Beyond Mobility —
what next after CSP/π

A Presentation to: CPA

Michael Goldsmith
e-Security Group
Digital Laboratory
University of Warwick

2nd November 2009

Outline

Historical Background

Bigraphs

Reaction Systems

Semantic Challenges

Programming Challenges

Outline

Historical Background

Bigraphs

Reaction Systems

Semantic Challenges

Programming Challenges

A (blinkered) Potted History of Process Algebra

In the Very Beginning, there was CCS...

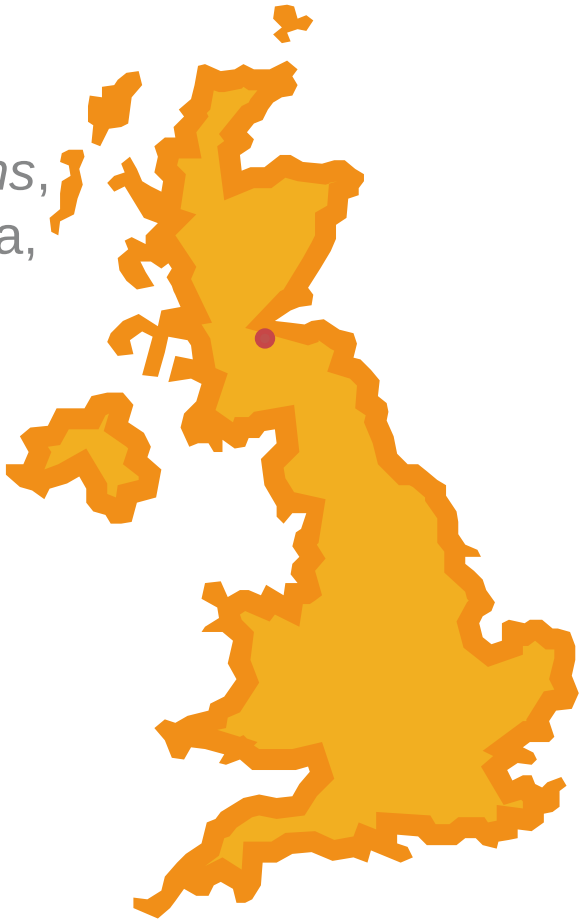
A (blinkered) Potted History of Process Algebra

In the Very Beginning, there was CCS...

Robin Milner
An approach to the semantics of parallel programs,
Proceedings of Convegno di Informatica Teoretica,
1973.

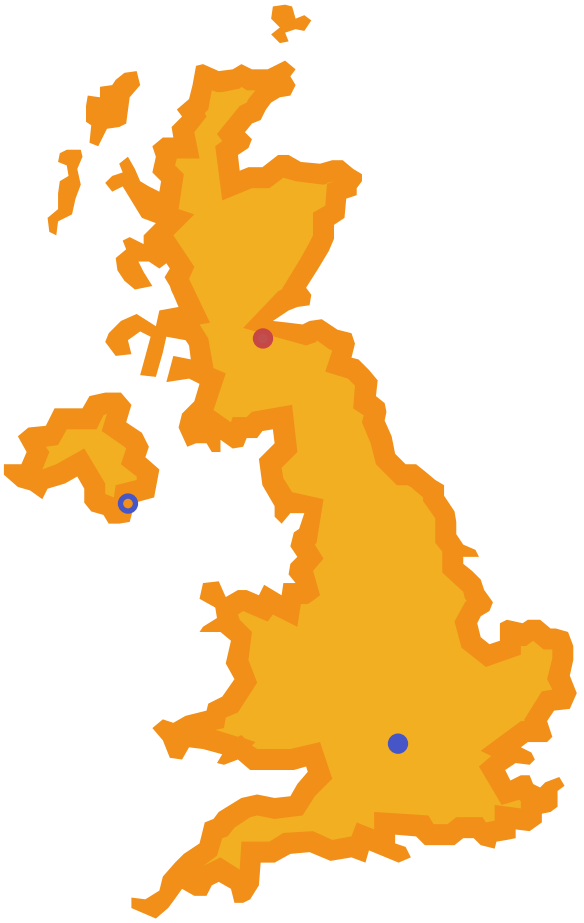
Robin Milner,
A Calculus of Communicating Systems,
Springer Lecture Notes in Computer Science **92**,
1980.

Robin Milner,
Communication and Concurrency,
Prentice Hall,
1989.



A (blinkered) Potted History of Process Algebra

In the Beginning, there were CCS and CSP



A (blinkered) Potted History of Process Algebra

In the Beginning, there were CCS and CSP

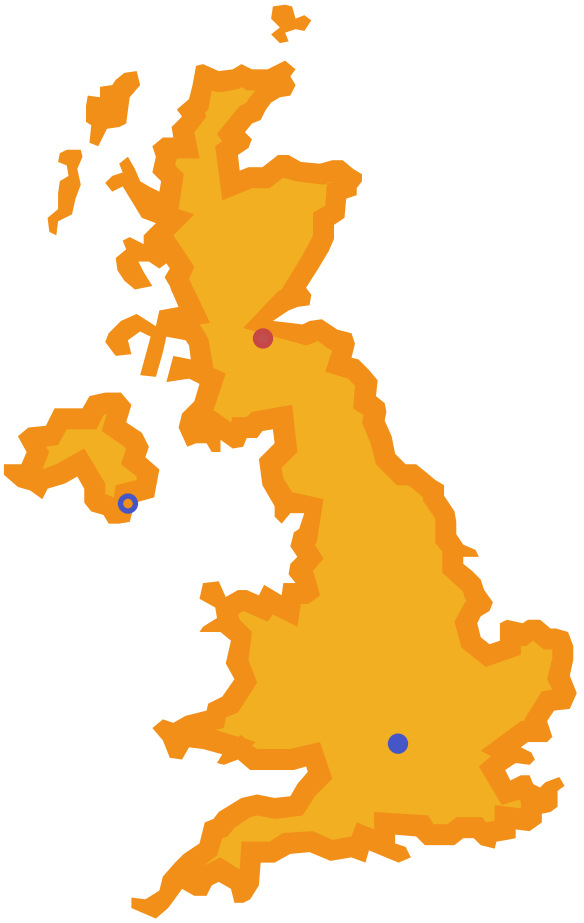
C.A.R. Hoare,
Communicating Sequential Processes,
Communications of the ACM, **21**(8), 1978.

S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe,
A theory of communicating sequential processes,
Journal of the ACM, **31**(3), 1984.

C.A.R. Hoare,
Communicating Sequential Processes,
Prentice Hall, 1985.

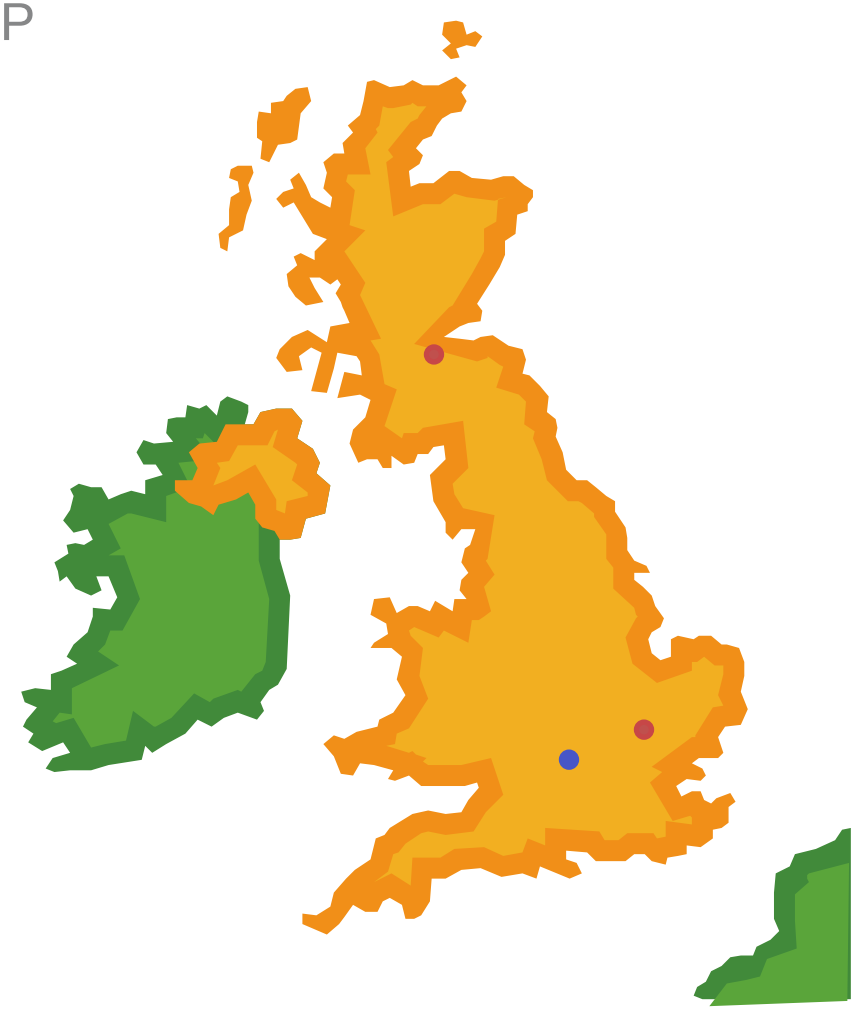
A.W. Roscoe,
The Theory and Practice of Concurrency,
Prentice-Hall, 1998.

[*watch this space*]



A (blinkered) Potted History of Process Algebra

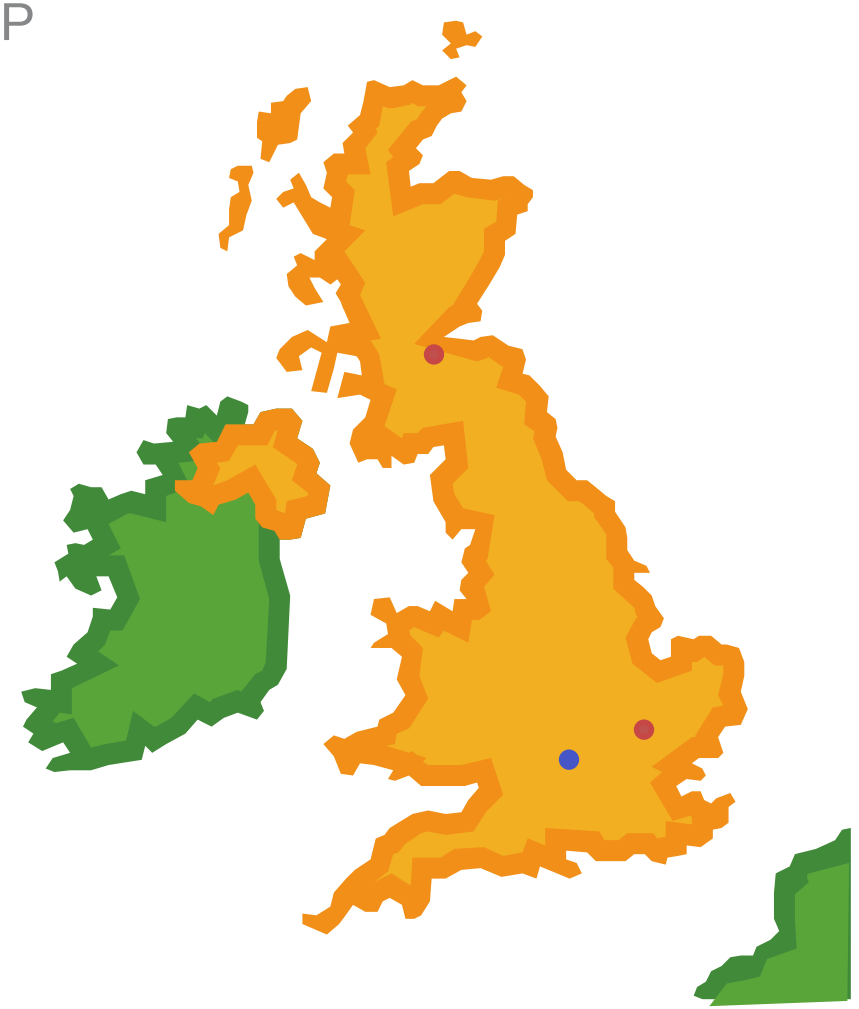
In the Beginning, there were CCS and CSP



... and some other efforts on the continent...

A (blinkered) Potted History of Process Algebra

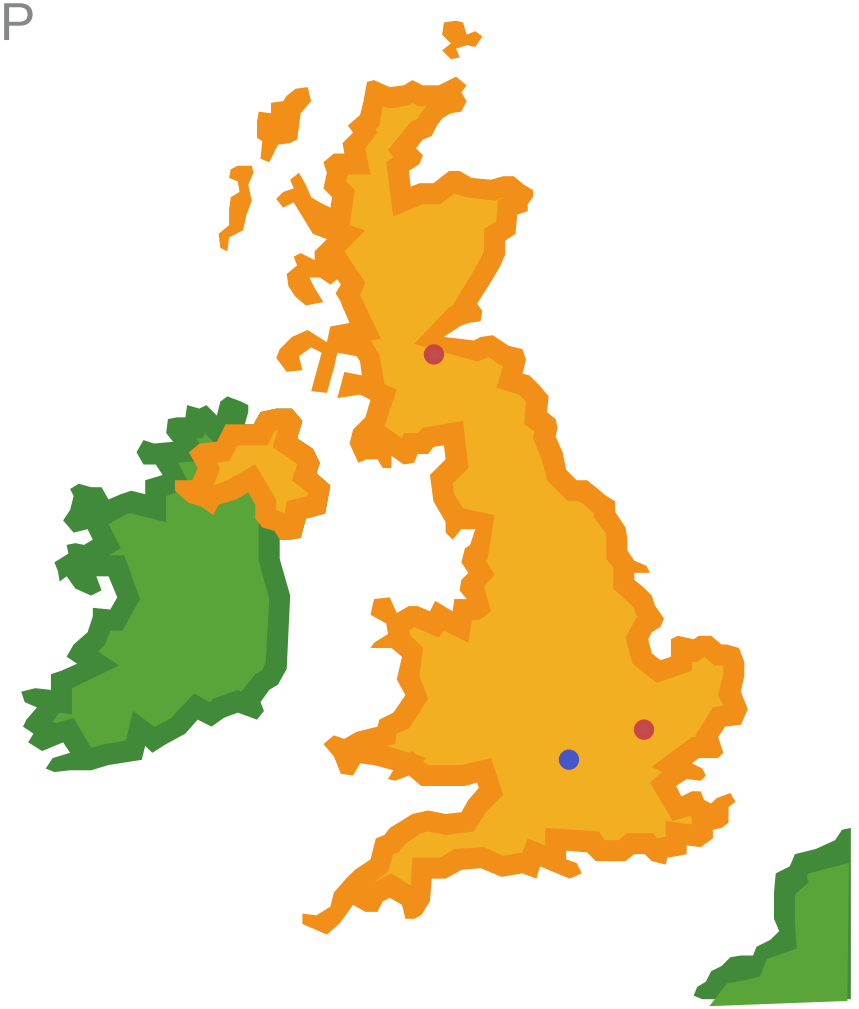
In the Beginning, there were CCS and CSP



... and some other efforts on the continent... ACP

A (blinkered) Potted History of Process Algebra

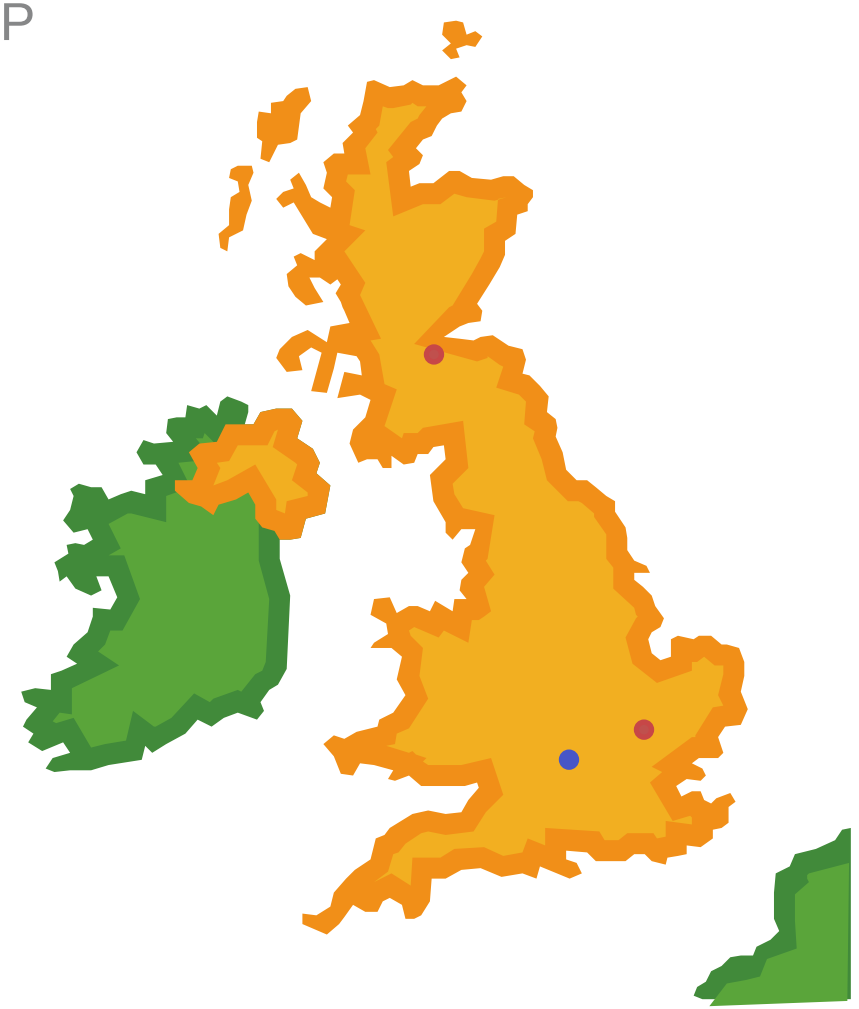
In the Beginning, there were CCS and CSP



... and some other efforts on the continent... RSL

A (blinkered) Potted History of Process Algebra

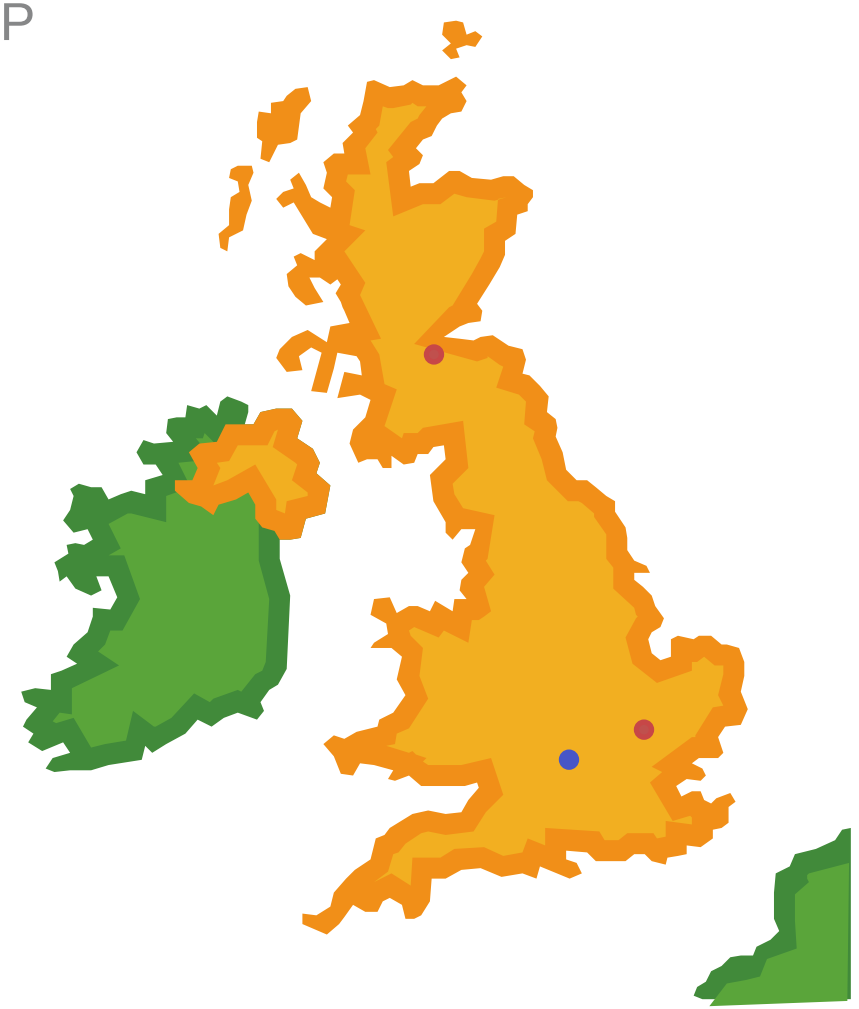
In the Beginning, there were CCS and CSP



... and some other efforts on the continent... MEIJE

A (blinkered) Potted History of Process Algebra

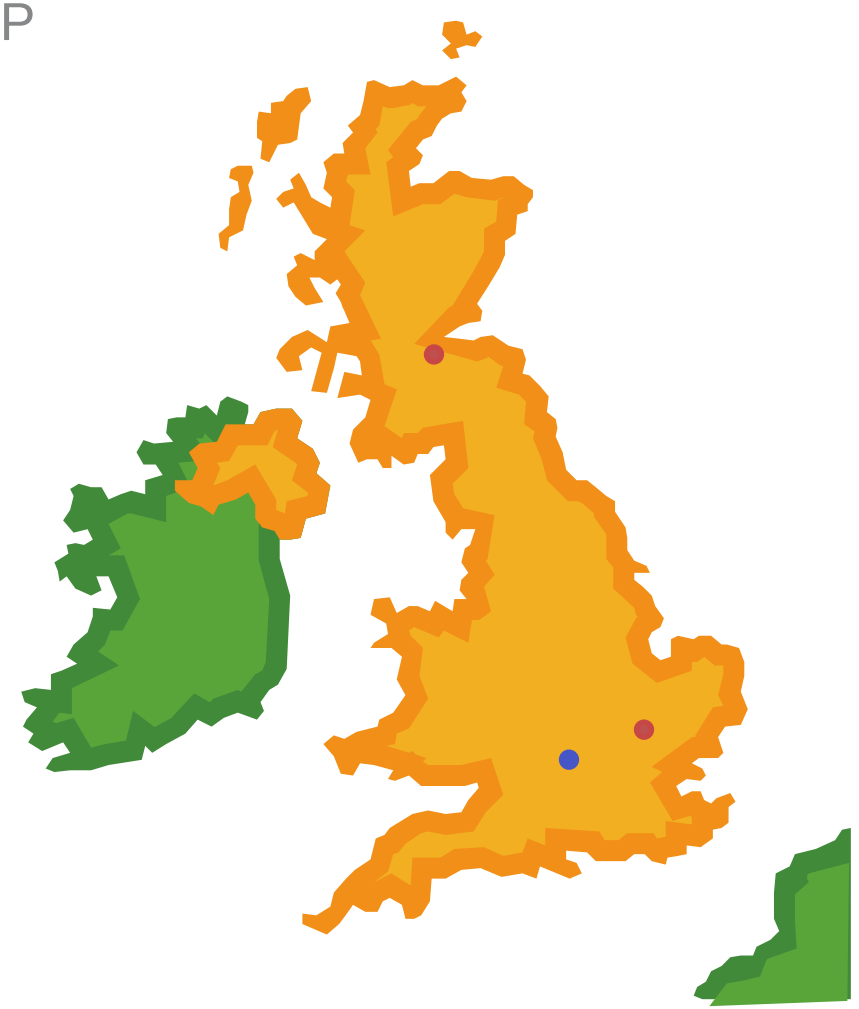
In the Beginning, there were CCS and CSP



... and some other efforts on the continent... μCRL

A (blinkered) Potted History of Process Algebra

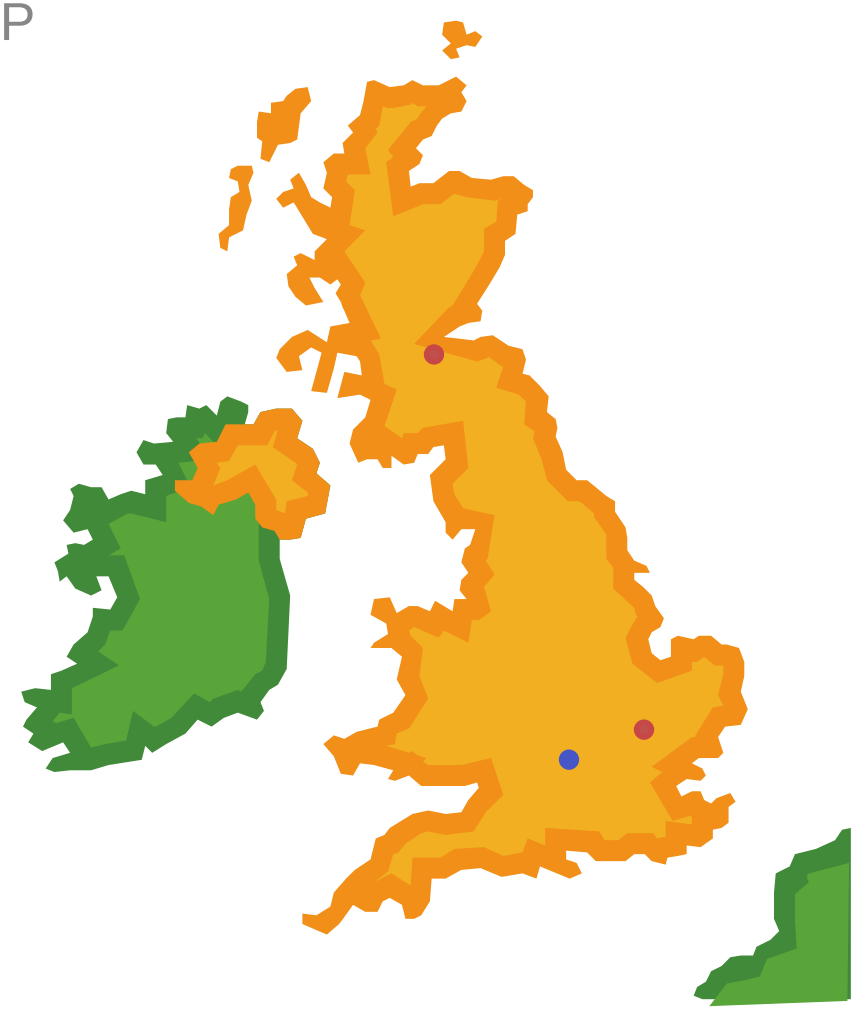
In the Beginning, there were CCS and CSP



... and some other efforts on the continent... PSF

A (blinkered) Potted History of Process Algebra

In the Beginning, there were CCS and CSP



... and some other efforts on the continent... mCRL

A (blinkered) Potted History of Process Algebra

In the Beginning, there were CCS and CSP

... but for the sake of this story, there were Oxford and Edinburgh/Cambridge

A (blinkered) Potted History of Process Algebra

In the Beginning, there were CCS and CSP

Two households, both alike in dignity…

CCS and CSP — Alike and yet Unalike

CCS	CSP
atomic synchronisable events	atomic synchronisable events
point-to-point communication	*multi-way* synchronisation
synchronisation between *duals*	synchronisation between *equals*
result of synchronisation hidden	result of synchronisation visible
abstraction operator is *blocking*	abstraction operator is *hiding*
single simple parallel operator	interleaving, sharing, fully synch...
no need for alphabets	need alphabets or interface sets
allows auto-concurrency	allows auto-concurrency
only one choice operator	distinguish internal and external
only one terminal process	termination distinct from deadlock

occam Draws on Both

occam

(more-or-less) atomic synchronised communication of data

point-to-point communication

synchronisation between *inputs* and *outputs* on a given channel

result of synchronisation hidden

abstraction operator (channel declaration/**`PAR`**) is *blocking*

single simple parallel operator, **`PAR`** (and generalised variant)

alphabets inferred for usage checking

does not allow auto-concurrency (violates channel usage rules)

only one choice operator, **`ALT`** (and generalised variant)

termination (**`SKIP`**) distinguished from deadlock (**`STOP`**)

So **occam** is more CCS-like?

occam and CSP

occam actually lies squarely in the intersection of the two languages

- the CCS features are all restrictions/simplifications

Long historic connection between Oxford and **inmos**

- **occam** Transformation System (algebraic semantics – common)
- analysis and instrumentation of high-fidelity occam models of T9000 transputer
- denotational semantics of **occam 2** (foreign to CCS bisimulation-based approach)

... both at University and at Formal Systems

- culminating in FDR refinement checker
  - early version applied to verification of T9000's Virtual Channel Processor
  - verification of CSP designs implemented in/abstracted from **occam**
    - Draper Labs' quad-transputer reliable computing module
  - [analysis of cryptoprotocols (with Lowe) and information flow ⇨ e-Security]

Mobility [*i.e., the ability to move processes and channel(-end)s around*]

Always been a practitioner's goal:

- load balancing
- agent-based services

Implemented in SPOC ~1991 – long before any theoretical underpinnings

π-Calculus adds name creation and communication

- Milner, *Communicating and Mobile Systems: the Pi-Calculus*, CUP, 1999.
- Sangiorgi & Walker, *The π-calculus: A Theory of Mobile Processes*, CUP, 2001.
- process migration effected by moving the environment around the process

Realisations in JCSP, **occam-π**, CHP, CSO, ...

But this notion of mobility is very abstract

- destinations are at best logical processors or TCP/IP sockets
- little notion of even abstract "geography" – adjacency, …

Outline

Historical Background

Bigraphs

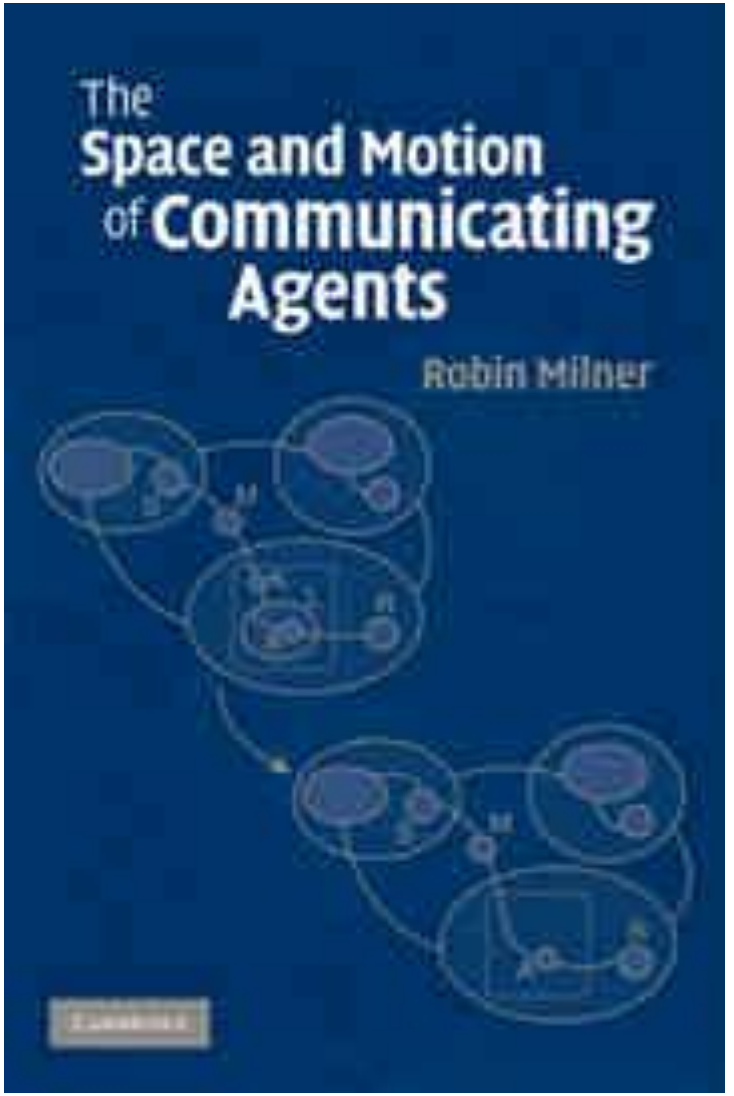
Reaction Systems

Semantic Challenges

Programming Challenges

R. Milner,
The Space and Motion of Communicating Agents,
Cambridge University Press,
2009.

http://www.cl.cam.ac.uk/~rm135



Cover picture "borrowed" from Cambridge University Press website, http://www.cambridge.org/

Bigraphs

Two graphs, with a shared set of nodes and individual peripheries

The *Place Graph* is a forest (of rose trees)

- external interface is its vector of roots
- internal interface is the vector of its *sites* – place-holder leaves of the trees
- represents logical – *or physical* – structure (drawn using nesting)
  - a node is *located* in its parent
  - two sibling nodes are *adjacent*

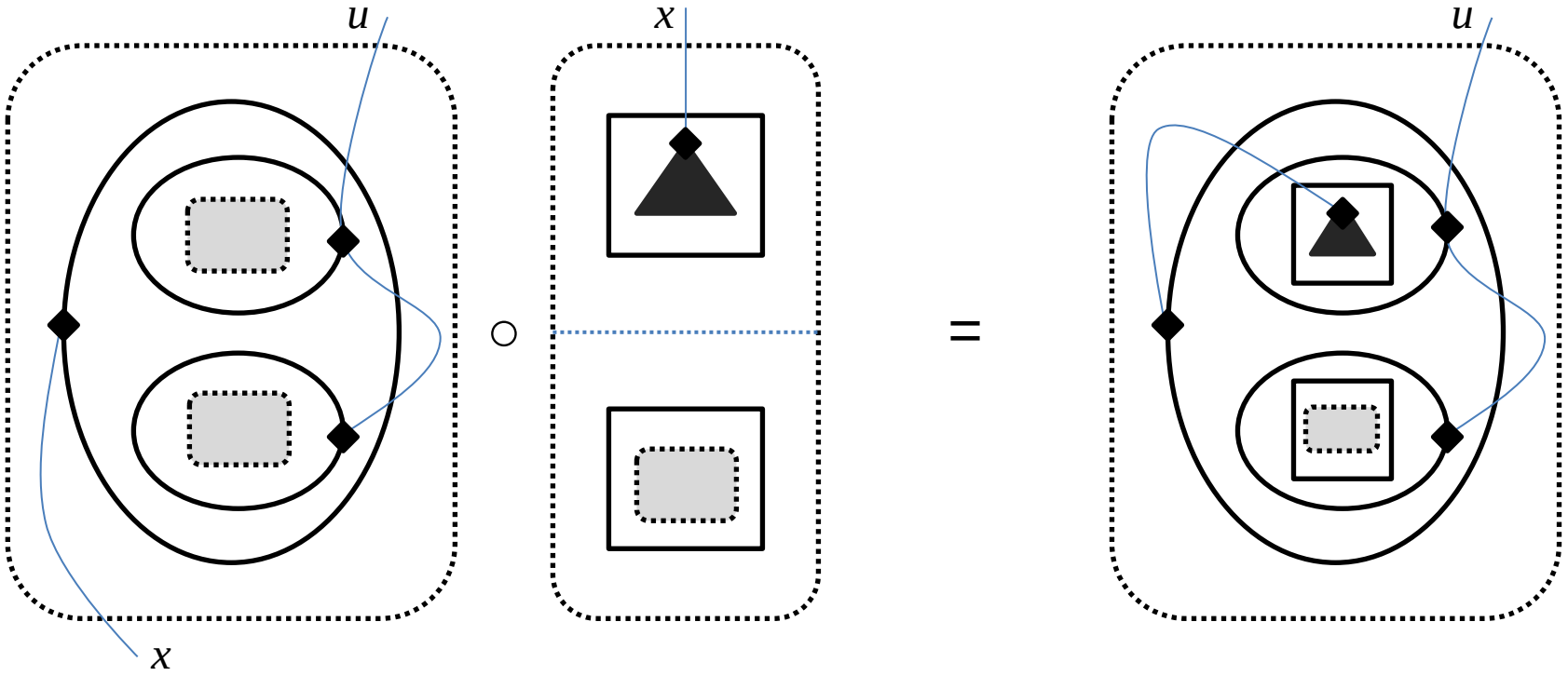
The *Link Graph* relates nodes to one another and to interface elements

- a hypergraph – not just binary relations
- both external and internal interfaces are finite sets of names to facilitate “plumbing”
- might represent communication channels (or barriers)…
- … or equally more abstract relations

Bigraph Composition

If you have two bigraphs, with the external interface of one compatible with the internal interface of the other, you can compose them

- graft the trees from one forest into the sites in the other (#roots = #sites)
- wire together the external interface of one to the corresponding names of the other



The C-Word

Thus you can (if so minded) view bigraphs as arrows in a category

- between their inner and outer interfaces

Left hand graphs in example have type

- $(2,\{x\}) \rightarrow (1,\{u\})$
- $(1,\varnothing) \rightarrow (2,\{x\})$

Thus the inner interface of the first equals the outer interface of the second and we may compose them

The composition has type

- $(1,\varnothing) \rightarrow (1,\{u\})$

There are identity arrows id for each interface

- one site per root
- wire all names straight through

Flexibility Demands Discipline

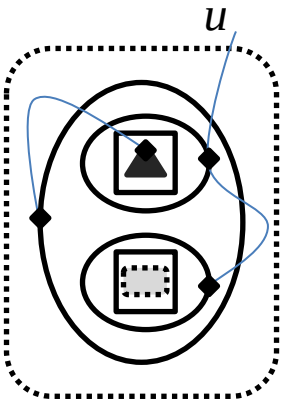
Arbitrary bigraphs would not be terribly useful

- need to be able to understand role of various nodes (and links) in the bigraph

A *signature* assigns an *arity* (number of ports) to each of finite *kinds* of *control*

- different controls may be drawn with different shapes and/or labelled with their kind
- *sorting disciplines* provide well-formedness criteria
  - *"polygons may only be nested inside polygons with more edges"*
  - *"polygons and rounded controls must alternate (as parents and children")*
  - *"rectangles have precisely one child"*
- may equally constrain links
  - define controls to have active and passive ports and require that "*links connect no more than one active port together*"

Well-formedness criteria constrain validity of compositions

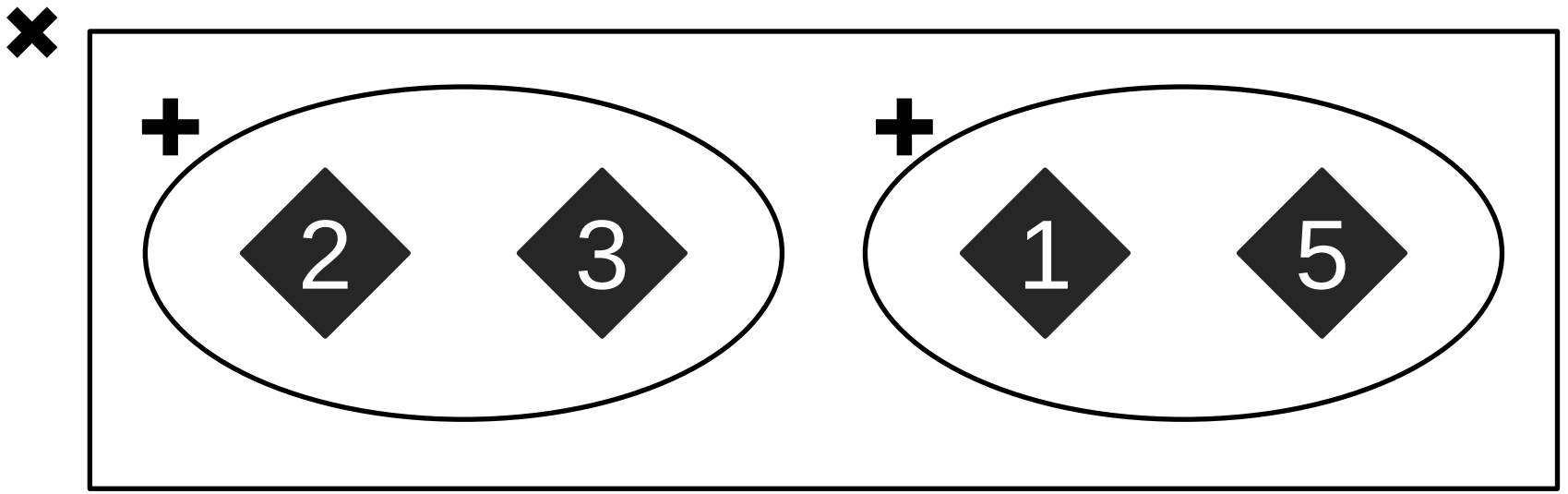


Bigraphs are State Representations

Can vary from pure place graphs (no link-graph edges)

- e.g. syntax trees for arithmetic expressions:

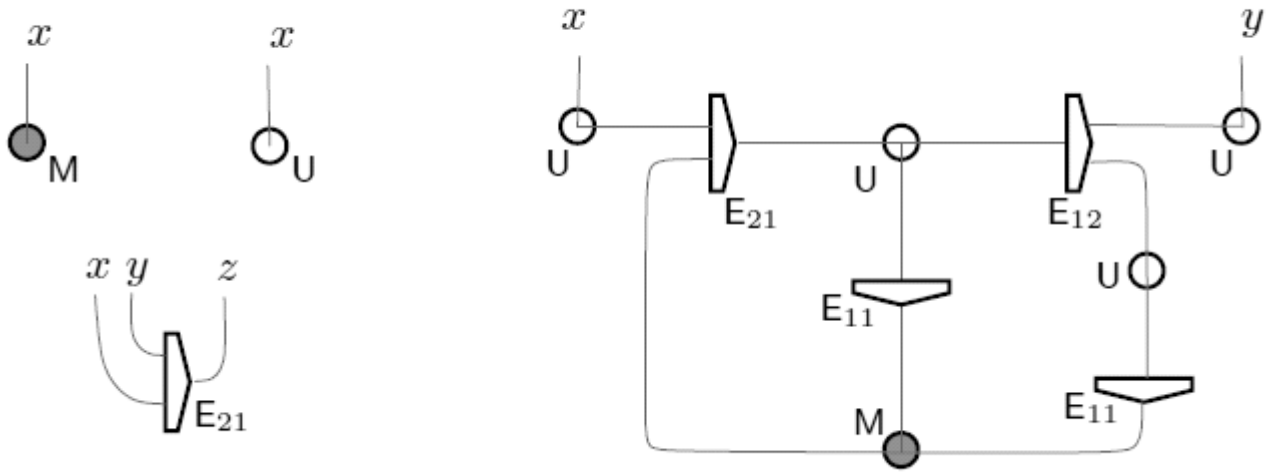
$(2 + 3) \times (1 + 5)$



Bigraphs are State Representations

... to pure link graphs (every node its own root, no sites)

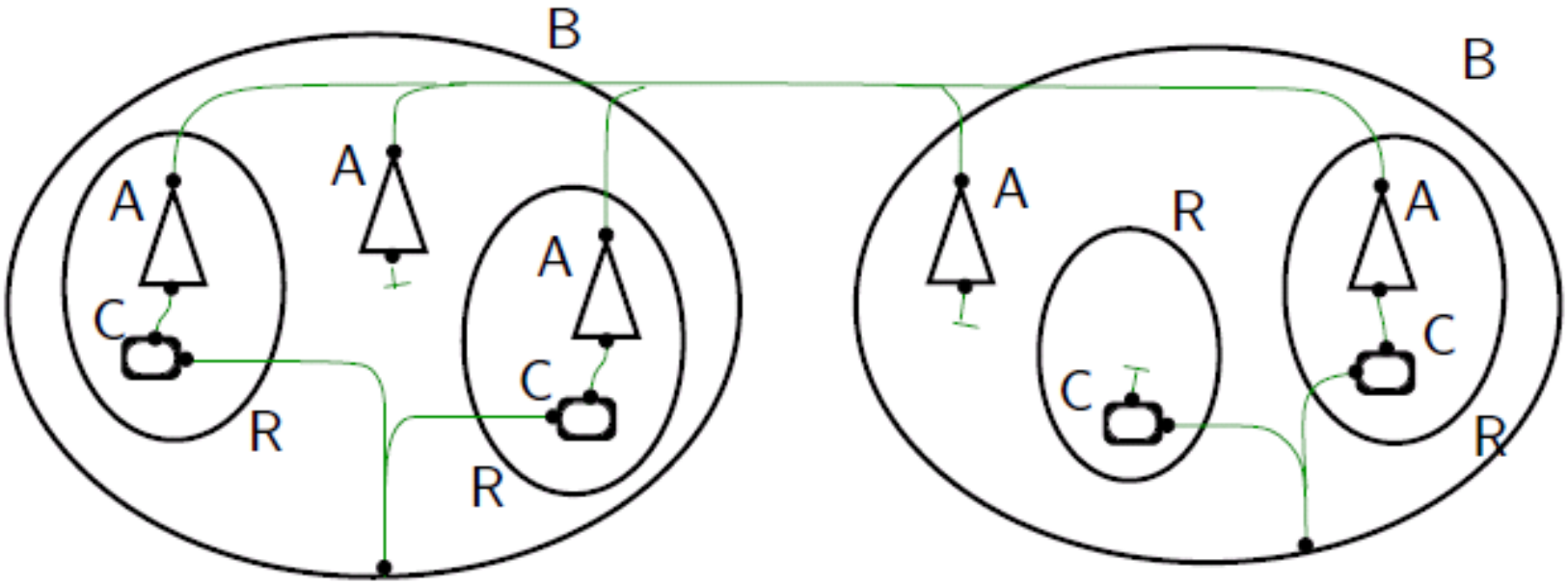
- e.g. Milner's example of condition-event Petri nets



Bigraphs are State Representations

Most, of course, have non-trivial elements of both

- e.g. Milner's example of a "built environment"



Don’t Themselves Describe Dynamic Behaviour

A process-algebra term is equally just a state representation

... but if you understand what the term “means”, then you know how it evolves

Bigraphs have meaning at too many levels for this to be true

- you can have a good intuitive understanding of the way that buildings, rooms, computers and people interact, without necessarily being able to guess the rules that the model imposes to govern the evolution of their interactions

We need some way of describing the dynamics of a bigraphically presented system

Outline

Historical Background

Bigraphs

Reaction Systems

Semantic Challenges

Programming Challenges

Redex and Reactum

A bigraphical reaction system is defined by a number of reaction rules

Each consists of two bigraphs of the same (atomic) type

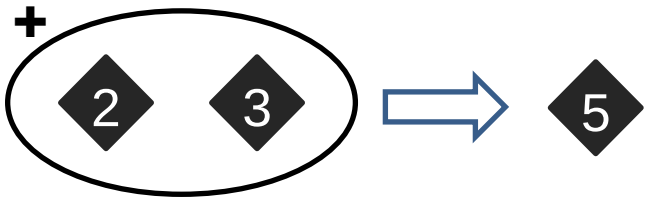
The left-hand side, the *redex*, must match part of bigraph describing the state

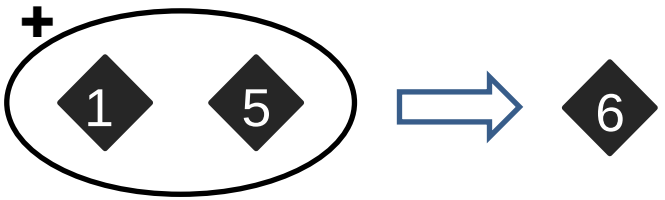
The result of the transition is what you get when you substitute the right-hand side of the rule, the *reactum*, for the part that matched

... or at least, that's the simplified version of the story

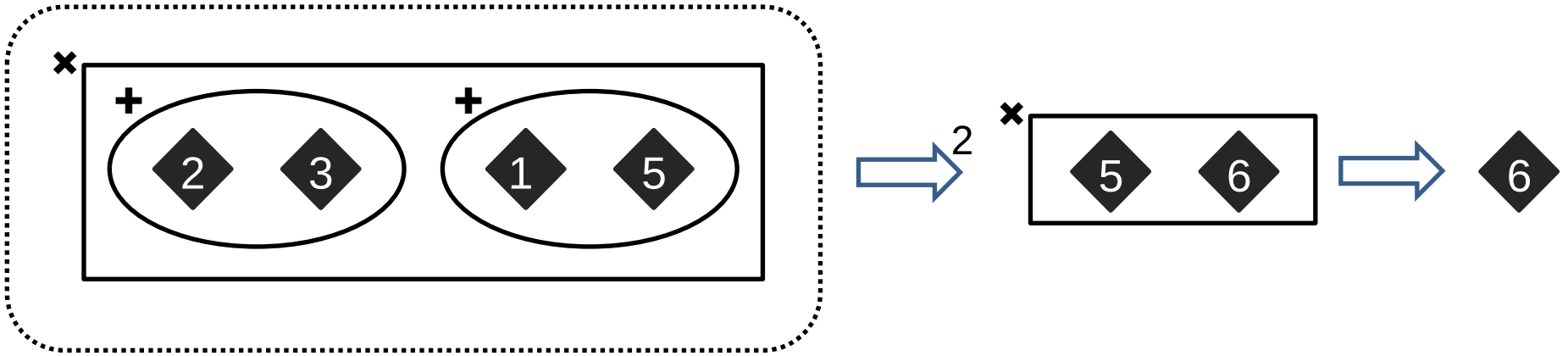
Familiar Reduction Systems

Addition and multiplication tables:





(modulo 8)



Familiar Reduction Systems

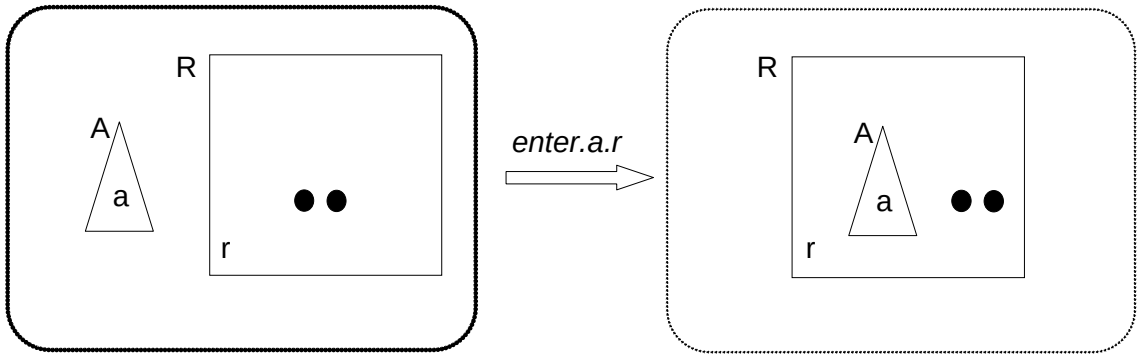
Lambda calculus

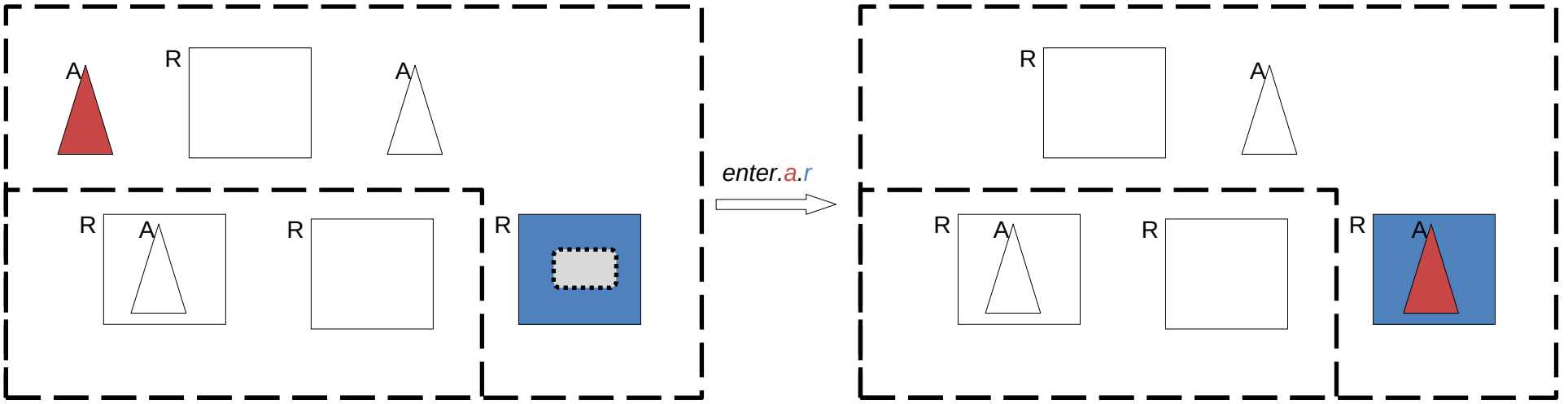
$$(\lambda x.E)\ E' \Rightarrow E[E'/x]$$

$$(\lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))) \Rightarrow (\lambda f.f\ ((\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x)))) \Rightarrow (\lambda f.f\ (f\ ((\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))))) \Rightarrow \ldots$$

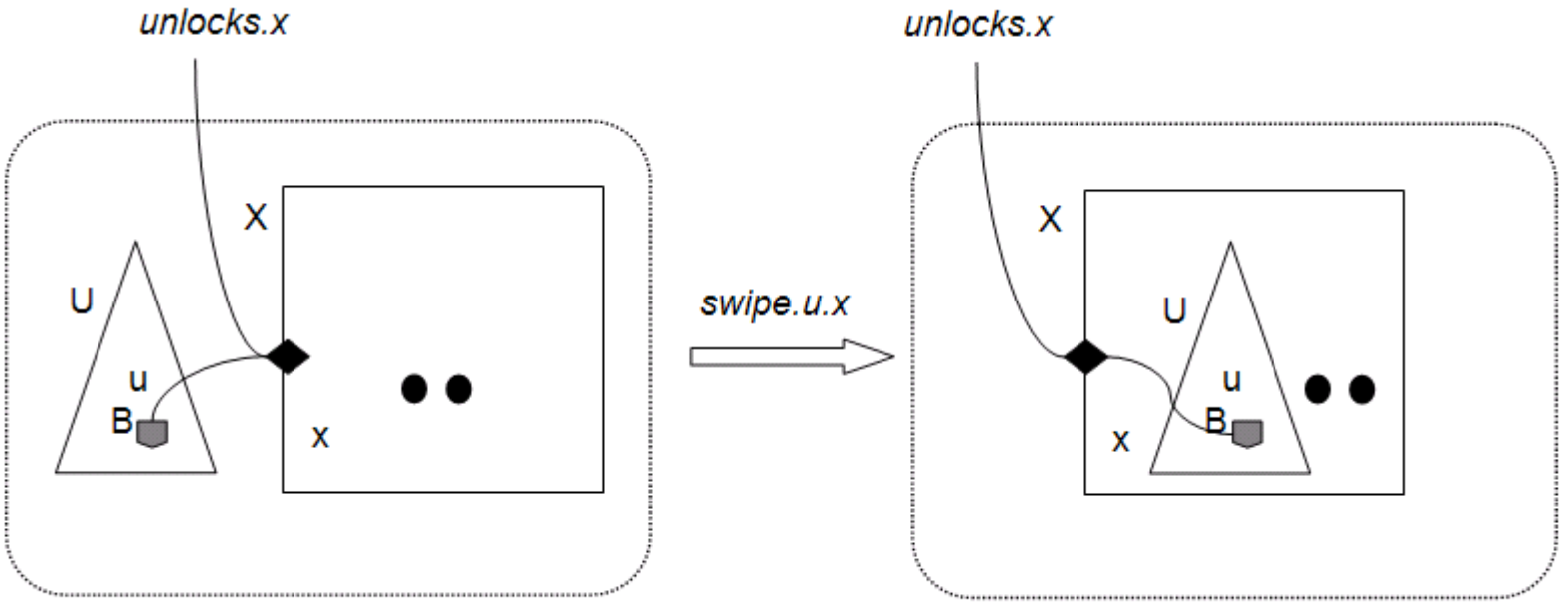
- Don't necessarily want to be able to apply rules *everywhere* in the term
  - head-normal reduction – only outermost application control
  - normal-order reduction – only leftmost application control (even if inside λ)
  - applicative-order reduction – rightmost application control outside λ
- Bigraphs allow places to be marked as *active* or *inactive* to achieve such effects

Bigraphical Reactive Systems – place only

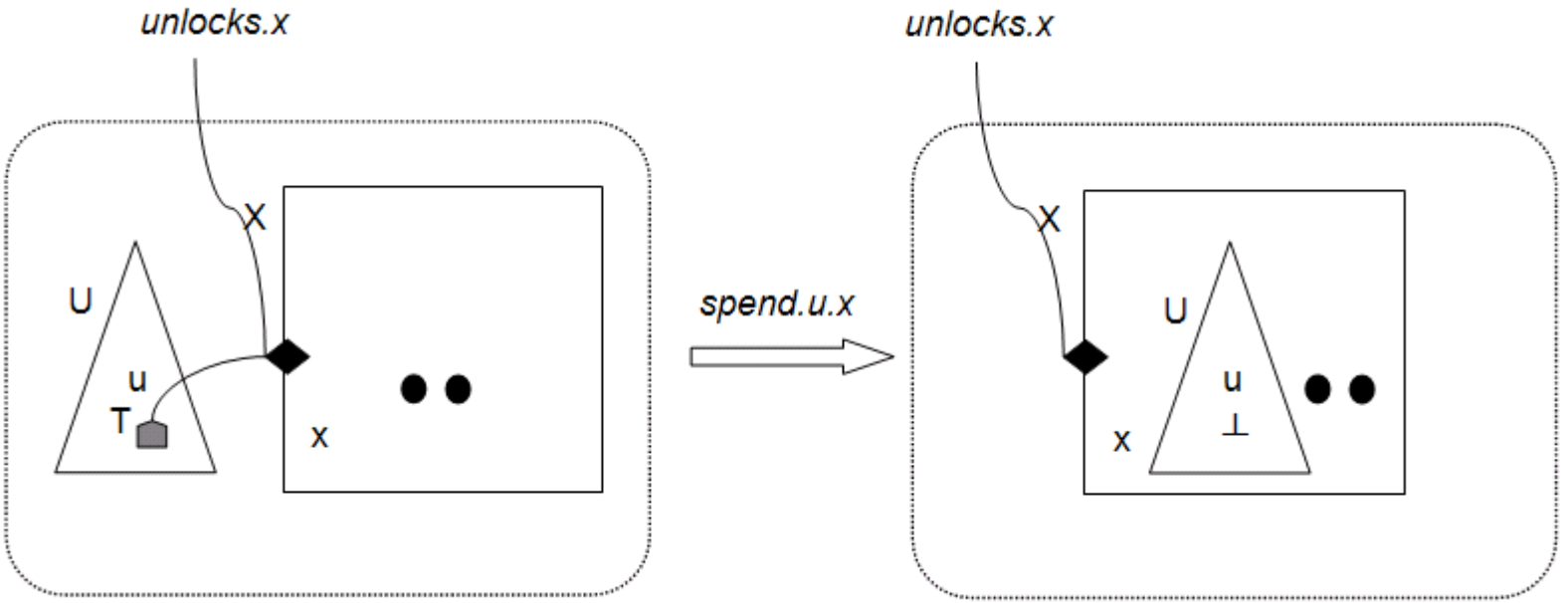




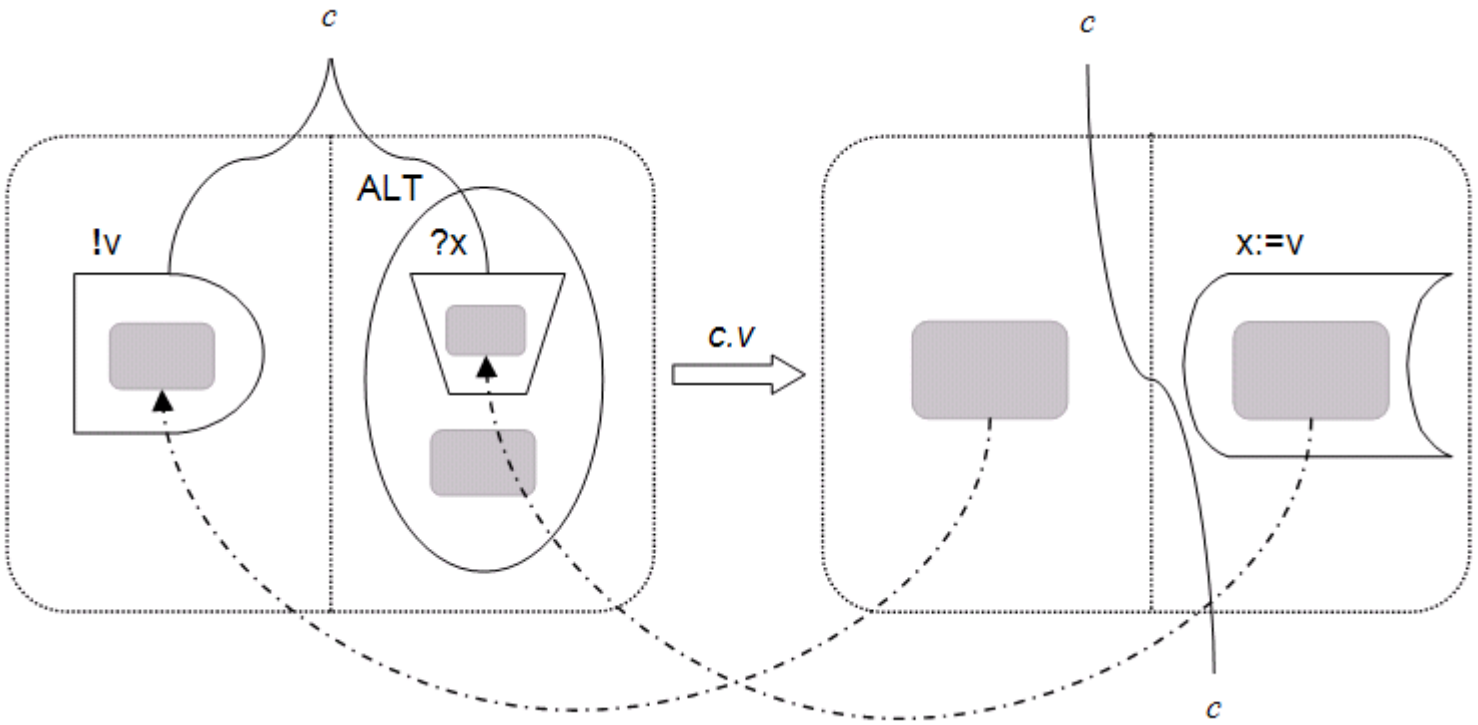
Bigraphical Reactive Systems – link as property



Bigraphical Reactive Systems – non-conservation



BRS: Remote Interaction – CSP/**occam**



Outline

Historical Background

Bigraphs

Reaction Systems

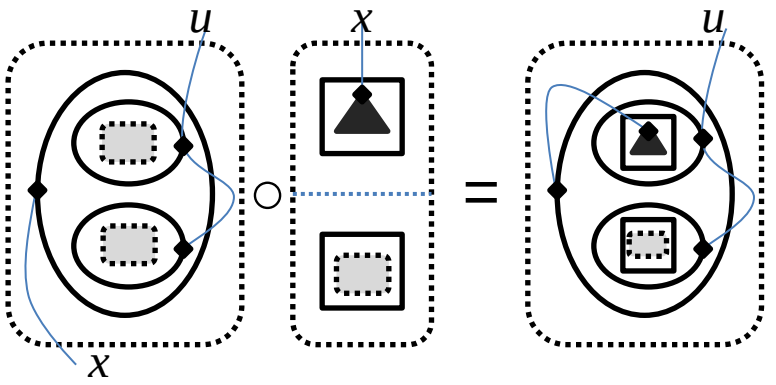
Semantic Challenges

Programming Challenges

Canonical Labelled Transition System of a BRS

… or The Full Story About Matching Redexes

Recall the definition of composition of bigraphs



A reaction rule (r,r') is *applicable* to a ground bigraph b of a reactive system B

iff there exist contexts (other bigraphs) Γ and Δ such that $\Gamma \circ b = \Delta \circ r$

Under these circumstances, b has a Γ-transition: $b \xrightarrow{\Gamma} \Delta \circ r'$

The collection of all such transitions gives the labelled transition system for B

(Why not allow non-ground redexes and source bigraphs, and allow unifying inner contexts?)

What's Wrong with That?

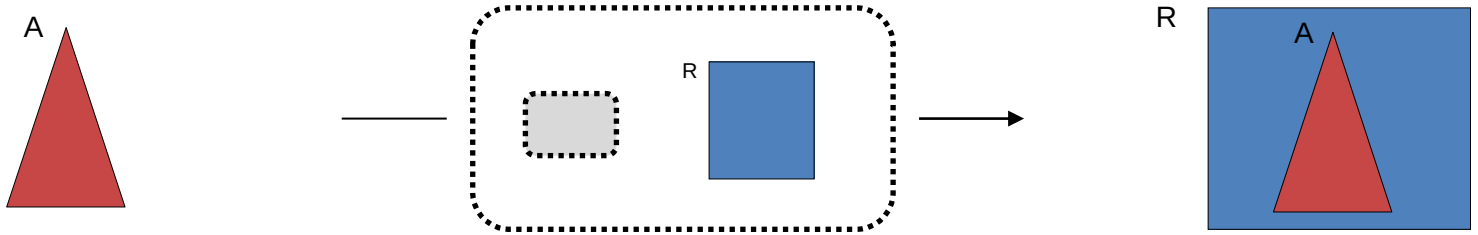
Two things: one definite, one debatable

If $\Gamma \circ b = \Delta \circ r$, then for any compatible bigraph Ω, $(\Omega \circ \Gamma) \circ b = (\Omega \circ \Delta) \circ r$

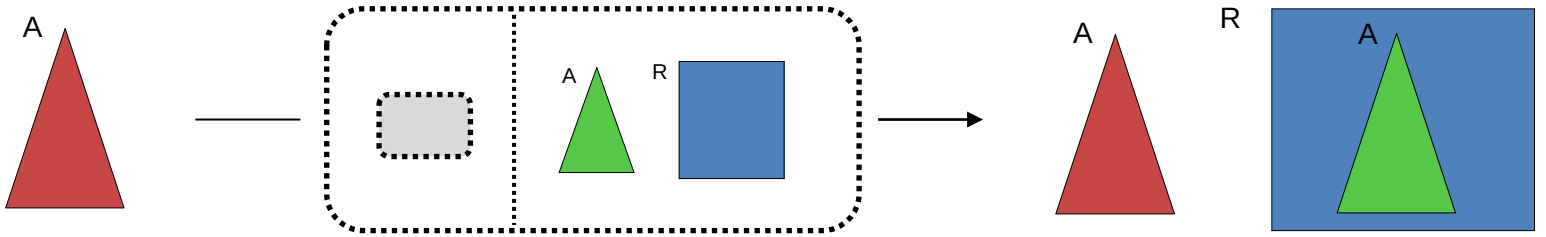
… so in general there are infinitely many, arbitrarily large labels

- fortunately can show *minimal transition system* equivalent (in some exact sense)
  - under reasonably weak assumptions

If b does a Γ-transition, then the resulting bigraph retains the "baggage" from Γ



Unengaged Transitions



The red agent plays no role in this transition – but the canonical LTS includes it and we must continue to explore the behaviour of the augmented system

Fortunately, for a *nice* BRS, we can restrict our attention to the *prime engaged* transition system, which excludes such monsters

- only consider redexes which “intersect” the source bigraph

Niceness?

Unfortunately...

A BRS is nice if it is safe*,* simple*,* unary*,* affine *and* tight.

- *a BRS is* safe *if its sorting is so*
  - *a sorting is* safe *if its forgetful functor creates RPOs and reflects pushouts*
- *a BRS is* simple*, or* unary*, if all its reaction rules are so*
  - *a reaction rule is* simple*, or* unary*, if its redex is so*
  - *a redex is* simple *if it is* open, guarding*, and* inner-injective
  - *a redex is* unary *if its outer face is unary*
- *a BRS is* affine *when all its rules are so*
  - *a rule is* affine *if its instantiation map is injective*
- *a BRS is* tight *if every redex of a rule is tight*
  - *a redex is* tight *if every* unary split *for it is tight*
    - *a split is* tight *if some port in each side of the composition is connected to the other*

Opacity of Transitions

As in CCS, transitions become trivialised (too?) early

Any transition that a bigraph can do “without external assistance” is identified

- if the redex is contained in the source, the minimal Γ context is the identity arrow id

This could be a problem:

Consider a system with two slots, and transitions that fills an empty slot with an adjacent token, and transitions that empty slots without external help

Start it with one slot filled

Then a set of rules which prefers to empty one slot over the other gives a BRS which is bisimilar to one which uses round-robin

But one can (nondeterministically) starve service for our initial token

The other can’t

Intuitive Transition Systems

Cunning coding can (often? always?) rescue the situation

- add links and transitions involving them to monitor the situation

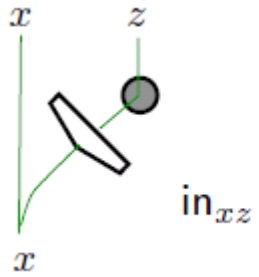
But the “what contexts let me do extra things” labels are rarely what we want

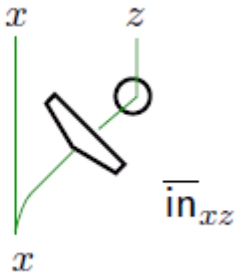
In most cases, we are arguably interested in which transitions (can) fire

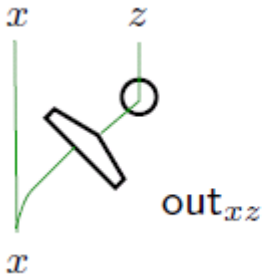
- not what we need to do to enable an arbitrary something to happen

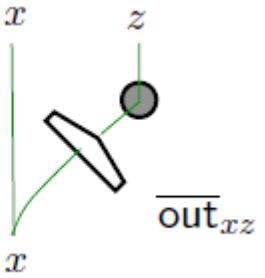
Again, it may be possible to add artificial context-dependence via links

- cf Milner’s condition-event net example, emulating standard experiments with probes









What I Want

I want to be able to reason about which transitions take place where

... to be able to hide internal transitions at my discretion (and look inside later)

... to compare systems according to their interesting behaviour (refinement)

... to have a bigraphical setup suitable for FDR-style checking!

So I'd better do something about it:

announce that I'm about to start a three-year project ✗

announce that I'll use my spare time wondering about it ✗

announce that I expect to be starting a one-year pilot project funded by ONR ✓

Outline

Historical Background

Bigraphs

Reaction Systems

Semantic Challenges

Programming Challenges

Why Is It Worth It?

The interface between physical and cyber-security

- that certain critical functions can only be performed on a guarded console
  - together with models of the measures controlling access to it
- Clive Blackwell's *Spygraphs*

Pervasive applications, location-based services

Vehicle-vehicle and vehicle-roadside telematics

Routing attacks on sensor mote networks

Key management in the wireless edge

Business process modeling

Role-based security architectures

How Do We Implement Things?

A bigraph model is likely (for me) to describe people and environment...

... but also components which will require implementing

What's the appropriate language?

- CCS/CSP inspired occam
- π-calculus inspired current-generation CPA languages
- bigraphs inspire ???

Notation supports local and remote interactions, drastic reconfigurations

- not just local process spawning and load-balancing operations

Who's going to come up with *the* language for pervasive/location-aware code?

Go home and put on your thinking caps!

The Race is On...

Already activity at Bigraphical Programming Languages group (ITU Denmark)

http://www.itu.dk/research/bpl

Reactive XML

… but mostly execution of bigraphs, not how to program inspired by them

Questions?