}\mathsf{P2}_i^{r,v,s,L} \stackrel{\text{def}}{=} & \text{ if } |L_1^r| < \lceil \tfrac{n+1}{2} \rceil \\ & \text{ then } c1_i\,(\tilde{x})\,.\,\mathsf{P2}_i^{r,v,s,(1,\tilde{x})::L} \\ & \text{ else } \tau.\Big(\prod_{i \neq k=1}^{n} \overline{c2_k}\langle k, r, \mathrm{best}(L_1^r)\rangle \mid \mathsf{P4}_i^{r,\mathrm{best}(L_1^r),r,L} \Big)\end{aligned}$$

$$\begin{aligned}\mathsf{P3}_i^{r,v,s,L} \stackrel{\text{def}}{=} & \text{ if } L_2^r - \emptyset \\ & \text{ then } \Big(c2_i\,(\tilde{x})\,.\,\mathsf{P3}_i^{r,v,s,(2,\tilde{x})::L} + \mathrm{susp}_{\mathrm{crd}(r)}\,.\,\big(\overline{c3_{\mathrm{crd}(r)}}\langle i, r, \mathsf{f}\rangle \mid \mathsf{R}_i^{r,v,s,L} \big)\Big) \\ & \text{ else } \tau.\big(\overline{c3_{\mathrm{crd}(r)}}\langle i, r, \mathsf{t}\rangle \mid \mathsf{R}_i^{r,\mathrm{val}(L_2^r),r,L} \big)\end{aligned}$$

$$\begin{aligned}\mathsf{P4}_i^{r,v,s,L} \stackrel{\text{def}}{=} & \text{ if } |L_3^r| < \lceil \tfrac{n+1}{2} \rceil - 1 \\ & \text{ then } c3_i\,(\tilde{x})\,.\,\mathsf{P4}_i^{r,v,s,(3,\tilde{x})::L} \\ & \text{ else if } \bigwedge_{l \in L_3^r} \mathrm{bool}(l) \text{ then } \tau.\Big(\prod_{k=1}^{n} \overline{b_k}\langle i, k, 1, r, v\rangle \mid \mathsf{Z}_i^{r,v,s,L} \Big) \text{ else } \mathsf{R}_i^{r,v,r,L}\end{aligned}$$

$$\mathsf{Z}_i^{r,v,s,L} \stackrel{\text{def}}{=} \mathbf{0}$$

$$\mathsf{R}_i^{r,v,s,L} \stackrel{\text{def}}{=} \mathit{undecided}_i\,.\,\mathsf{P1}_i^{r+1,v,s,L}$$

$$\mathsf{D}_i \stackrel{\text{def}}{=} \overline{\mathit{undecided}_i}\,.\,\mathsf{D}_i + b_i\,(j, \cdot, m, r, v)\,.\Big(\overline{\mathit{decide}_i}\langle j, i, m, r, v\rangle \mid \prod_{k=1}^{n} \overline{b_k}\langle i, k, 2, r, v\rangle \Big)$$

FD-Runs in OS

System configurations $t \bullet N$, where $t \in T$.

Fix a pattern F and a history H (satisfying ◆S).

(ACT)
$$\frac{i \notin F(t) \qquad N \xrightarrow{\tau@i} N'}{t \bullet N \;\rightarrow\; t{+}1 \bullet N'}$$

(SUSP)
$$\frac{i \notin F(t) \qquad N \xrightarrow{susp@i} N' \qquad j \in H(i,t)}{t \bullet N \;\rightarrow\; t{+}1 \bullet N'}$$

FD-runs are
complete (in)finite sequences of system steps.

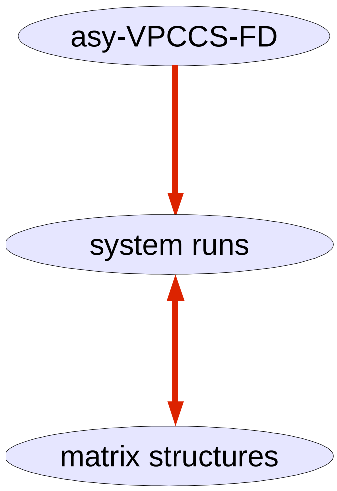
From Runs to Rounds: Matrices

- captures the ***complete message history*** of a state
- abstracts from when-in-time, emphasizes in-which-round
- close to the intuitive pictures used for "runs" ...

can be formalized in many ways (see the blackboard):

- as an ***independent matrix semantics***
  with separate rules mimicking the algorithm
  requires proofs of tight operational correspondence
  (as done in the context of our CONCUR-submission)
- generated ***on-the-fly*** by ***augmented OS-rules***
  (currently favored for the final full version ...)
- by ***reordering of runs*** using partial order information
  (would make it more P.A. like ... :-)

Process Calculus (II)



Consensus as
a term in the calculus
(explicitly distributed processes)

Consensus runs as
generated by the O.S.
(mathematical structure)

formal (!) proofs
(along the ideas of Chandra/Toueg)

Theorems

Agreement:
Let Consensus $\rightarrow^*$ N.
If both $N\downarrow_{i[vi]}$ and $N\downarrow_{j[vj]}$, then **vi=vj**.

by induction over rounds

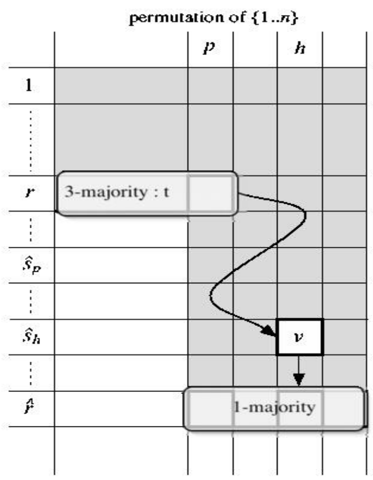
Pre-Termination:
All FD-runs are finite.

by contradiction

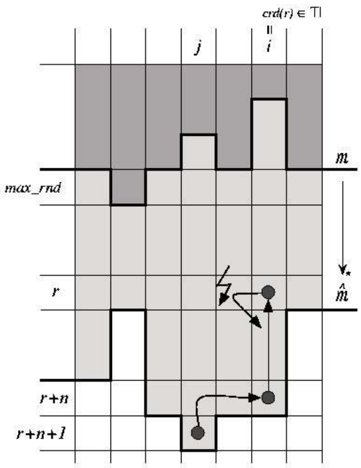
Termination:
Let $0 \bullet$ Consensus $\rightarrow^*$ $t \bullet N$ with $\neg\, t \bullet N \rightarrow$.
Then for all **i** : $N\downarrow_{i[vi]}$.

by contradiction

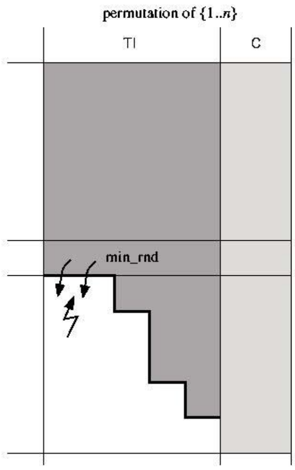
Proof: Agreement



Proof: Finiteness



Proof: Termination



Conclusions

Reusability: **FUTURE DIRECTIONS**
- many (most?) proofs in distributed algorithms are carried out using "global views"

Mechanization: **OPEN PROBLEM**
- "difficult": infinite/unbounded state spaces

Conjecture: **FUTURE DIRECTIONS**
- matrix semantics may be generated automatically
- base for simulation and animation of algotihms

Extension: **FUTURE DIRECTIONS**
- new modeling for all 4 (resp. 8) Chandra/Toueg-FDs
- probabilistic algorithms
- protocol composition & -refinement ...

"**Most papers in Computer Science**

describe how their author learned

what someone else already knew."

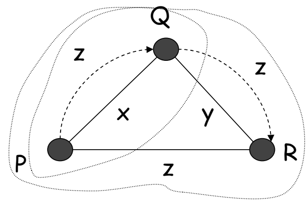
But they may nevertheless be useful ...

Probabilistic asynchronous π-calculus

Catuscia Palamidessi, INRIA Futurs, France
joint work with
Mihaela Herescu, IBM, Austin

Historic motivations

- Distributed implementation of the π calculus (with mixed choice)
  - Basic constructs to express parallelism, communication, mixed choice, generation of new names (which can be communicated and in turn used as channels), scope
  - Scope extrusion: a name can be communicated and its scope extended to include the recipient



π : the π calculus (w/ mixed choice)

Syntax

g ::= x(y) | x^y | τ prefixes (input, output, silent)

P ::=	$\Sigma_i\, g_i\, .\, P_i$	mixed guarded choice
|	P | P	parallel
|	(x) P	new name
|	rec_A P	recursion
|	A	procedure name

Operational semantics

- Transition system P -a‡ Q
- Rules

Choice $\Sigma_i\, g_i\, .\, P_i$ –g_i‡ P_i

Open
$$\frac{P \text{ -x\^{}y‡ } P'}{(y)\,P \text{ -x\^{}(y)‡ } P'}$$

Operational semantics

- Rules (continued)

Com
$$\frac{P \text{ -x(y)‡ } P' \qquad Q \text{ -x\^{}z‡ } Q'}{P \mid Q \text{ -τ‡ } P'[z/y] \mid Q'}$$

Close
$$\frac{P \text{ -x(y)‡ } P' \qquad Q \text{ -x\^{}(z)‡ } Q'}{P \mid Q \text{ -τ‡ } (z)\,(P'[z/y] \mid Q')}$$

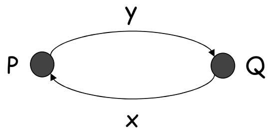
Par
$$\frac{P \text{ -g‡ } P'}{Q \mid P \text{ -g‡ } Q \mid P}$$
f(Q) and b(g) disjoint

Expressive Power of π

- link mobility
  - network reconfiguration
  - expresses HO (e.g. λ calculus) in a natural way
- mixed choice
  - solution to distributed problems involving distributed agreement

The expressive power of π

- Example of distributed agreement: the leader election problem



- A symmetric and fully distributed solution in π

x.P_{wins} + y^.P_{loses} | y.Q_{wins} + x^.Q_{loses} −τ‡ P_{loses} | Q_{wins}

−τ‡ P_{wins} | Q_{loses}

Implementation problem

- It is well known that formalisms able to express distributed agreement are difficult to implement in a distributed fashion

- For this reason, the field has evolved towards asynchronous variants of π or other asynchronous formalisms
  - for instance, the asynchronous π calculus [Honda-Tokoro'91, Boudol, '92]

π_a : the Asynchonous π

Version of [Amadio, Castellani, Sangiorgi '97]

Syntax

g ::= x(y) | τ prefixes

P ::=	$\Sigma_i\, g_i\, .\, P_i$	input guarded choice
|	x^y	output action
|	P | P	parallel
|	(x) P	new name
|	rec_A P	recursion
|	A	procedure name

Operational semantics of π_a

- Additional rule:

Out x^y −x^y‡ 0

- Asynchronous communication:
  - we can't write a continuation after an output, i.e. no x^y.P, but only x^y | P
  - so P will proceed without waiting for the actual delivery of the message

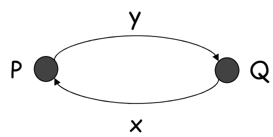
- Note: the original asynchronous π–calculus did not contain a choice construct. However the version presented here was shown equivalent to the original asynchronous π–calculus by [Nestmann and Pierce, '96]

π vs. π_a

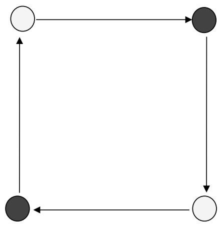
- π_a is suitable for distributed implementation, in contrast to π
- However, despite the difficulties regarding implementation, the π calculus is still very appealing, because of its superior expressive power

- Examples of problems that can be solved in π and not in π_a :
  - dining philosophers (following [Francez and Rodeh, '82])
  - the symmetric leader election problem , for any ring of processes
    - This problem cannot be solved in π_a , nor in CCS, nor in π_I [Palamidessi '97] and [Palamidessi '03]
    - The solution in π uses name mobility to fully connect the graph, and then mixed choice to break the symmetry [Palamidessi '03]

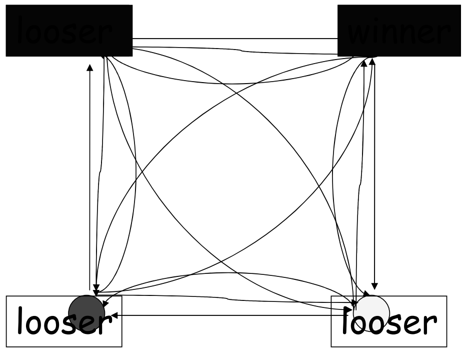
Example of a ring where the leader election problem cannot be solved in π_a



Example of a ring where the leader election problem cannot be solved in CCS (nor in π_I)

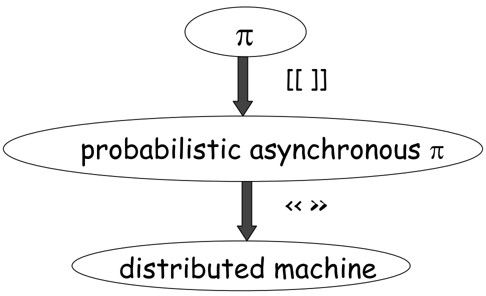


A solution to the leader election problem in π



Towards a fully distributed implementation of π

- The results of previous pages show that a fully distributed implementation of π must necessarily be randomized
- A two-steps approach:



Advantages: the correctness proof is easier since [[ ]] (which is the difficult part of the implementation) is between two similar languages

π_{pa}: the Probabilistic Asynchonous π

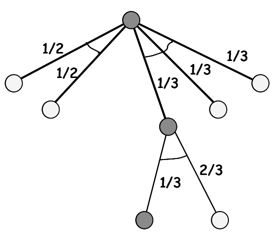
Syntax

g ::= $x(y) \mid \tau$ prefixes

P	::=	$\Sigma_i\, p_i\, g_i \,.\, P_i$	pr. inp. guard. choice $\Sigma_i\, p_i = 1$
	|	x^y	output action
	|	P | P	parallel
	|	(x) P	new name
	|	rec_A P	recursion
	|	A	procedure name

The operational semantics of π_{pa}

- Based on the **Probabilistic Automata** of Segala and Lynch
  - Distinction between
    - nondeterministic behavior (choice of the scheduler) and
    - probabilistic behavior (choice of the process)



Scheduling Policy:
The scheduler chooses the group of transitions

Execution:
The process chooses probabilistically the transition within the group

The operational semantics of π_{pa}

- Representation of a group of transition

$$P \{ \text{--}g_i\text{->}_{p_i} P_i \}_i$$

- Rules

Choice $\quad \Sigma_i\, p_i\, g_i \,.\, P_i \{\text{--}g_i\text{->}_{p_i} P_i\}_i$

Par
$$\frac{P\{\text{--}g_i\text{->}_{p_i} P_i\}_i}{Q \mid P\ \{\text{--}g_i\text{->}_{p_i} Q \mid P_i\}_i}$$

The operational semantics of π_{pa}

- Rules (continued)

Com
$$\frac{P\{\text{--}x_i(y_i)\text{->}_{p_i} P_i\}_i \qquad Q\{\text{--}x\hat{\ }z\text{->}_1 Q'\}_i}{P \mid Q\ \{\text{--}\tau\text{->}_{p_i} P_i[z/y_i] \mid Q'\}_{x_i=x} \cup \{\text{--}x_i(y_i)\text{->}_{p_i} P_i \mid Q\}_{x_i=/=x}}$$

Res
$$\frac{P\{\text{--}x_i(y_i)\text{->}_{p_i} P_i\}_i}{(x)\,P\{\text{--}x_i(y_i)\text{->}_{q_i} (x)\,P_i\}_{x_i=/=x}} \qquad q_i \text{ renormalized}$$

Implementation of π_{pa}

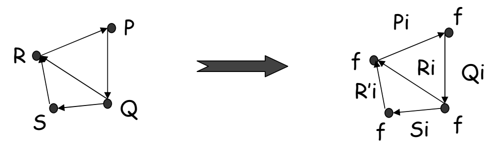
- **Compilation in Java** **<< >> : π_{pa} ‡ Java**
  - **Distributed**
    << P | Q >> = << P >>`.start();` << Q >>`.start();`
  - **Compositional**
    << P op Q >> = << P >> jop << Q >> for all op
  - Channels are one-position buffers with test-and-set (synchronized) methods for input and output

Encoding π into π_{pa}

- **[[ ]] : π ‡ π_{pa}**
- **Fully distributed**
  [[P | Q]] = [[P]] | [[Q]]
- **Preserves the communication structure**
  [[P σ]] = [[P]] σ
- **Correct wrt a notion of probabilistic testing semantics**
  P must O iff [[P]] must [[O]] with prob 1

Encoding π into π_{pa}

- Idea (from [Nestmann'97]):
  - Every mixed choice is translated into a parallel comp. of processes corresponding to the branches, plus a lock f
  - The input processes compete for acquiring both its own lock and the lock of the partner
  - The input process which succeeds first, establishes the communication. The other alternatives are discarded

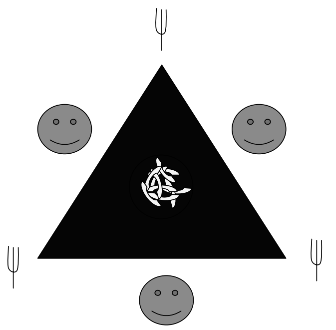


The problem is reduced to a generalized dining philosophers problem where each fork (lock) can be adjacent to more than two philosophers

Further, we can reduce the generalized DP to the classic case, and then apply the algorithm of Lehmann and Rabin

Dining Philosophers: classic case

Each fork is shared by exactly two philosophers



Dining Philosophers, classic case

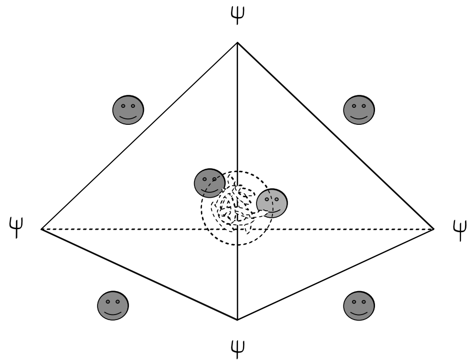
- The requirements on the encoding π ‡ π_{pa} imply symmetry and full distribution
- There are many solution to the DP problem, but in order to be symmetric and fully distributed a solution has necessarily to be randomized.
  Proved by [Lehmann and Rabin 81] - They also provided a randomized algorithm (for the classic case)
- Note that the DP problem can be solved in π in a fully distributed, symmetric way. Hence the need for randomization is not a characteristic of our approach: it would arise in any encoding of π into an asynchronous language.

The algorithm of Lehmann and Rabin

1. Think
2. choose first_fork in {left,right}   %commit
3. if taken(first_fork) then goto 3
4. take(first_fork)
5. if taken(first_fork) then goto 2
6. take(second_fork)
7. eat
8. release(second_fork)
9. release(first_fork)
10. goto 1

Dining Phils: generalized case

Each fork can be shared by more than two philosophers



Reduction to the classic case: each fork is initially associated with a token. Each phil needs to acquire a token in order to participate to the competition. The competing phils determine a set of subgraphs in which each subgraph contains at most one cycle

Generalized philosophers

- Another problem we had to face: the solution of Lehmann and Rabin works only for fair schedulers, while π_{pa} does not provide any guarantee of fairness

- Fortunately, it turns out that the fairness is required only in order to avoid a busy-waiting livelock at instruction 3. If we replace busy-waiting with suspension, then the algorithm works **for any scheduler**

  This result was achieved independently also by Fribourg et al, TCS 2002

The algorithm of Lehmann and Rabin Modified so to avoid the need for fairness

1. Think
2. choose first_fork in {left,right}   %commit
3. if taken(first_fork) then wait
4. take(first_fork)
5. if taken(first_fork) then goto 2
6. take(second_fork)
7. eat
8. release(second_fork)
9. release(first_fork)
10. goto 1

Conclusion

- We have provided an encoding of the π calculus into a probabilistic version of its asynchronous fragment
  - fully distributed
  - compositional
  - correct wrt a notion of testing semantics

- Advantages:
  - high-level solutions to distributed algorithms
  - Easier to prove correct (no reasoning about randomization required)

Orthogonality and Logic
(of course in Process Algebra)

July 23, 2003

Alban Ponse

Programming Research Group
University of Amsterdam
www.science.uva.nl/~alban/

Based on joint work with
Jan Bergstra (UvA, UU) and
Mark van der Zwaag (KUN)

Orthogonal bisimulation equivalence: what is it, why does it exist?

Outline, 22 slides:

First Idea: $a(\tau + \tau\tau)$ is orthogonally bisimilar to its compressed form $a\tau$.

Both represent action a followed by some internal activity, and neither is orthogonally bisimilar to a. So, the axiom $x = x\tau$ is not sound (its weakened version $x\tau\tau = x\tau$ is).

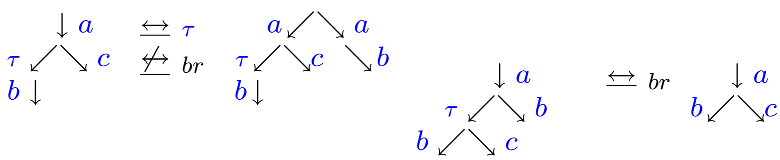
Typically, in orthogonal bisimilarity one may abstract from the *structure* of finitary internal activity, but not from its *presence*: one may call this compression.

Orthogonal bisimulation equivalence is a congruence for ACP with abstraction and priority operators (hence: orth.bis.equivalence $\subsetneq$ branching bis.equivalence).

Branching bisimulation equivalence

[van Glabbeek, 1989] is in many respects superior to other weak equivalences, e.g., it is a "fixed point" in the setting of back and forth bisimulations [De Nicola, Montanari, Vaandrager, 1990],

bis.eq.	notation	back and forth generalization
strong	($\underline{\leftrightarrow}$)	$\underline{\leftrightarrow}$
weak	($\underline{\leftrightarrow}_\tau$)	$\underline{\leftrightarrow}_{br}$
branching	($\underline{\leftrightarrow}_{br}$)	$\underline{\leftrightarrow}_{br}$



Let $O(e)$ be the observable content of a trace e (of some process), and $\leq$ a prefix ordering on traces.

Proposition [van Glabbeek, 1994].
Processes p, q have the *same branching structure* iff $\exists R \subseteq traces(p) \times traces(q)$ with $domain(R) = traces(p)$ and $range(R) = traces(q)$ s.t.

i) $eRf \;\Rightarrow\; O(e) = O(f)$,

ii) $\exists e'(e \leq e' R f') \;\Leftrightarrow\; \exists f(eRf \leq f')$,

iii) $\exists f'(f \leq f' R^{-1} e') \;\Leftrightarrow\; \exists e(f R^{-1} e \leq e')$.

Famous $O(e)$: as e but with all τ's removed, then:

branching bisimilarity $\Leftrightarrow$ same branching structure.

5

The equation $x(\tau(y+z)+z) = x(y+z)$ holds in branch.bis.eq, e.g.,

$$a(b+c) \xrightarrow{a} b+c \xrightarrow{b,c} \surd$$

$$\underline{\leftrightarrow}_{br} \quad a(\tau(b+c)+c) \xrightarrow{a} \tau(b+c)+c \xrightarrow{\tau} b+c \xrightarrow{b} \surd$$

$$\tau(b+c)+c \xrightarrow{c} \surd \qquad b+c \xrightarrow{c} \surd$$

But, with priority operator θ and partial priority ordering on actions, e.g., action `to` (`time-out`) has lower priority than action b, $\underline{\leftrightarrow}_{br}$ is no longer a congruence:

$$\theta(a(b+\texttt{to})) \xrightarrow{a} \theta(b+\texttt{to}) \xrightarrow{b} \surd$$

$$\not\underline{\leftrightarrow}_{br} \quad \theta(a(\tau(b+\texttt{to})+\texttt{to})) \xrightarrow{a} \theta(\tau(b+\texttt{to})+\texttt{to}) \xrightarrow{\tau} \theta(b+\texttt{to}) \xrightarrow{b} \surd$$

$$\theta(\tau(b+\texttt{to})+\texttt{to}) \xrightarrow{\texttt{to}} \surd$$

6

[van Glabbeek, 1994]: Branching bisimulation equivalence is the coarsest equivalence that respects the branching structure of processes with silent steps.

[van Glabbeek & Weijland, 1996]: *"We know of no useful operator for which some abstract equivalence in the linear time-branching time spectrum is a congruence, but rooted branching bisimulation equivalence is not."*

Well, we think we found one that is (still) outside that (large?) spectrum: one that is a congruence for priority operators.

7

Consider a labelled transition system with a set S of states, a set L of labels, and with a transition relation over $S \times L \times S$ (notation $s \xrightarrow{a} s'$) and a termination predicate $\surd$ on S.

A binary, symmetric relation R on S is an orthogonal bisimulation if sRr implies

i) if $\surd s$, then $\surd r$;

ii) if $s \xrightarrow{a} s'$ for some s' and $a \neq \tau$, then $r \xrightarrow{a} r'$ for some r' with $s'Rr'$;

iii) $s \xrightarrow{\tau} s'$ for some s', then $r \xrightarrow{\tau}$ and there is a path $r_0 \ldots r_n \in \tau\text{-}paths(r)$ with $n \geq 0$ such that $s'Rr_n$ and sRr_i for all $i < n$.

8

States s and r are orthogonally bisimilar, notation $s \;\underline{\leftrightarrow}_{orth}\; r$, if they are related by some orthogonal bisimulation.

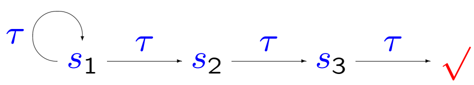
And, for congruence properties one needs a rooted version, as usual: $\tau \;\underline{\leftrightarrow}_{orth}\; \tau\tau$, while $a+\tau \;\not\underline{\leftrightarrow}_{orth}\; a+\tau\tau$.

Example: states s_1 and s_2 below are rooted orthogonally bisimilar to each other but not to s_3, while s_2 and s_3 are rooted branching bisimilar.

$$s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} s_3 \xrightarrow{\tau} s_4$$

9

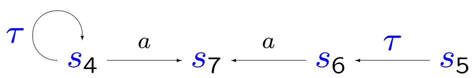
Divergence: sometimes discarded, sometimes not. Example: states s_1 and s_2 below are (rooted) orthogonally bisimilar



or,

$\tau^*(\tau\tau\tau) \;\underline{\leftrightarrow}_{orth}\; \tau\tau$

versus



where $s_4 \;\not\underline{\leftrightarrow}_{orth}\; s_5$ and $s_4 \;\not\underline{\leftrightarrow}_{orth}\; s_6$.
So τ-divergence is context-dependent and thus motivates a divergence sensitive variant: we defined it.

10

Modal characterization: transition labels act as existential modal operators; formulas are formed with negation, conjunction, an until operator U, a termination predicate $\surd$, and a τ-enabledness predicate:

$$\phi ::= \ \surd \mid \tau \mid a\phi \mid \neg\phi \mid \phi \wedge \phi \mid \phi\, U\, \phi,$$

with e.g. state $s \models \tau$ if $s \xrightarrow{\tau}$, etc.

States s and r are equivalent, notation $s \sim r$, if, for all formulas ϕ, $s \models \phi$ if and only if $r \models \phi$.

For any transition system over $L_\mathcal{T}$ with termination, $s \leftrightarrow_{orth} r$ implies $s \sim r$.

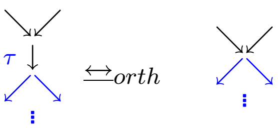
In the other direction, the characterization is less general: restricting to transition systems that are finitely branching (each state has finitely many successors) and τ-path-image-finite:

for all states s there are finitely many states s' with $s' \in \tau\text{-}paths(s)$.

For any such transition system, $s \sim r$ implies $s \leftrightarrow_{orth} r$.

A nice relation with strong bisimulation:

A τ-step is inert whenever



A state in a trans.system is compact if no inert τ-transitions can be reached.

For compact states $s,\ t$,

$$s \leftrightarrow_{orth} t \quad \Leftrightarrow \quad s \leftrightarrow t$$

Axioms for orthogonal bisimulation equivalence:

(O1) $\quad x\tau\tau = x\tau$

(O2) $\quad x\tau(y+z) = x(y+z) \quad$ if $\tau y = \tau\tau y,\ \tau z = \tau\tau z$

(O3) $\quad x(\tau(y+z)+z) = x(y+z) \quad$ if $\tau y = \tau\tau y$

A condition $\tau x = \tau\tau x$ is true iff x does not equal δ (deadlock) and all initial actions equal τ.

Compare with those for branching bisimulation:

(B1) $\quad x\tau = x$

(B2) $\quad x(\tau(y+z)+z) = x(y+z)$

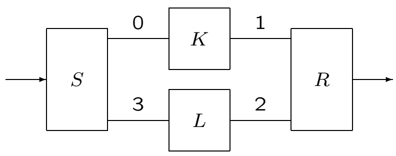
Completeness of the axiom system BPA with $(\delta,)\tau$ for $\leftrightarrow_{orth}$:

1. Any two rooted orthogonally bisimilar closed terms are derivably equal to terms with only compact successors;

2. For such terms, strong bisimilarity coincides with rooted orthogonal bisimilarity.

This yields completeness of ACP with τ, τ_I (renaming into τ) and priority operators, for $\leftrightarrow_{orth}$.

Example: a Par protocol (Positive Acknowledgment and Retransmission) P, where channels K and L are unreliable (may corrupt or lose data(-packages)):



$P = f(S \parallel K \parallel R \parallel L)$,

f *a composition of abstraction, encapsulation and priority*

E.g., $L = (r_2(i \cdot s_3 + i))^*\delta$, so an acknowledgement received via r_2 may be communicated to S via port 3, or can be lost. Upon inactivity, a time-out action can trigger resending of the last data-package.

Restriction: compression of silent loops requires at least one silent loop-exit (for sound fairness laws):

$x(\tau^*(y+\tau z)) = x(y+\tau z)$ while $a(\tau^* b) \not\leftrightarrow_{orth} a(\tau)b$

Often: communication protocol after abstraction equals a one-place (or n-place) buffer in $\leftrightarrow_{br}$.

Par protocol: a best case, i.e.,

$P \leftrightarrow_{orth} (\Sigma_{d\in D}\, r(d)\cdot\tau\cdot s(d)\cdot\tau)^*\delta$.

A worst case, e.g., CAPB (Concurrent Alternating Bit Protocol) [van der Zwaag, 1998]:

CABP $\leftrightarrow_{orth} (\tau^*\Sigma_{d\in D}\, r(d)\cdot\tau\cdot(\tau^* s(d))\cdot\tau)^*\delta$.

17

Let me recall: $O(e)$, the observable content of a trace e of some process, is as e without the τ-occurrences, and $\leq$ is the prefix ordering on traces.

Proposition [van Glabbeek, 1994].
Processes p,q have the *same branching structure* iff $\exists R \subseteq traces(p)\times traces(q)$ with $domain(R) = traces(p)$ and $range(R) = traces(q)$ s.t.

i)	eRf	$\Rightarrow$	$O(e) = O(f)$,
ii)	$\exists e'(e \leq e' R f')$	$\Leftrightarrow$	$\exists f(eRf \leq f')$,
iii)	$\exists f'(f \leq f' R^{-1} e')$	$\Leftrightarrow$	$\exists e(f R^{-1} e \leq e')$.

branching bisimilarity $\Leftrightarrow$ same branching structure.

18

Let $C(e)$ be the compression content of a trace e: as e but with all second and consecutive τ's left out.

Proposal.
Processes p,q have the *same compression structure* iff $\exists R \subseteq traces(p)\times traces(q)$ with $domain(R) = traces(p)$ and $range(R) = traces(q)$ s.t.

i)	eRf	$\Rightarrow$	$C(e) = C(f)$,
ii)	$\exists e'(e \leq e' R f')$	$\Leftrightarrow$	$\exists f(eRf \leq f')$,
iii)	$\exists f'(f \leq f' R^{-1} e')$	$\Leftrightarrow$	$\exists e(f R^{-1} e \leq e')$.

Conjecture:
orthogonal bisimilarity $\Leftrightarrow$ same compression structure.

19

Conclusions and questions:

1. Orthogonal bisimulation equivalence respects the branching structure of processes,

2. Orth. bis. in the setting of ACP with priority operators (partial priority ordering) is a congruence (incl. axiomatization);

even a correspondence result holds:

for I a set of internal actions, all of which have the same priority as τ. Then τ_I and θ commute modulo orthogonal bisimilarity: $\theta\circ\tau_I(x) = \tau_I\circ\theta(x)$, the strongest commutation result that can be expected.

20

3. Protocol specification and verification: nice results only if τ-loops have a τ-exit — so that the *presence* of internal activity is preserved.

? A reasonable price for freely using priority operators. Cf. channel modelling in a non-deterministic style, e.g. $L = (r_2(i\cdot s_3 + i))^*\delta$ or $L = (r_2(i\cdot s_3 + i\cdot\bot))^*\delta$.

?. This Conjecture:
orthogonal bisimilarity $\Leftrightarrow$ same compression structure.
Appears reasonable — still to prove...

21

References:

J.A. Bergstra, A. Ponse, and M.B. van der Zwaag, Branching time and orthogonal bisimulation equivalence, to appear in TCS — some time, or `www.science.uva.nl/~alban`

M.B. van der Zwaag, Some verifications in process algebra with iota, Report P9806, PRG, University of Amsterdam, 1998.

22

Logic of ACP: correspondence, and sequentiality vs. symmetry.

- Some background (Kleene's 3-valued logic, 1938)
- The lattice/different interpretations of this logic:
- The operations, sequential vs. symmetric
- Correspondence with BPA_δ

References:

M.B. van der Zwaag. Chapter 2 of Models and Logic for Process Algebra. The logic of ACP. Chapter 2 of Models and Logic for Process Algebra. PhD. Thesis, University of Amsterdam, 2002 (IPA 2002-11).

J.A. Bergstra and A. Ponse. Process Algebra and Conditional Composition. Information Processing Letters, 80(1):41-49, 2001.

23

J.A. Bergstra and A. Ponse. Bochvar-McCarthy logic and process algebra. Notre Dame Journal of Formal Logic, 39(4):464-484, 1998.

J.A. Bergstra and A. Ponse. Process algebra with four-valued logic. Journal of Applied Non-Classical Logics, 10(1):27-53, 2000.

J.A. Bergstra and A.Ponse. Process algebra with five-valued logic. In C.S. Calude and M.J. Dinneen, editors, Combinatorics, Computation, and Logic, Proceedings of DMTCS'99 and CATS'99, Auckland, vol. 21, nr. 3 of Australian Computer Science Communications, pages 128-143. Springer-Verlag, 1999.

J.A. Bergstra and A. Ponse. Kleene's three-valued logic and process algebra. Information Processing Letters 67(2):95-103, 1998.

24

Three-valued logic $\mathbb{K}_3$ with undefinedness value u:

$\neg$	
T	F
F	T
u	u

$\wedge$	T	F	u
T	T	F	u
F	F	F	F
u	u	F	u

Two interpretations for u: undefined (D) and overdefined (C): $\mathbb{K}_4$, that is,

$\neg$	
C	C
T	F
F	T
D	D

$\wedge$	C	T	F	D
C	C	C	F	F
T	C	T	F	D
F	F	F	F	F
D	F	D	F	D

25

A complete axiomatization of $\mathbb{K}_3$:

(N0)	$\neg(x \wedge y) = \neg x \vee \neg y$
(N1)	$\neg\neg x = x$
(N2)	$\neg \mathsf{T} = \mathsf{F}$
(N3)	$\neg u = u$
(K1)	$x \wedge y = y \wedge x$
(K2)	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
(K3)	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
(K4)	$x \vee (x \wedge y) = x$
(K5)	$\mathsf{T} \wedge x = x$

26

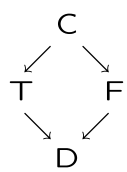
A complete axiomatization of $\mathbb{K}_4$:

(N0)	$\neg(x \wedge y) = \neg x \vee \neg y$
(N1)	$\neg\neg x = x$
(N2)	$\neg \mathsf{T} = \mathsf{F}$
(N3)	$\neg \mathsf{C} = \mathsf{C}$
(N4)	$\neg \mathsf{D} = \mathsf{D}$
(K1)	$x \wedge y = y \wedge x$
(K2)	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
(K3)	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
(K4)	$x \vee (x \wedge y) = x$
(K5)	$\mathsf{T} \wedge x = x$
(K6)	$\mathsf{C} \wedge \mathsf{D} = \mathsf{F}$

27

An equivalent logic $\mathbb{L}_4$ with one primitive $x \lhd y \rhd z$ (if y then x else z).

The information ordering lattice:



$x \lhd \mathsf{C} \rhd z = x \sqcup z$ (least upper bound)
$x \lhd \mathsf{T} \rhd z = x$
$x \lhd \mathsf{F} \rhd z = z$
$x \lhd \mathsf{D} \rhd z = \mathsf{D}$

Both logics $\mathbb{K}_4$ and $\mathbb{L}_4$ are expressively adequate: each monotone function can be expressed, and back and forth translations are defined.

28

A complete axiomatization of $\mathbb{L}_4$:

$$\begin{array}{lrcl}
\text{(L1)} & x \lhd (x' \lhd y \rhd z') \rhd z & = & \\
 & & & (x \lhd x' \rhd z) \lhd y \rhd (x \lhd z' \rhd z) \\
\text{(L2)} & (x \lhd y \rhd z) \lhd y' \rhd (x' \lhd y \rhd z') & = & \\
 & & & (x \lhd y' \rhd x') \lhd y \rhd (z \lhd y' \rhd z') \\
\text{(L3)} & (x \lhd y \rhd x') \lhd y \rhd z & = & x \lhd y \rhd (x' \lhd y \rhd z) \\
\text{(L4)} & \mathsf{T} \lhd x \rhd \mathsf{F} & = & x \\
\text{(LT)} & x \lhd \mathsf{T} \rhd y & = & x \\
\text{(LF)} & x \lhd \mathsf{F} \rhd y & = & y \\
\text{(LD)} & x \lhd \mathsf{D} \rhd y & = & \mathsf{D} \\
\text{(Lc1)} & x \lhd \mathsf{C} \rhd y & = & y \lhd \mathsf{C} \rhd x \\
\text{(Lc2)} & x \lhd \mathsf{C} \rhd \mathsf{D} & = & x \\
\text{(Lc3)} & \mathsf{C} \lhd \mathsf{C} \rhd x & = & \mathsf{C}
\end{array}$$

29

We lift $\mathbb{L}_4$ to process algebra: $x \lhd \phi \rhd y$, and mostly then use the notation

$$x +_\phi y$$

instead, and the shorthand $+$ for $+_\mathsf{C}$. A typical axiom:

$$(x +_\phi y)z = xz +_\phi yz$$

A typical derivation:

$$\begin{array}{rcll}
x + x & = & (x +_\mathsf{T} y) +_\mathsf{C} (x +_\mathsf{T} y) & \\
 & = & x +_{\mathsf{T} \lhd \mathsf{C} \rhd \mathsf{T}} y & \\
 & = & x & (\mathsf{T} \lhd \mathsf{C} \rhd \mathsf{T} = \mathsf{T} \sqcup \mathsf{T} = \mathsf{T})
\end{array}$$

30

A correspondence result:

Let $t_1(\vec{x}, \vec{v}) = t_2(\vec{x}, \vec{v})$ be a process identity with process variables $\vec{x}$ and condition variables $\vec{v}$ in which the only constants are in $\mathsf{T}, \mathsf{F}, \mathsf{C}, \mathsf{D}$ and the only operation is $+_\phi$, written as $\lhd \phi \rhd$. Then

$$\text{gBPA}_\delta / \underline{\leftrightarrow} \models t_1(\vec{x}, \vec{v}) = t_2(\vec{x}, \vec{v})$$

if and only if $\mathbb{L}_4 \models t_1(\vec{x}, \vec{v}) = t_2(\vec{x}, \vec{v})$, where in the latter statement, $\vec{x}$ also represents condition variables.

Consequence: $x + \delta = (x +_\mathsf{T} y) +_\mathsf{C} (x +_\mathsf{D} y) =$
$x +_{\mathsf{T} \lhd \mathsf{C} \rhd \mathsf{D}} y = x +_\mathsf{T} y = x$.

31

Actions in formats of SOS rules and Ordered SOS rules

Irek Ulidowski

Department of Computer Science,
University of Leicester,
UK

I.Ulidowski@mcs.le.ac.uk

Plan

1. Structural Operational Semantics.
2. Process Languages (or TSSs).
3. Formats of SOS rules.
4. Ordered SOS format.
5. Ordered SOS and Priority Rewriting.
6. Actions in SOS rules
   - Silent and visible actions,
   - Timed actions.
7. Summary of open problems and future research.

Structural Operational Semantics

Operational semantics defines behavioural and computational meaning of programs and systems.

Concurrent systems are modelled by processes, which are ground terms (over Σ) of a Process Language (PL) or TSS.

The behaviour of concurrent systems is represented by means of actions ($\in$ Act), and is defined by a labelled transition relation $\rightarrow$:

$t \stackrel{\alpha}{\rightarrow} s$ if t evolves to s by performing action α.

Within the SOS framework, given a PL with a signature Σ, transition relation $\rightarrow$ is defined by transition rules (SOS rules, or simply rules) for the operators in Σ. Rules are expressions of the form

$$\frac{H}{C}$$

where H is a set of literals called premises, (both positive and negative), and C is a positive literal called conclusion.

Format is a set of syntactic forms of SOS rules. It specifies precisely the form of literals in H and C.

Examples of SOS rules

Actions: α, β; visible actions a, b; silent action τ.

Operators: $X;Y$, $X|Y$, $a\&b(X)$.

Predicates: $X\surd$, $X\overline{\surd\!\!\surd}$, $X\surd\!\!\surd$(global successful termination).

$$\frac{X \stackrel{\alpha}{\rightarrow} X'}{X;Y \stackrel{\alpha}{\rightarrow} X';Y} \qquad \frac{\{X \stackrel{\beta}{\not\rightarrow}\}_{\forall\beta} \quad Y \stackrel{\alpha}{\rightarrow} Y'}{X;Y \stackrel{\alpha}{\rightarrow} Y'} \; or \; \frac{X\surd \quad Y \stackrel{\alpha}{\rightarrow} Y'}{X;Y \stackrel{\alpha}{\rightarrow} Y'}$$

$$\frac{X \stackrel{\sigma}{\rightarrow} X' \quad Y \stackrel{\sigma}{\rightarrow} Y' \quad X|Y \stackrel{\tau}{\not\rightarrow}}{X|Y \stackrel{\sigma}{\rightarrow} X'|Y'}$$

$$\frac{X \stackrel{a}{\rightarrow} X' \quad X \stackrel{b}{\rightarrow} X''}{a\&b(X) \stackrel{ok}{\rightarrow} \mathbf{0}} \qquad \frac{X \stackrel{\tau}{\rightarrow} X' \quad X' \stackrel{a}{\rightarrow} X''}{X \stackrel{a}{\rightarrow} X''}$$

$$\frac{X \stackrel{\varepsilon}{\rightarrow} X' \quad X' \stackrel{\tau}{\not\rightarrow} \quad \neg(X'\surd)}{X\overline{\surd\!\!\surd}} \qquad \frac{\neg(X\overline{\surd\!\!\surd})}{X\surd\!\!\surd}$$

Process Languages, TSSs

A PL or TSS is $(\Sigma, \mathsf{Act}, R)$, where R is a "method for defining the operational meaning of Σ". Typically, R is just a set of SOS rules. In our work R is often a set of rules equipped with orderings.

The behaviour of systems is modelled by the labelled transition system (LTS) $(\mathsf{T}(\Sigma), \mathsf{Act}, \rightarrow)$, where $\mathsf{T}(\Sigma)$ is the set of ground terms over Σ.

Process relations, such as bisimulation, can now be defined over the LTS.

A class of PLs is the set of all PLs whose "method" R is of a certain format. Hence, a format defines a class of PLs.

What results do we prove about formats?

1. How to associate $\rightarrow$ with a PL in a given format.

2. How expressive is a format?

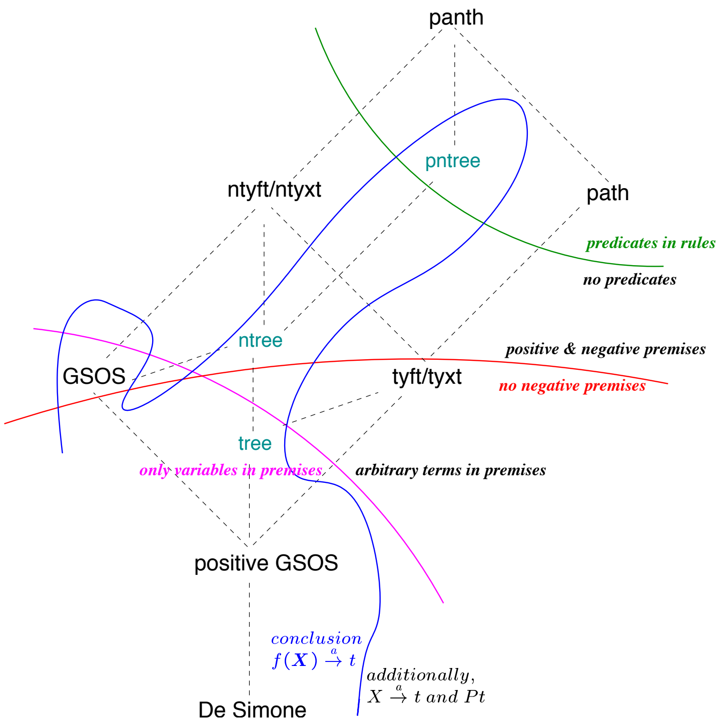
3. Is a process relation $\precsim$ a congruence?

$$\forall f \in \Sigma, \forall \boldsymbol{t}, \boldsymbol{s}.\ \boldsymbol{t} \precsim \boldsymbol{s} \quad \text{implies} \quad f(\boldsymbol{t}) \precsim f(\boldsymbol{s})$$

4. Does a PL in a format enjoy certain properties?

- time determinacy, maximal progress, ... ,
- information flow: non interference.

SOS rule: $\frac{H}{C}$; H is a set of expressions of the form $t \xrightarrow{a} t'$, Pt (positive), $t \stackrel{a}{\nrightarrow}$, $\neg Pt$ (negative); C has one of the forms $f(\boldsymbol{X}) \xrightarrow{a} t'$, $X \xrightarrow{a} t'$ or $Pf(\boldsymbol{X})$, PX.



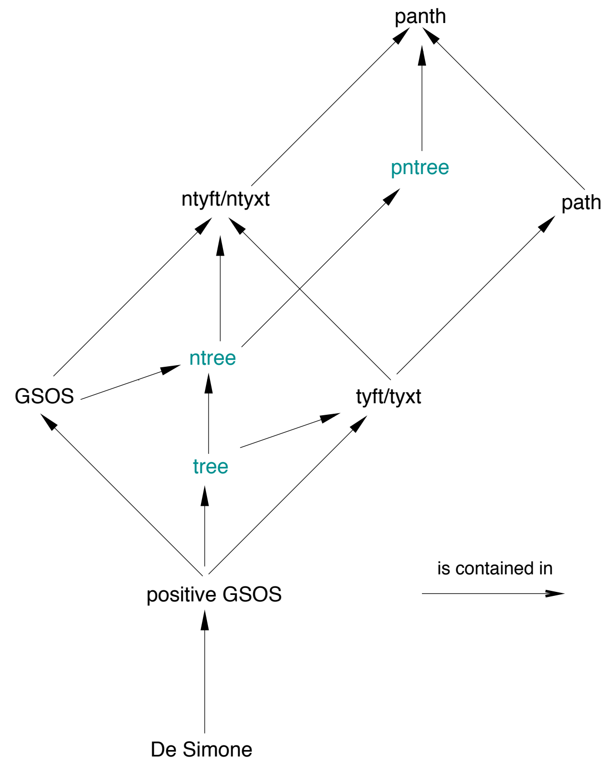
Meaning of TSSs

A TSS is meaningful iff it has a unique stable transition relation: Groote, Bol, van Glabbeek.

A TSS is meaningful iff it is (ws-) complete: Groote, Bol, van Glabbeek.

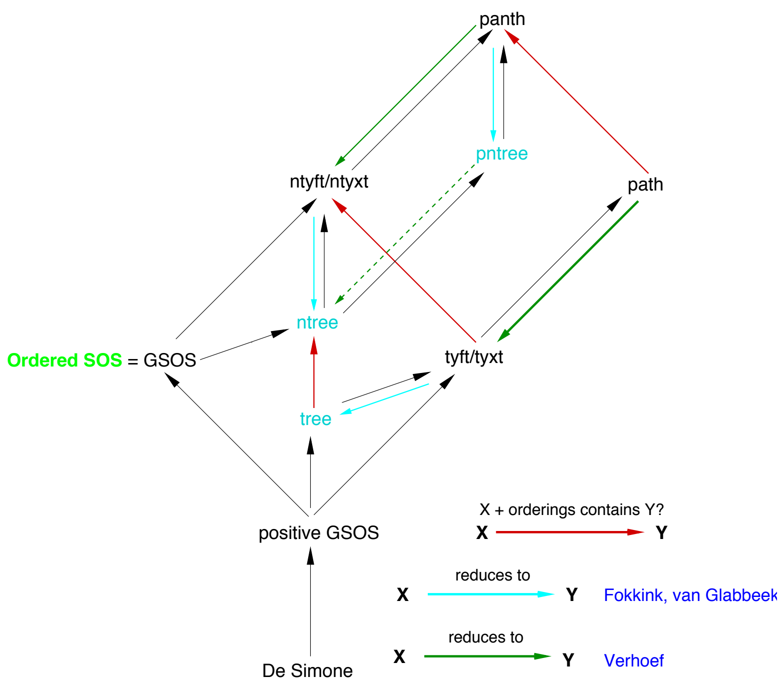
Formats of SOS rules

Format: a set of syntactic forms of SOS rules.



Plotkin, Milner, De Simone, Bloom et al., Vaandrager, Groote, Bol, van Glabbeek, Fokkink, Baeten, Verhoef

Expressiveness of formats



Formats with no negative premises reduce to, or are contained in, the tree format. All formats reduce to, or are contained in, the ntree format.

Ordered SOS (OSOS) format (Ulidowski, Phillips) = positive GSOS + orderings on rules.

What properties do tree, tyft/tyxt and path rules with orderings enjoy? Do the red arrows hold?

Ordered SOS format

We employ positive GSOS rules

$$\frac{\{\ X_i \overset{\alpha_{ij}}{\to} Y_{ij}\ \}_{i\in I, j\in J_i}}{f(X_1,\ldots,X_n) \overset{\gamma}{\to} t[\boldsymbol{X},\boldsymbol{Y}]}$$

and orderings on rules: $>_f\ \subseteq\ rules(f) \times rules(f)$.

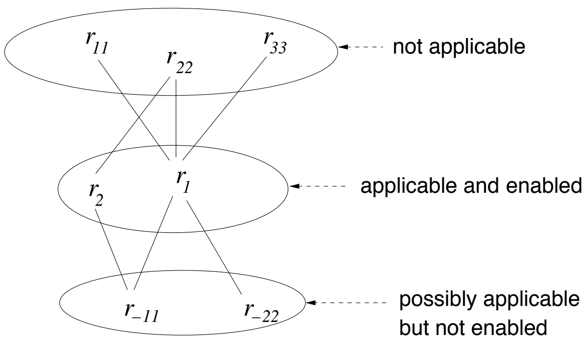
$>_f$ controls the order of application of rules when deriving transitions of processes with the outermost f.

How is $\to$ defined?

A rule r applies to $f(\boldsymbol{t})$ if its premises are valid for $\boldsymbol{t}$.

A rule r is enabled at $f(\boldsymbol{t})$ if r applies to $f(\boldsymbol{t})$ and no rule higher than r applies to $f(\boldsymbol{t})$.

Consider $f(\boldsymbol{t})$ and the following OSOS rules for f. Suppose that r_1 and r_2 are enabled at $f(\boldsymbol{t})$. Then



Order on rules has the same effect as negative premises:

$$\frac{X \overset{\alpha}{\to} X'}{X;Y \overset{\alpha}{\to} X';Y}\, r_{\alpha *} \qquad \frac{Y \overset{\beta}{\to} Y'}{X;Y \overset{\beta}{\to} Y'}\, r_{*\beta}$$

The ordering is $r_{\alpha *} > r_{*\beta}$ for all α and β. Another example (page 4):

communication rule, 2 τ-rules $>$ $$\frac{X \overset{\sigma}{\to} X' \quad Y \overset{\sigma}{\to} Y'}{X|Y \overset{\sigma}{\to} X'|Y'}$$

If orderings are not limited to $rules(f) \times rules(f)$ but can be subsets of $rules(\Sigma) \times rules(\Sigma)$, then

$$\frac{X \overset{a}{\to} X' \quad X' \overset{a}{\to} Y}{g(X) \overset{a}{\to} \mathbf{0}} \qquad \frac{g(X) \overset{a}{\not\to}}{f(X) \overset{a}{\to} \mathbf{0}}$$

can be expressed as

$$\frac{X \overset{a}{\to} X' \quad X' \overset{a}{\to} Y}{g(X) \overset{a}{\to} \mathbf{0}} \quad > \quad \frac{}{f(X) \overset{a}{\to} \mathbf{0}}$$

Could ntree be reformulated as tree with orderings?

The corresponding question can be asked about

- ntyft/ntyxt and tyft/tyxt with orderings,
- panth and path with orderings.

Ordered SOS and Priority Rewriting

Priority Rewrite Systems (PRSs) are TRSs with priority order on rewrite rules (Baeten et al., Toyama, Sakai).

A rewrite rule r_2 with lower priority than a rewrite rule r_1 can be applied to reduce term t (in favour of r_1) if no internal reduction of t can produce an r_1-redex.

We can produce PRSs for OSOS PLs (Ulidowski).

van de Pol gives PRSs operational semantics by translating them into TSSs with rules with universal quantification, lookahead and negative premises:

the sound and complete rewrite set for PRS $\mathcal{P}$
coincides with
the unique stable model for TSS($\mathcal{P}$).

Can PRSs be given semantics using simpler rules but with orderings?

Operational semantics of popular PLs have been given in f rewriting logic (Maude) and Conditional TRSs.

What forms of rules (with orderings) can be represented neatly in terms of TRSs (PRSs)?

Actions in SOS rules

We wish to show congruence results for weak process relations and prove properties concerning special actions, for example time determinacy and non interference.

Types of actions

- silent action τ and visible actions,
- timed actions and untimed actions,
- higher level and lower level actions.

How to use special actions in formats?

- decide what is observable $\leftrightarrow$ choose weak semantics,
- need specific rules that preserve the special character of actions,
- in view of what is observable, reassess the use of negative premises, copying, lookahead and predicates,
- perhaps, adjust some of the definitions.

Silent action τ and visible actions

τ represents unobservable and uncontrollable behaviour.

Patience rules (τ-rules) embody the uncontrollable, independent character of silent actions:

$$\frac{X_i \xrightarrow{\tau} X_i'}{f(X_1, \cdots, X_i, \cdots, X_n) \xrightarrow{\tau} f(X_1, \cdots, X_i', \cdots, X_n)}$$

The unobservable character of silent action is preserved by disallowing rules like the one below.

$$\frac{X \xrightarrow{\tau} X'}{see{-}\tau(X) \xrightarrow{a} see{-}\tau(X')}$$

Disallow rules that make the unobservable behaviour (according to the chosen weak semantics) observable:

- disallow lookahead, reassess negative premises if $a.\tau.b.\mathbf{0} = a.b.\mathbf{0}$,
- disallow implicit copying, reassess negative premises, if $a.\mathbf{0} + \tau.(a.\mathbf{0} + b.\mathbf{0}) = recX.(a.\mathbf{0} + \tau.(b.\mathbf{0} + \tau.X))$.
- disallow any copying and negative premises if working with testing preorder.

Divergence and the choice of weak semantics

1. Divergence is abstracted away: weak bisimulation, branching bisimulation.

   Formats: De Simone, WB cool, RBB cool, RBB safe: Bloom, Vaandrager, Fokkink.

2. Divergence sensitive weak semantics: testing and refusal preorders, refusal simulation, eager bisimulation and a version of branching bisimulation.

   Formats: τDeS, ISOS, ebo, bbo, rebo, rbbo: Ulidowski, Phillips and Yuen.

   The formats use action rules (r, r'), τ-rules and silent choice rules for all active arguments i ($tau(i)$), and the orderings that satisfy

$$\text{if } r' < r \text{ and } i \in active(r) \text{ then } r' < tau(i)$$

$$\text{if } tau(i) < r \text{ and } i \in active(r') \cup active(higher(r')) \text{ then } r' < r$$

$$\text{not } (tau(i) < tau(i))$$

$$\text{if } i \in implicit\,copies(premises(r)) \text{ then } r < tau(i)$$

Features of RBB safe and rbbo formats

RBB safe $\subset$ panth and rbbo $\subset$ OSOS.

Feature	RBB safe	rbbo
$\frac{H}{C} : X_i \xrightarrow{a} Y_i$	$\surd$	$\surd$
$\frac{H}{C} : t \xrightarrow{a} Y_i$	$\surd : X_i$ **tame**	$\times$
$\frac{H}{C} : X_i \stackrel{a}{\not\rightarrow}$	$\surd : X_i$ **tame**	$\surd : X_i$ **wild, stable**
$\frac{H}{C} : t \stackrel{a}{\not\rightarrow}, \quad Pt, \quad \neg Pt$	$\surd : X_i$ **tame**	$\times$
$\frac{H}{C}$: copying X_i	$\surd : X_i$ **tame**	$\surd : X_i$ **wild, stable**
$\frac{H}{C}$: lookahead	$\times$	$\times$

Feature	RBB safe	rbbo
$\frac{H}{C} : f(\vec{X}) \xrightarrow{a} t$	$\surd$	$\surd$
$\frac{H}{C} : Pf(\vec{X})$	$\surd$	$\times$
$\frac{H}{C}$: copying X_i	$\surd$	$\surd$

If we develop formats of rules with orderings that are more general than OSOS, then

I would like to find subformats for branching bisimulation and eager bisimulation.

Can we find subformats for both versions of other weak semantics?

Timed actions

General assumptions for a format of timed PLs:

- action σ: the passage of one time unit,
- time determinacy and time additivity hold,
- appropriate notion of time synchrony holds,
- semantics based on eager bisimulation.

σ-rules guarantee, in part, time determinacy, e.g.:

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X + Y \xrightarrow{\sigma} X' + Y'}$$

Time determinacy holds because the rules like the one below are disallowed (consider $g(a.b + a.c)$).

$$\frac{X \xrightarrow{a} X'}{g(X) \xrightarrow{\sigma} X'}$$

Operators are grouped into time preserving and time altering operators. We obtain timed rebo format.

Open problems and future research

Operators of timed rebo PL preserve a timed version of rooted eager bisimulation and time determinacy (Ulidowski, Yuen).

If additional constraints are met, other time properties such as maximal progress, patience, urgency, timelock freeness are satisfied.

Maximal progress is

$$\textbf{if} \quad p \xrightarrow{\tau} \quad \textbf{then} \quad p \stackrel{\sigma}{\not\rightarrow}$$

and the additional constraint, where r, r' are any rules for f and J is any set of arguments of f with the same priority, is

$$\textbf{if} \;\; act(r) = \sigma \;\; \textbf{and} \;\; act(r') = \tau \;\; \textbf{and} \;\; active(r) \cup active(r') \subseteq J$$
$$\textbf{then} \;\; r < r'$$

Can this approach be adapted, extended to other notions of discrete time and to real time?

1. The tree and ptree (and other) formats with orderings: the expressiveness and the meaning.

2. Find formats for remaining weak semantics.

3. Consider PLs where different groups of actions play different rôles, identify main desired properties of PLs, choose semantics. Use the "format approach" to find pleasant formats and precise definitions of the properties and semantics that are preserved by the formats.
   - PLs with time,
   - PLs with security,
   - PLs with stochastic delays, probabilities,
   - PLs with action refinement.

4. The link between formats of rules with orderings (e.g. OSOS) and PRSs is intriguing. It would be interesting to verify whether or not
   - PRSs can be given semantics in terms of simpler rules with orderings, and
   - there are interesting formats with orderings that can be represented conveniently by PRSs.

5. Procedures for generating PRSs for PLs based on formats with orderings on rules and for a variety of semantics.

Measuring the Performance of Asynchronous Systems with PAFAS

Walter Vogler, Augsburg

joint work with
Flavio Corradini

1

PAFAS

Process Algebra for Faster Asynchronous Systems

~CCS / TCSP

2

general framework (old stuff)

faster-than relation for processes

- plug faster component into synchronous system → time coordination and system might fail
- faster-than only for asynchronous processes where timing doesn't influence functionality (Moller/Tofts 91)
  timing ??
- only upper time bounds (Lynch 96, PAFAS, Petri nets, TACS)
  [or only lower time bounds (Moller/Tofts, TACS^{II})]
- give worst-case performance of atomic components (= actions)
  ⇒ good for considering worst-case performance

3

classical must-testing (considers worst-case)

- P satisfies test O if every run of P ‖ O reaches success (special action ω)
- P ⊒ Q (implements, improves), if P satisfies all tests that Q satisfies (and maybe even more)
  ⤳ coarsest precongruence, failures semantics

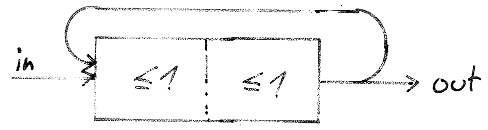
timed testing (WV 93)

- test (O, D) : O – test environment, user
  $D \in \mathbb{N}_0$ – test duration (discrete time)
- P satisfies (O, D) if every run of P ‖ O reaches success within time D
- P ⊒ Q ... (and maybe more users or the same users faster) ⤳ P is faster than Q

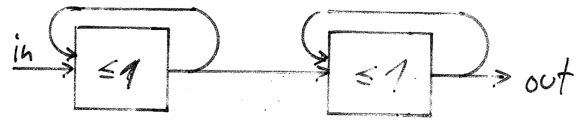
⤳ coarsest precongruence, (timed) refusal traces

4

atomic processing with 2 stages

SEQ 

pipelining

PIPE 

more parallelism, faster?

no! in in $\in L(PIPE) \setminus L(SEQ)$

only under assumptions on user behaviour.

Problems

- Which restricted user classes make sense?
- What sort of results can we expect?
  (presumably no precongruence)

5

PAFAS process algebra with upper time bounds

a incorporates time bound 1

$a.P \xrightarrow{1} \underline{a}.P$

$\underline{a}$: time bound 0 $\qquad \neg\, \underline{a}.P \xrightarrow{1}$

$\underline{a}.P \xrightarrow{a} P$ and $a.P \xrightarrow{a} P$

parallel composition $\|_A$

$a.b.c.P \|_{\{c\}} c.Q \xrightarrow{1} \underline{a}.b.c.P \|_{\{c\}} \underline{c}.Q$

$\xrightarrow{a} b.c.P \|_{\{c\}} \underline{c}.Q \xrightarrow{1}$ (patience)

also: $c.P \|_{\{c\}} \underline{c}.Q \xrightarrow{1} \underline{c}.P \|_{\{c\}} \underline{c}.Q$

⇝ $DL(P)$: discrete traces of P

important result:

continuous time leads to the same faster-than relation

6

formally: testable P (i.e. no ω) satisfies (O,D) if $\forall v \in DL(P \| O)$:

duration of $v > D \Rightarrow \omega$ in v

⇝ faster-than

characterization: (timed) refusal traces

$P \xrightarrow{X}_r P'$ with $X \subseteq Act$:

P performs a partial time step; at this time, at least all $a \in X$ (and τ) can be refused

(possible if context refuses the others)

$\xrightarrow{1} \;\hat{=}\; \xrightarrow{Act}_r$

7

$a.P \xrightarrow{\{a,b\}}_r \underline{a}.P \xrightarrow{a} P$

$\underline{a}.P \xrightarrow{\{b\}}_r \underline{a}.P$, $\quad \neg\, \underline{a}.P \xrightarrow{\{a,..\}}_r$

$\neg\, \underline{\tau}.P \xrightarrow{X}_r$

$\underline{a}.P \|_{\{b\}} \underline{b}.Q \xrightarrow{\{b\}}_r \qquad \not\xrightarrow{\{a,..\}}_r$

$(\underline{a}.P + \underline{b}) \|_{\{b\}} (b.Q + c) \xrightarrow{\{b,c\}}_r$, $\quad \not\xrightarrow{\{a\}}_r$

⇝ $RT(P)$: refusal traces of P

Thm.: P faster than Q iff $RT(P) \subseteq RT(Q)$

Applications:

- buffer implementation
- MUTEX algorithms (Petri nets)

- $w \in RT(P)$ is a witness of slow behavi
- FastAsy (Petri net tool) produces $w \in RT(P) \setminus RT(Q)$ if P is not faster than Q.

8

formally:

SOS-rules ⇝ refusal transition system RTS(P) ⇝ refusal traces

- $X \subseteq sort(P)$ sufficient, $sort(P)$ finite
- maximal X sufficient

Par-Rule

$$\frac{P \xrightarrow{X}_r P',\ Q \xrightarrow{Y}_r Q',\ Z\cap A \subseteq X\cup Y,\ Z\setminus A \subseteq X\cap Y}{P\|_A Q \xrightarrow{Z}_r P'\|_A Q'}$$

$$\frac{\underline{a}+\underline{b}+c \xrightarrow{\{c\}}_r,\quad b \xrightarrow{\{a,b,c\}}_r}{(\underline{a}+\underline{b}+c)\|_{\{b\}}\, b \xrightarrow{\{b,c\}}_r}$$

RTS(P) & RTS(O) determine RTS(P‖O), hence RT(P‖O) and DL(P‖O)

If you know RTS(O), you can read off test result from RTS(P).

9

Par-Rule: the same as for ordinary failures semantics

10

faster-than:

- qualitative (yes/no)
- better for all possible users

Quantification of Performance

performance function of P:

$$p_P(O) = \sup\{n \in \mathbb{N}_0 \mid \exists v \in DL(P\|O): duration(v)=n \text{ and no } \omega \text{ in } v\}$$

Thm.: P faster than Q $\iff p_P \leq p_Q$

possible idea: group tests into $\mathcal{U}_1, \mathcal{U}_2, \ldots$ according to "size";

p_P gives function $\mathbb{N} \to \mathbb{N}_0 \cup \{\infty\}$,

$n \to \sup\{p_P(O) \mid O \in \mathcal{U}_n\}$

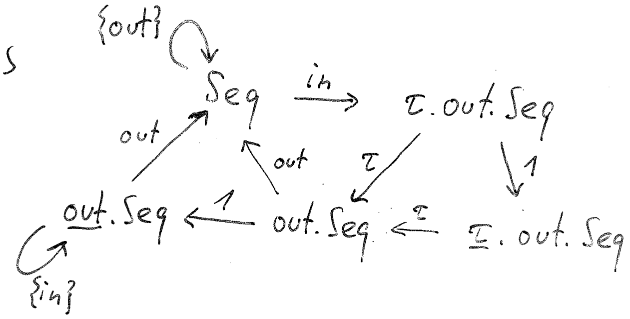
Like usual efficiency of algorithms

11

Seq $\mu x.\ \underline{in}.\ \tau.\ out.\ x$

input 1st stage 2nd stage output

RTS



12

X

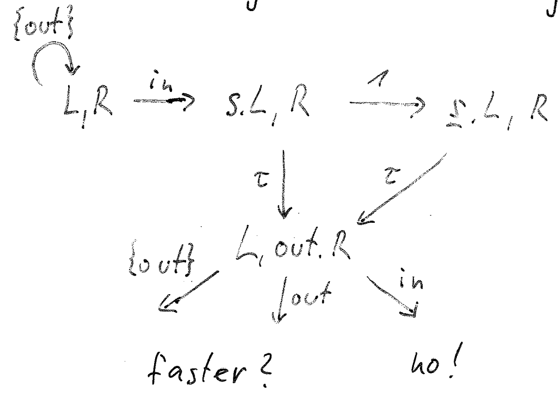
OK □

13

Pipe $((\mu x.\, \underline{in}\,.\, s\,.\, x) \parallel_{\{s\}} (\mu x.\, \underline{s}\,.\, \underline{out}\,.x))$

(L = first part, 1st stage; R = second part, 2nd stage)

RTS



in in ∈ RT(Pipe) \ RT(Seq)

"When waiting for out, user collapses after another in."

User needs Seq / Pipe for his internal coordination.

14

Claim Pipelining is good for sensible users!

$U_n = \underline{in}.\underline{out}.\underline{\omega} \parallel_{\{\omega\}} \cdots \parallel_{\{\omega\}} \underline{in}.\underline{out}.\underline{\omega}$ (n times)

here: abstraction from data (not $\underline{in}_d \cdot \underline{out}_{(d, f(d))} \cdot \underline{\omega}$)

$\mathcal{U}_n = \{U_n\}$

response performance $rp_P(n) = p_P(\mathcal{U}_n)$

Claim $rp_{Pipe} \leq rp_{seq}$

15

- functional correctness (P produces the correct f(d)) has to be checked by other means
- data do not influence performance of P

performance under heavy load

16

Restriction to response processes,
i.e. $P \xrightarrow{w}_r Q$ implies that
$d := \#(in \in w) - \#(out \in w) \geq 0$
and $Q (\overset{out}{\Longrightarrow})^d$.

- P never performs too many out
- P is always able to perform enough
  - but: • may still refuse out for arbitrarily long time
    • may similarly refuse in
  - in both cases: $\exists n: rp_P(n) = \infty$

Thm.: Being a response process is decidable in linear time.

Examples: Seq, Pipe

17

Composing a full time step of $P \| U_n$ for response process P

RTS(P)	RTS(U_n)	
$\emptyset$	{in, out, ω}	not possible
{in}	{out, ω}	no out is missing, {in} not maximal
{out}	{in, ω}	occurs only - when out is missing - "at the end"

rRTS(P): RTS(P), but only time steps
- $\xrightarrow{1}$ or
- $P_1 \xrightarrow{\{out\}}_r$ if out is missing in P_1

18

Path in rRTS(P) is n-critical, if
- $\leq n$ in's, $\leq n-1$ out's
- only time steps 1 before nth in

Thm.: $rp_P(n) = \sup\{$durations of n-critical paths$\}$

$rp_{Seq}(n) = 2n$ $\quad rp_{Pipe}(n) = n+1$

19

Cycle in rRTS(P):
- some time step, no in/out: catastrophic
- otherwise, if only time steps 1:
  average performance $\frac{\#\text{ time steps}}{\#\text{ in}}$
  if maximal: bad cycle

Bad-Cycle Theorem

P finite state response process:
- $\exists$ catastrophic cycle $\Leftrightarrow \exists n: rp_P(n) = \infty$
- Otherwise, rp is asymptotically linear, i.e. $\exists a, c\ \forall n: a \cdot n - c \leq rp_P(n) \leq a \cdot n + c$, and a is average performance of bad cycle

Decidable / computable in cubic time.

Examples

20

U_n – why so restrictive?

$P_{3/4} = \mu x. (\underline{in}.\overline{out}.)^3 \underline{in}.\tau^3.\overline{out}.x$

$P_1 = \mu x. \underline{in}.\tau.\overline{out}.x$

$rp_{3/4}(4n+i) = 3n$

$rp_1(4n+i) = 4n+i$ $\quad n \in \mathbb{N}_0, 0 \le i \le 3$

Clearly, we prefer $P_{3/4}$!

For $U_n' = \|_{\{\omega\}} {}_{i=1}^{n} \underline{in}.\tau.\overline{out}.\underline{\omega}$, we get

$rp_{3/4}(4n+i) = 6n+i$

$rp_1(4n+i) = 4n+i$

Our notion is amortized,
U_n' and the like stress worst-case
for single response. ?

21

Future work

- Find other sensible user classes!
- Find other types of results!

eventually

- general methodology

22

Recent BRICS Notes Series Publications

NS-03-3 **Luca Aceto, Zoltán Ésik, Willem Jan Fokkink, and Anna Ingólfsdóttir, editors. *Slide Reprints from the Workshop on Process Algebra: Open Problems and Future Directions, PA '03,* (Bologna, Italy, 21–25 July, 2003), November 2003. vi+138.**

NS-03-2 **Luca Aceto. *Some of My Favourite Results in Classic Process Algebra*. September 2003. 21 pp. To appear in the *Bulletin of the EATCS*, volume 81, October 2003.**

NS-03-1 **Patrick Cousot, Lisbeth Fajstrup, Eric Goubault, Maurice Herlihy, Kurtz Alexander, Martin Raußen, and Vladimiro Sassone, editors. *Preliminary Proceedings of the Workshop on Geometry and Topology in Concurrency Theory, GETCO '03,* (Marseille, France, September 6, 2003), August 2003. vi+54.**

NS-02-8 **Peter D. Mosses, editor. *Proceedings of the Fourth International Workshop on Action Semantics, AS 2002,* (Copenhagen, Denmark, July 21, 2002), December 2002. vi+133 pp.**

NS-02-7 **Anders Møller. *Document Structure Description 2.0*. December 2002. 29 pp.**

NS-02-6 **Aske Simon Christensen and Anders Møller. *JWIG User Manual*. October 2002. 35 pp.**

NS-02-5 **Patrick Cousot, Lisbeth Fajstrup, Eric Goubault, Maurice Herlihy, Martin Raußen, and Vladimiro Sassone, editors. *Preliminary Proceedings of the Workshop on Geometry and Topology in Concurrency Theory, GETCO '02,* (Toulouse, France, October 30–31, 2002), October 2002. vi+97.**

NS-02-4 **Daniel Gudbjartsson, Anna Ingólfsdóttir, and Augustin Kong. *An BDD-Based Implementation of the Allegro Software*. August 2002. 2 pp.**

NS-02-3 **Walter Vogler and Kim G. Larsen, editors. *Preliminary Proceedings of the 3rd International Workshop on Models for Time-Critical Systems, MTCS '02,* (Brno, Czech Republic, August 24, 2002), August 2002. vi+141 pp.**

NS-02-2 **Zoltán Ésik and Anna Ingólfsdóttir, editors. *Preliminary Proceedings of the Workshop on Fixed Points in Computer Science, FICS '02,* (Copenhagen, Denmark, July 20 and 21, 2002), June 2002. iv+81 pp.**