



Mobility Models and Analysis

Andy Gordon

Microsoft Research

2nd UK-UbiNet Workshop

**Security, Trust, Privacy and Theory for
Ubiquitous Computing**

May 5-7, 2004



The Pi-Calculus and Mobility

- The pi-calculus (MPW89, ...) is a tiny yet highly expressive concurrent language, with precise semantics, rich theory, and several implementations
 - Computation is name-passing between concurrent processes on named channels
 - Each name has a mobile scope, that tracks which processes can and which cannot communicate on the name
 - Pi has spawned a family of related *nominal calculi*
- Part 1 overviews these mobility models, and argues their use as formal methods is at a turning point
- Part 2 discusses one application in particular: the MSRC Samoa project on web services security



Example in the Pi-Calculus

Client: start virtual printer v ; use it:

```
new( $v$ ); (out start( $v$ ) | out  $v$ (job))
```

Server: handles real printer; makes virtual printers.

Driver code

Make new
virtual printer

Virtual
printer at x

```
new( $p$ ); (...  $p$  ... | repeat (inp start( $x$ );  
                                  repeat (inp  $x$ ( $y$ ); out  $p$ ( $y$ )))
```

All the data items are channel names.

All interactions are channel inputs or outputs.

Semantics of the Pi-Calculus

$P \approx Q$ means P and Q
are equivalent states

$P \approx P$
 $P \approx Q \Rightarrow Q \approx P$
 $P \approx Q, Q \approx R \Rightarrow P \approx R$

$P \mid \text{stop} \approx P$
 $P \mid Q \approx Q \mid P$
 $(P \mid Q) \mid R \approx P \mid (Q \mid R)$

$P \rightarrow Q$ means state P
transitions to state Q

$\text{out } x(y_1, \dots, y_n) \mid \text{inp } x(z_1, \dots, z_n); P \rightarrow P\{z_1 \leftarrow y_1, \dots, z_n \leftarrow y_n\}$
 $P \rightarrow Q \Rightarrow \text{new}(x); P \rightarrow \text{new}(x); Q$
 $P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$
 $P' \approx P, P \rightarrow Q, Q \approx Q' \Rightarrow P' \rightarrow Q'$

$\text{repeat stop} \approx \text{stop}$
 $\text{repeat } P \approx P \mid \text{repeat } P$
 $\text{new}(x); \text{stop} \approx \text{stop}$
 $\text{new}(x); \text{new}(y); P \approx \text{new}(y); \text{new}(x); P$
 $\text{new}(x); (P \mid Q) \approx P \mid \text{new}(x); Q$ if
 $x \notin \text{fn}(P)$

$P \approx Q \Rightarrow \text{new}(x); P \approx \text{new}(x); Q$
 $P \approx Q \Rightarrow P \mid R \approx Q \mid R$
 $P \approx Q \Rightarrow \text{repeat } P \approx \text{repeat } Q$
 $P \approx Q \Rightarrow \text{inp } x(z_1, \dots, z_n); P \approx \text{inp } x(z_1, \dots, z_n); Q$

Achieving such simplicity and expressiveness, and
establishing basic theory took many years



Some Applications of Pi Family

Pi as formal semantics

- Functions, objects (pi)
- Crypto (spi, applied pi)
- Async, distributed programming and **algorithms** (pi, join, dpi)
- Thread, device mobility, security perimeters (ambients, seal)
- Unifying frameworks for nominal calculi (action calculi, **bigraphs**)
- Biomolecular modelling (stochastic pi, brane)

Pi as source code

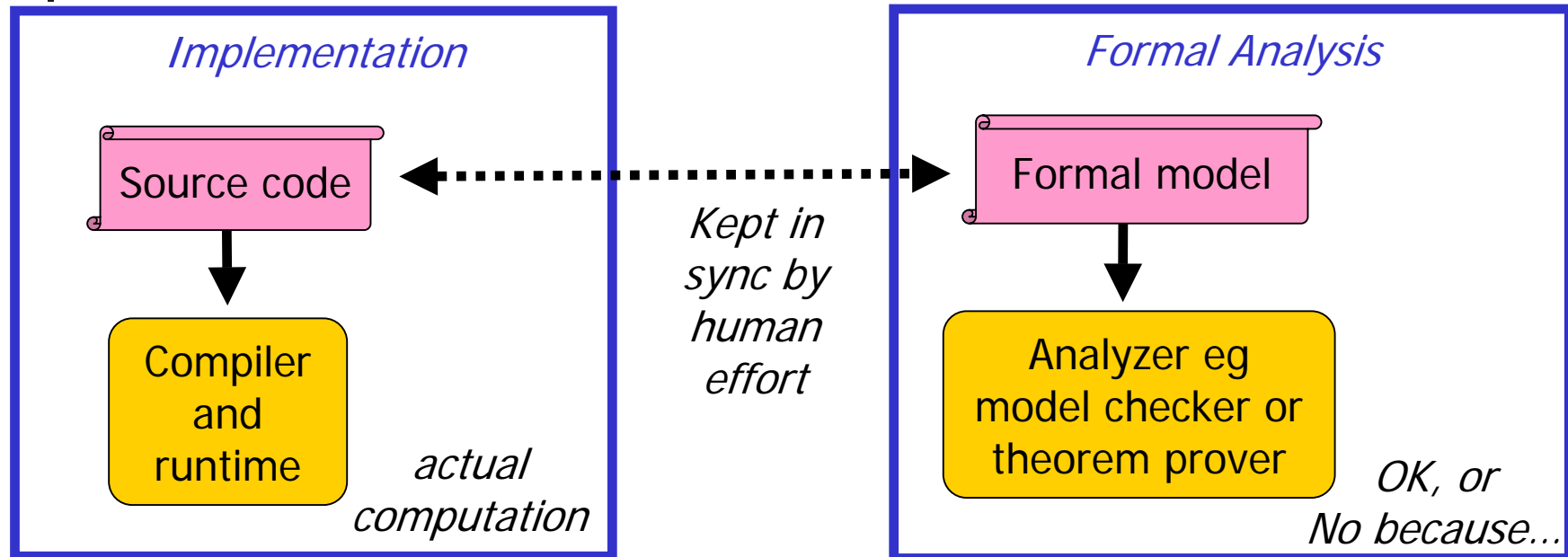
- Pict: channel types, concurrency, objects
- JoCaml: distribution
- Nomadic Pict: mobile agents, transactions
- Iota: untyped XML scripting for home area networking
- XLANG, BPEL: web services composition
- $C\omega$: C# + XML + join

Pi as a formal method

- Equivalences and refinements (eg applied to security)
- Logics: extensional eg HML, intensional eg spatial logics
- **Behavioural types**: graph types, secrecy & authenticity types (Cryptyc), CCS procs as types for pi procs (Behave)

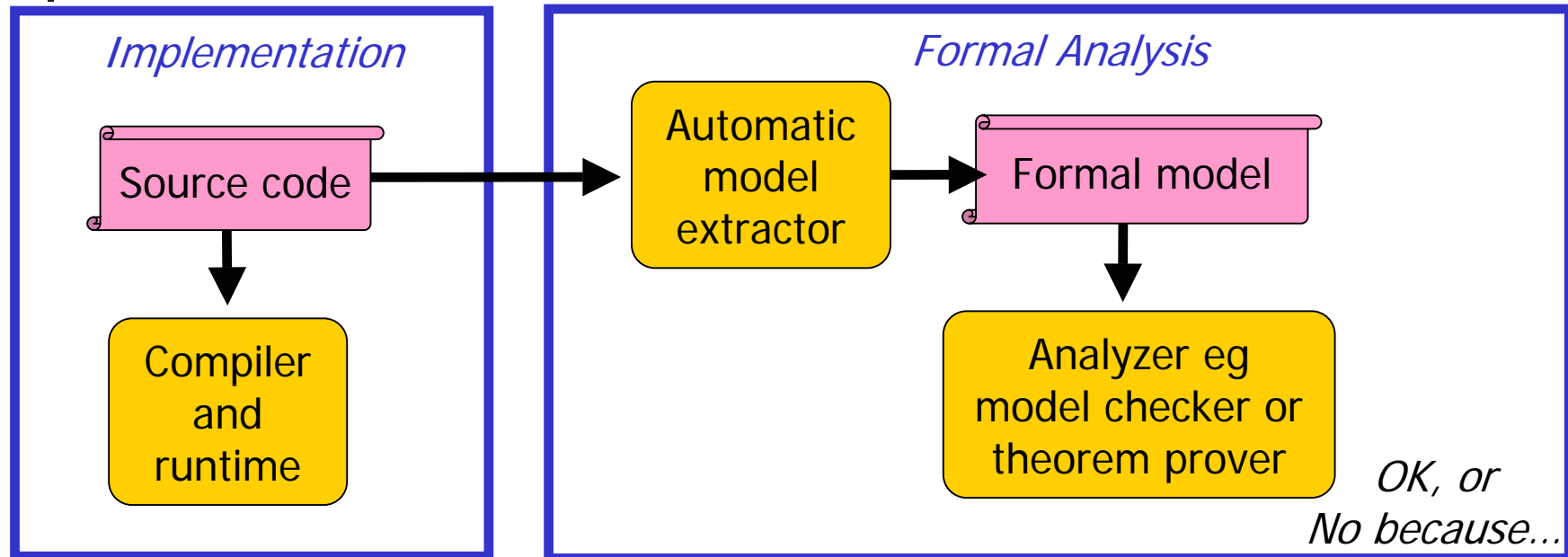
Next: a significant distinction concerning formal methods

Model-Based Formal Methods



- Modelling always abstracts “real world” detail, so may miss some bugs; still, effective when studying fixed, difficult algorithms
- Much worse, spec-based formal methods do not scale in practice; too costly to maintain two documents

Source-Based Formal Methods



- By extracting the model directly from the source code, formal tools remain applicable as design evolves
- "One document. One. It's the source code. You learn everything there and you know everything there."



A Turning Point for Pi?

- Some source-based methods:
 - Data types in compilers (60s)
 - Finite-state model checking hardware designs (early 90s)
 - Data flow analysis of software (Ariane, Prefix) (late 90s)
- In late 90s, mobility, the essence of pi, seemed a liability, as it prevents finite-state abstraction
- Due to increased use of pi as source, and various new extraction techniques, source-based FM for pi is growing and seems set to flourish, eg:
 - Model checking BPEL via SPIN (UCSB) and Zing (MSRR) (03)
 - Verifying WS-SecurityPolicy via TuleFale/ProVerif (MSRC) (04)
 - And behavioural typing of source code imminent (05???)