LANGUAGES, LOGICS, TYPES AND TOOLS FOR CONCURRENT SYSTEM MODELLING

Ramūnas Gutkovas

ramunas@fct.unl.pt

2016-12-14 NOVA LINCS


MYSELF
UPPSALA UNIVERSITET
2011 MSc
2016 PhD
Languages, Logics, Types and Tools for Concurrent System Modelling
RAMŪNAS GUTKOVAS
Uppsala, Sweden
ktu
1922
kauno technologijos universitetas
2007 BSc
Startup 2007-2009
Kaunas, Lithuania
www
HTML5
CSS3
PHP
FINLAND
Joensuu
Petrozavodsk
Trondheim
Bergen
Oslo
Tallinn
Baltic Sea
LATVIA
Kaunas
Kalingrad
Vilnius
Gdansk
Hrodna
DENMARK
Copenhagen
Aalborg
Kiel
Hamburg
NETHERLANDS
Münster
Bielefeld
Braunschweig
Warsaw
POLAND
Homyel'
Bryansk
Vyshniy Volo
Smolensk

BUGS

OR, WHEN MACHINES GO WRONG

Windows

An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this,
you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

Press any key to continue _

:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED

EXPENSIVE BUGS

Pentium FDIV bug, 1994

$475 million worth of recalls

Ariane 5 went "poof", 1996
Integer Overflow

$ 500 million loss

goo.gl/bU36B2

KILLER BUGS

Therac-25, 1980s

Due to a race condition produced a lethal radiation burst

5 killed

Toyota, 2010

Unintended acceleration
Software bug

89 killed

Many more:
goo.gl/GVQIIC

Microsoft calls them 1 million dollar bugs!

SECURITY

Heartbleed

OpenSSL

A bug that allows to obtain keys
Most of the internet affected

SSL is foundation for ecommerce.

CVE-2016-5195

DIRTY COW

Data race in the linux kernel since 2007 allows to escalate privileges

Millions of Android devices vulnerable

UNPRECEDENTED IMPLICATIONS

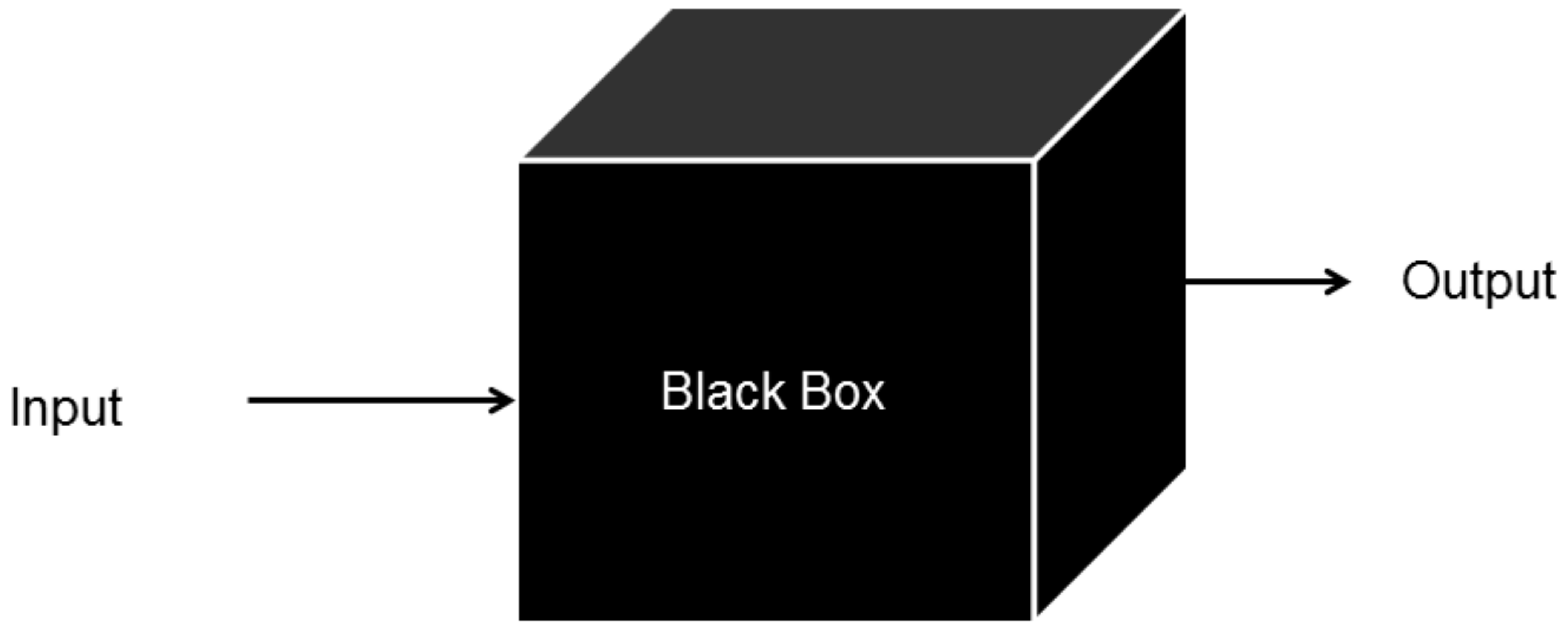
Celebgate

Celebrity Apple's iCloud accounts hacked

Influence foreign government elections

Voldemord

STANDARD APPROACH: TESTING



Internal behavior of the code is unknown

TESTING CAN'T BE COMPLETE

Testing is **essential**, however, it is not **sufficient!**

Suppose `int` is 32 bits

```
int multiply (int x, int y)
```

Thus, there are 2^{64} inputs

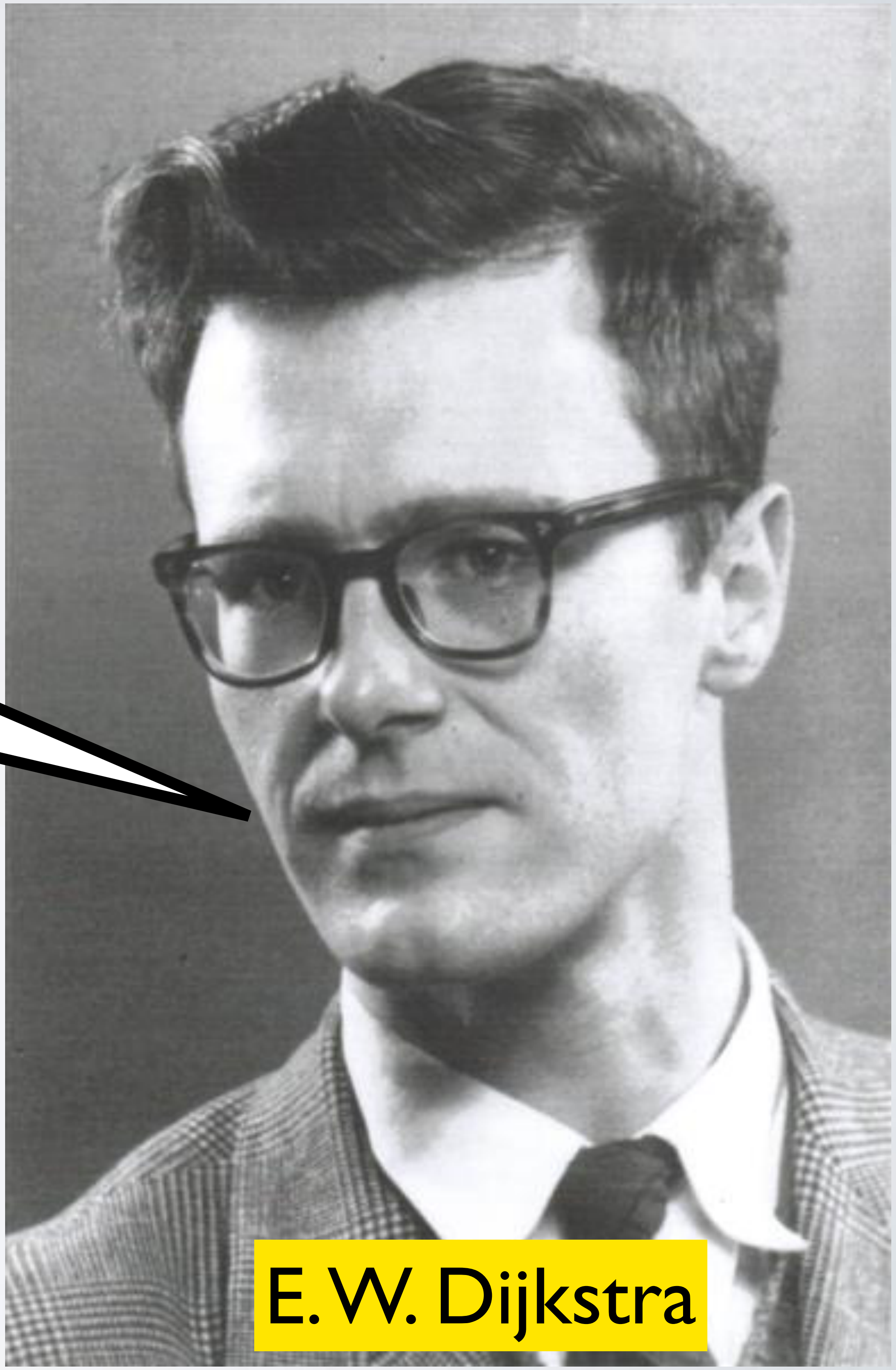
Intel Core i7 5960X (8 core)

can do about 2^{38} instructions per second

It would take 2^{26} sec ~ 18641 hours ~ **2 years** to test

TESTING

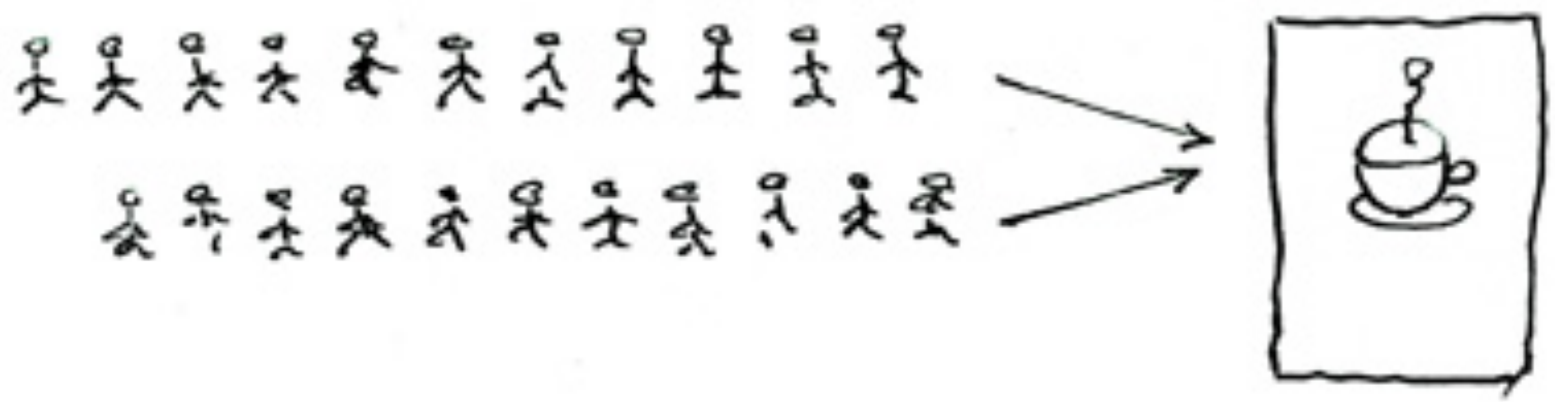
…program testing can be used to show the presence of bugs, but never to show their absence!



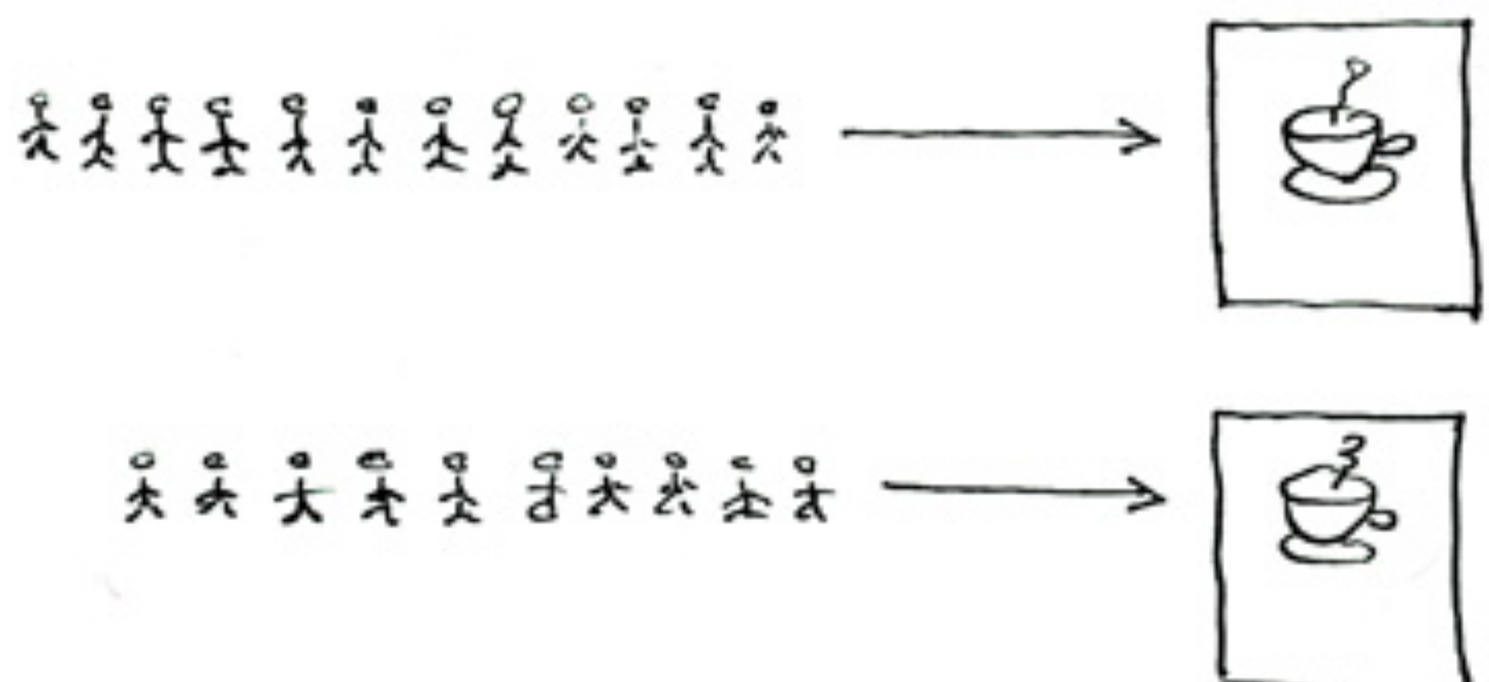
[EWD303]

CONCURRENT SYSTEMS

Concurrent = Two Queues One Coffee Machine



Parallel = Two Queues Two Coffee Machines



THE CHALLENGE

find

An adequate language for describing concurrent systems

A mathematical theory for capturing dynamics, i.e. semantics

Well-founded Verification Technique

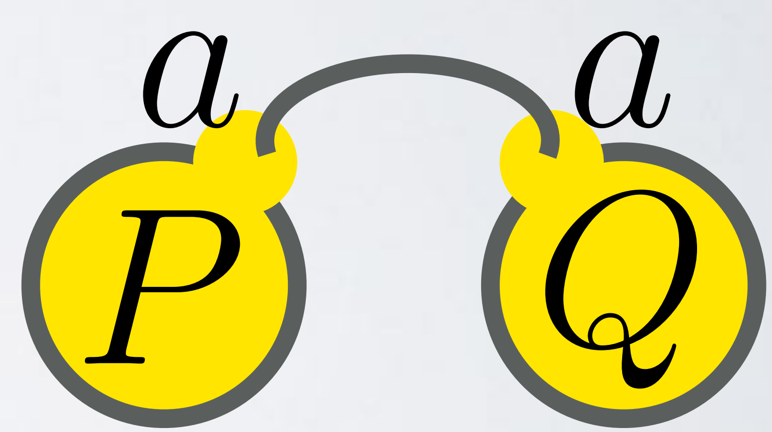
CALCULUS OF COMMUNICATING SYSTEMS

[Milner 1980]

$$
\begin{array}{rcll}
a & \in & \mathscr{A} & \text{action} \\
\\
P, Q & ::= & a.P & \text{input} \\
 & \mid & \overline{a}.P & \text{output} \\
 & \mid & \mathbf{0} & \text{inaction} \\
 & \mid & \tau.P & \text{silent} \\
 & \mid & P \mid Q & \text{parallel} \\
 & \mid & P + Q & \text{sum/choice} \\
 & \mid & !P & \text{replication}
\end{array}
$$

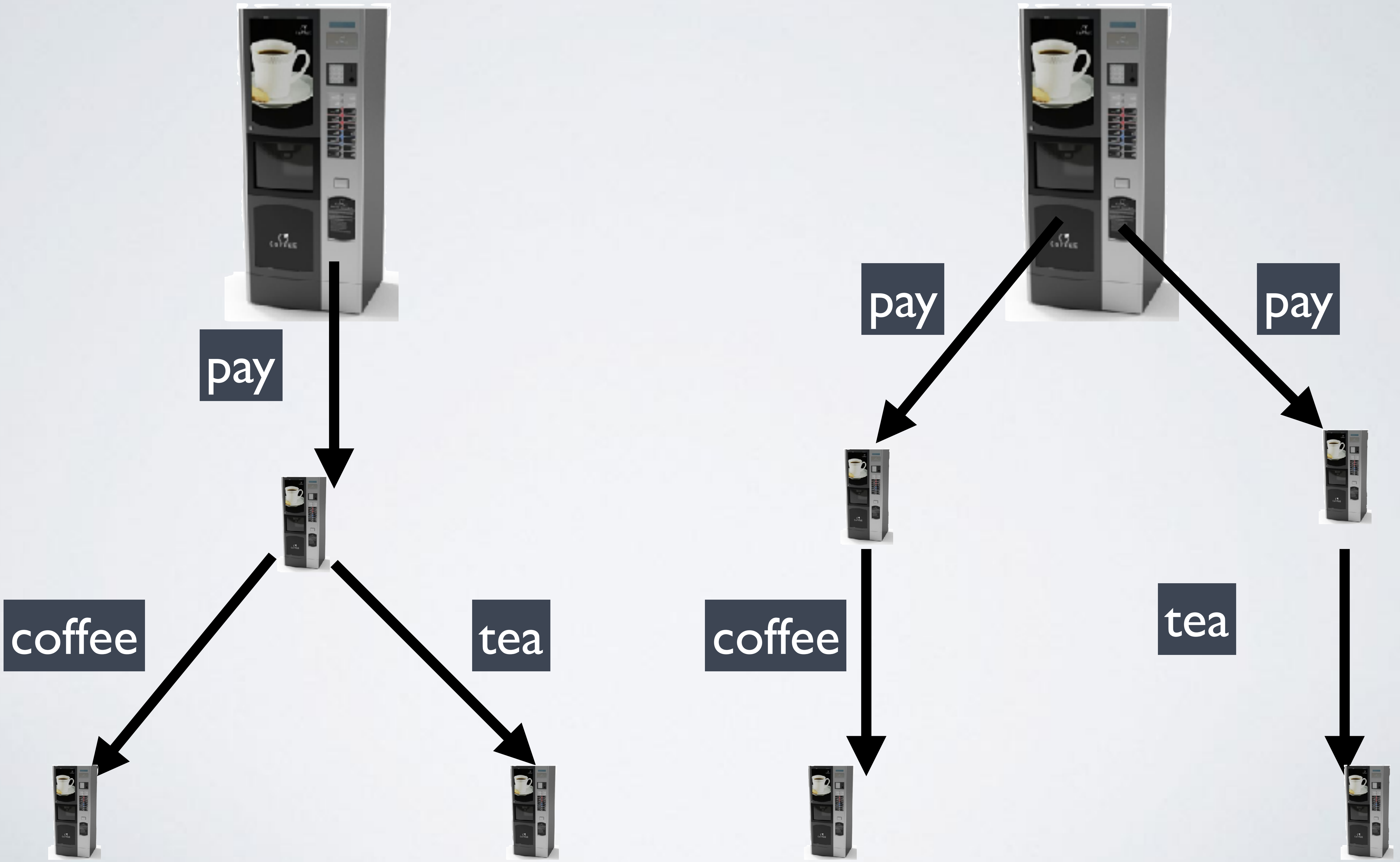
Ex.

$$a.P \mid \overline{a}.Q$$

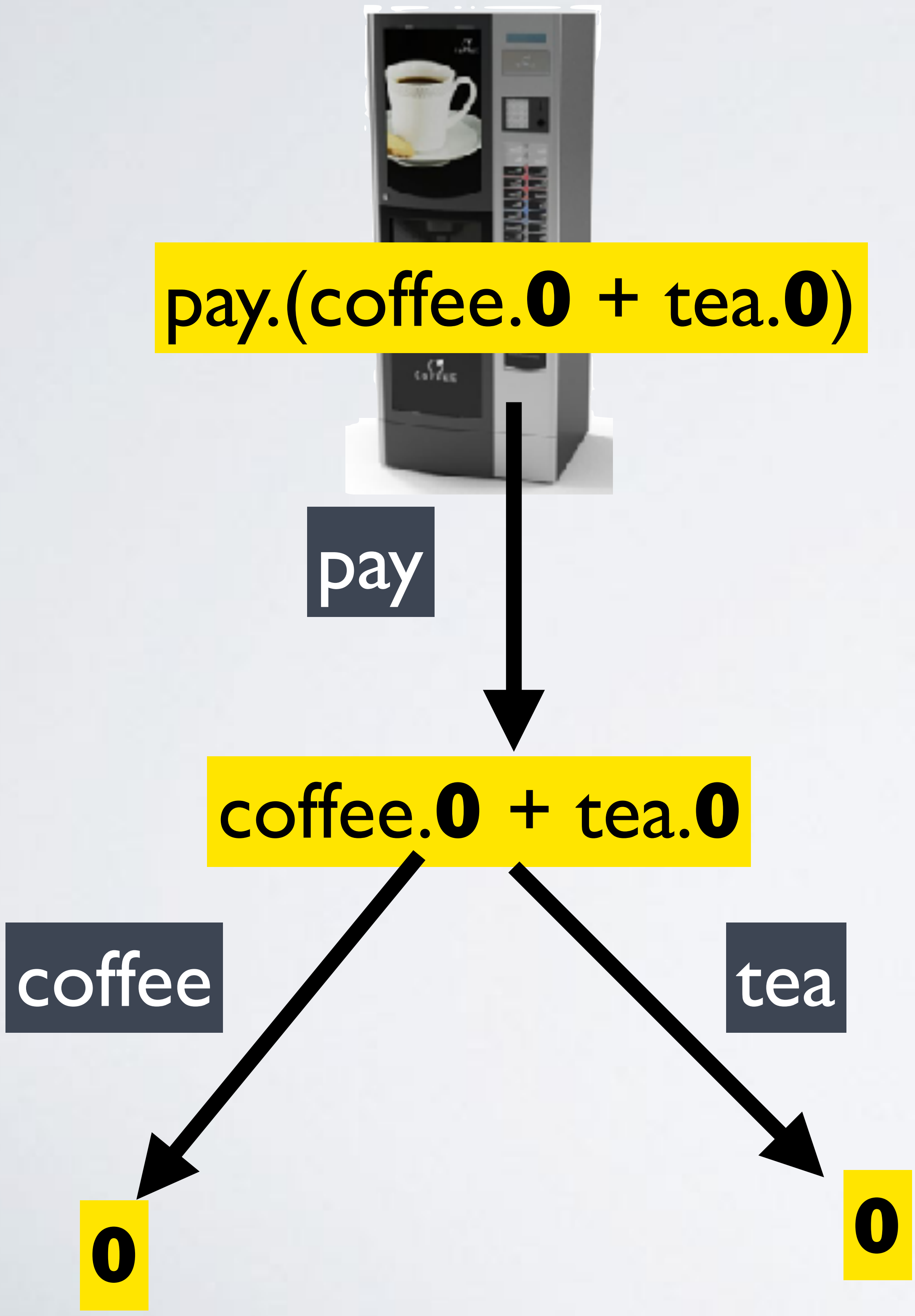


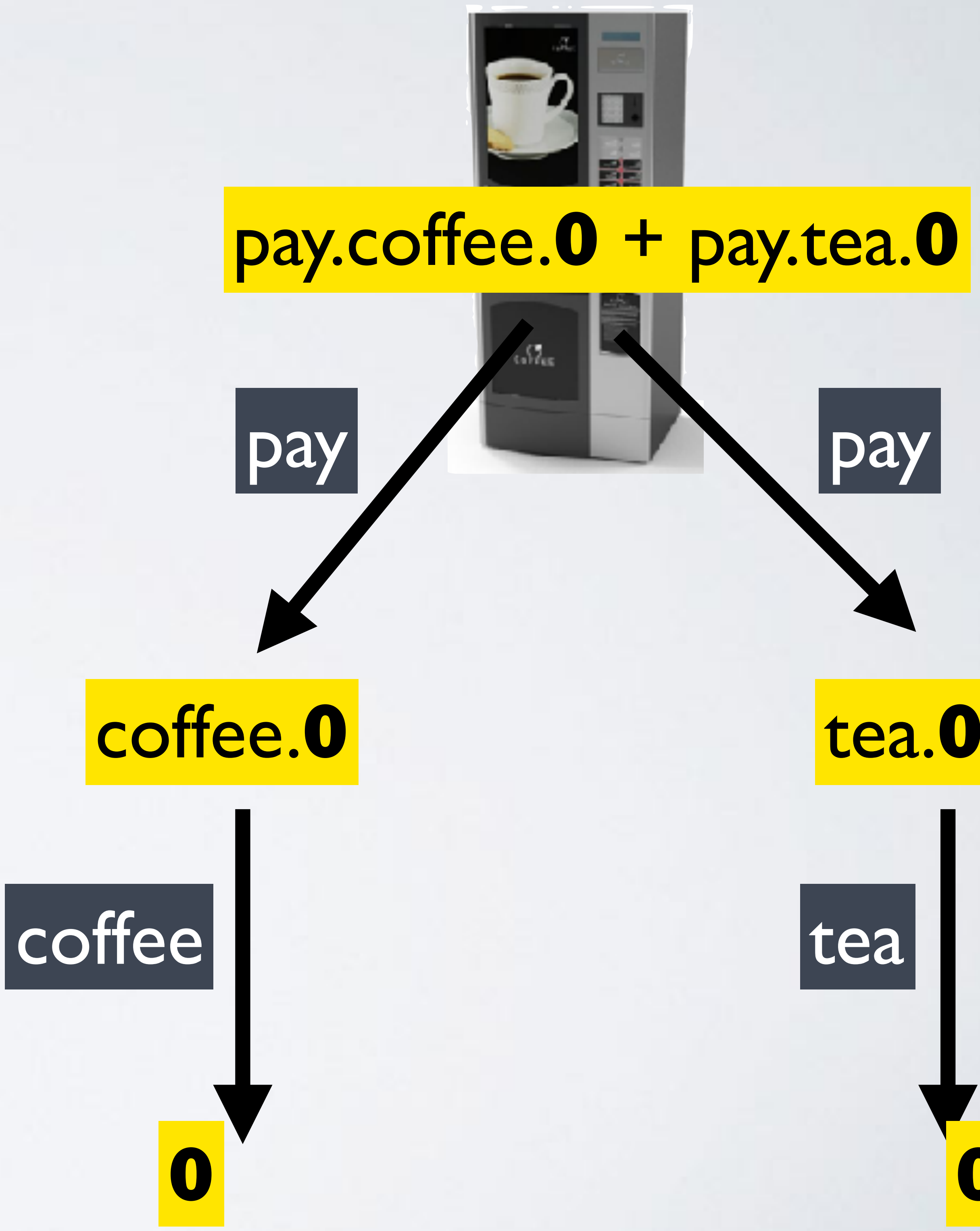
(can be extend with value passing)

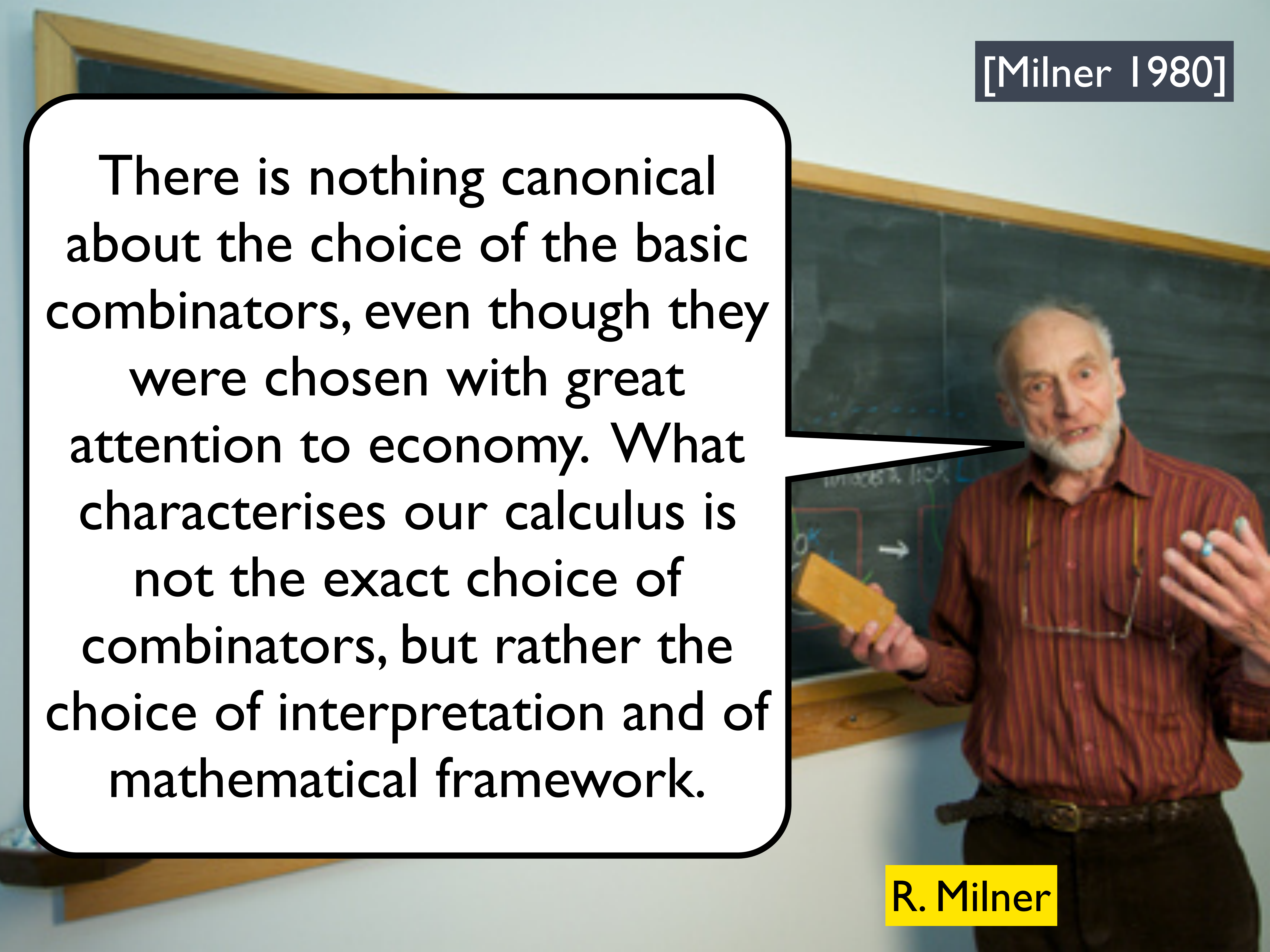
OBSERVABLE BEHAVIOUR



OBSERVABLE BEHAVIOUR






[Milner 1980]
There is nothing canonical about the choice of the basic combinators, even though they were chosen with great attention to economy. What characterises our calculus is not the exact choice of combinators, but rather the choice of interpretation and of mathematical framework.
R. Milner

ALGEBRA OF PROCESSES

Equivalence based on the observable behaviour

Bisimilarity

$P \sim Q$ at each state P can perform all the actions of Q, and vice versa, and states continue to be bisimilar

Alg. properties

$P|Q \sim Q|P$ $\quad P|(Q|R) \sim (P|Q)|R$

$\mathbf{0}|P \sim P$ etc.

$\mathbf{0} + P \sim P$

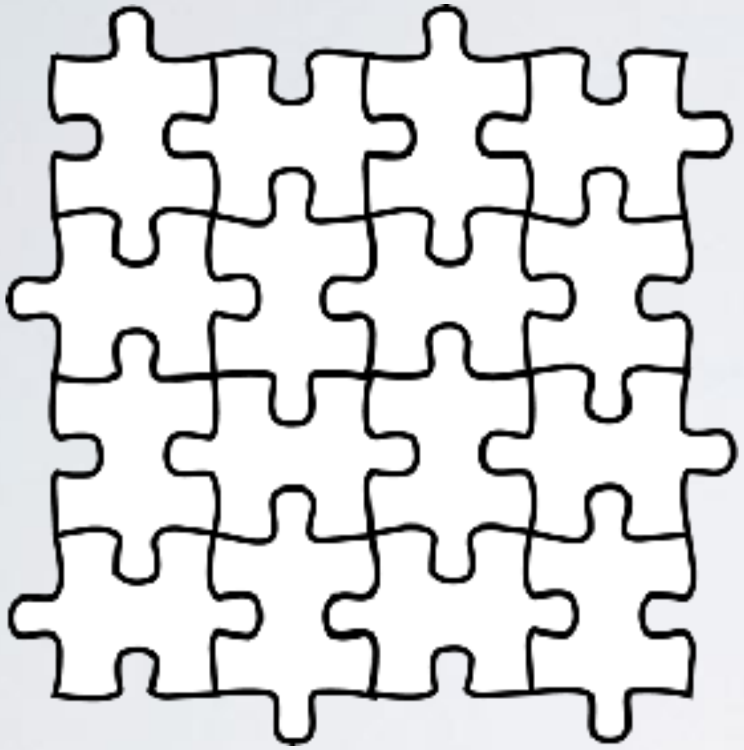
$P + Q \sim Q + P$

Weak bisimilarity roughly, ignoring silent actions

$P \approx \tau.P$

COMPOSITIONALITY

(Frege's principle)



Systems built from smaller systems

Component Modularity

Under all contexts a processes behaviour is indistinguishable (ie. bisimilar)

A congruence relation

Equivalence (bisimulation) preserved under all operations

Ex.

if $P \dot{\sim} Q$

then $R|P \dot{\sim} R|Q$

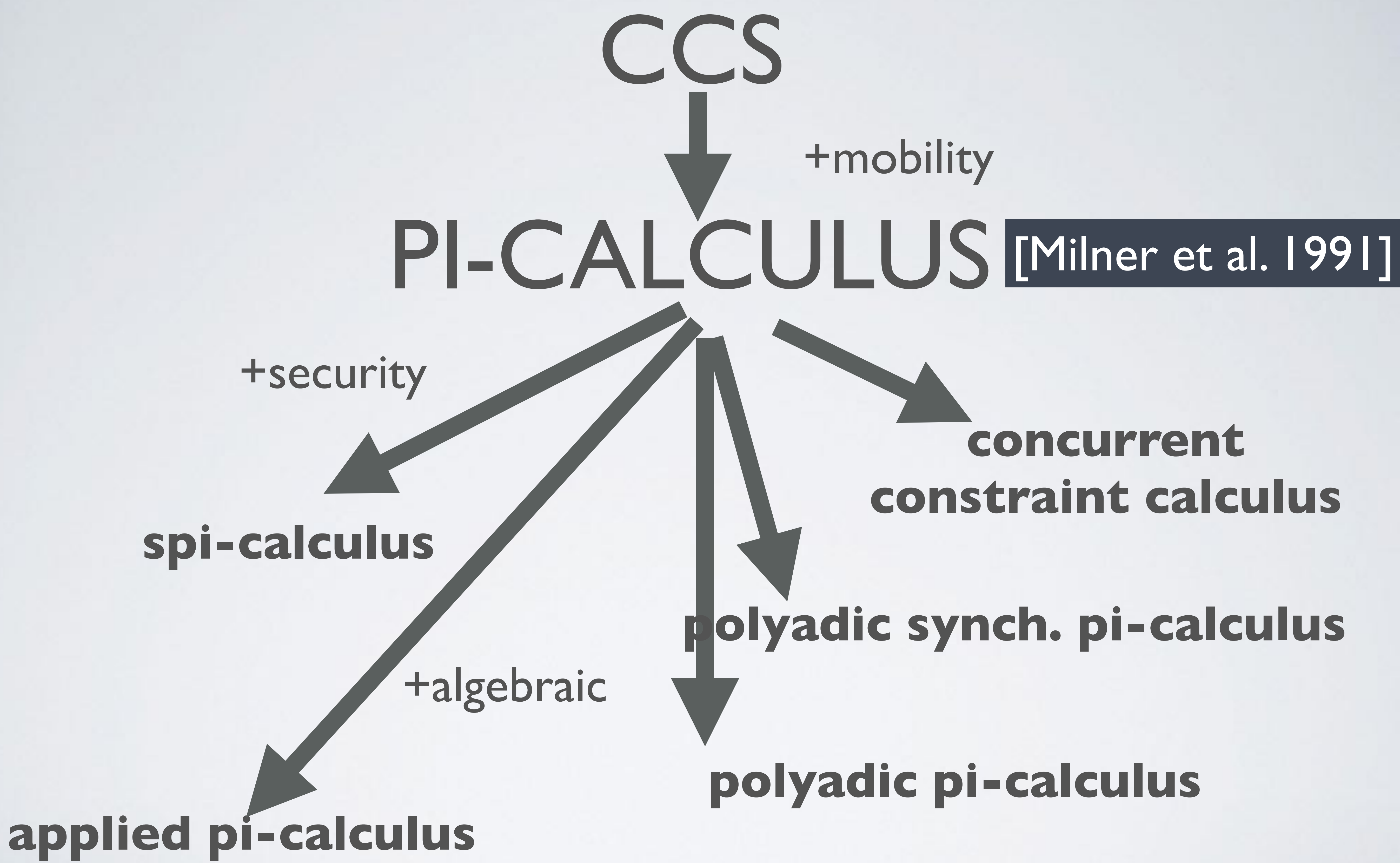
VERIFICATION TECHNIQUE

Specification $\approx$ Implementation

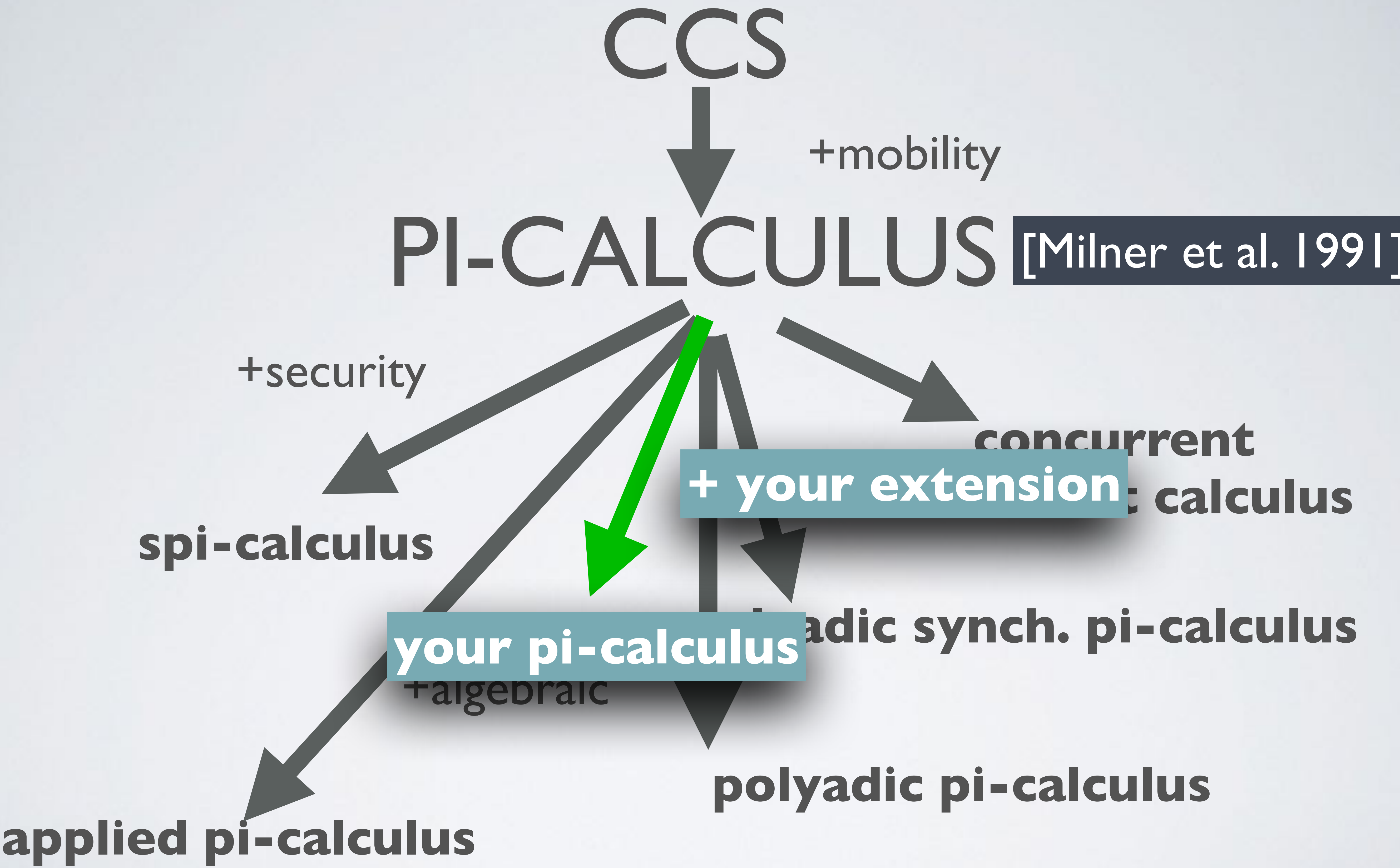
(weakly) bisimilar

$$\mathrm{Specification} = \mathsf{pay}.(\mathsf{coffee}.\mathbf{0} + \mathsf{tea}.\mathbf{0})$$

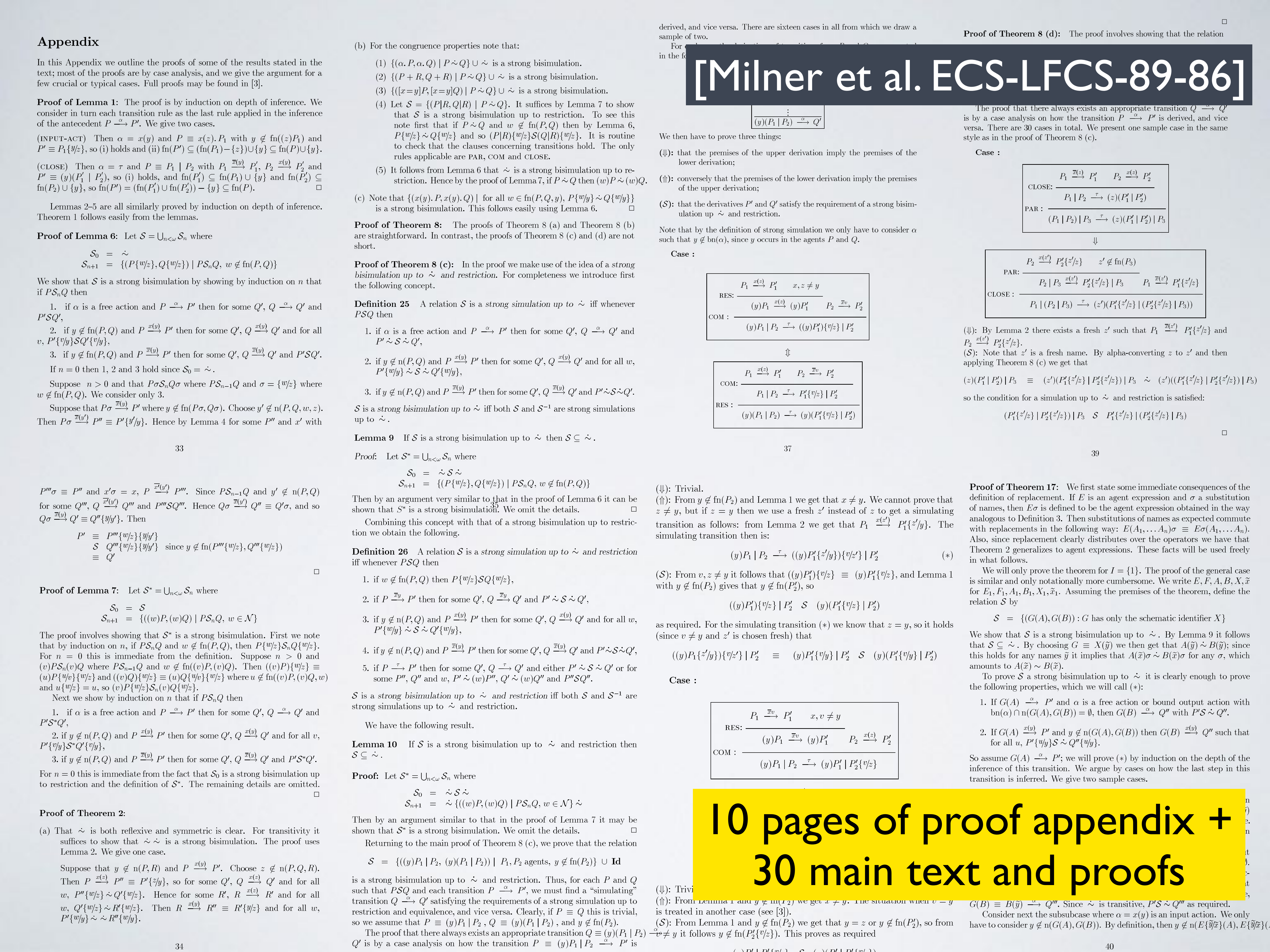
$$\begin{aligned}
&\mathrm{Implementation} = \\
&\quad \mathsf{pay}.(\nu \mathrm{interal})(\mathrm{internal}(\mathsf{amount}). \\
&\qquad \mathbf{if}\ \mathsf{amount} = 50 \\
&\qquad\quad \mathbf{then}\ \mathsf{coffee}.\mathbf{0} + \mathsf{tea}.\mathbf{0} \\
&\qquad\quad \mathbf{else}\ \mathsf{coffee}.\mathbf{0} + \mathsf{tea}.\mathbf{0} \\
&\qquad \mid \quad P)
\end{aligned}$$



… and myriad of other 'small' extensions of pi



… and myriad of other ‘small’ extensions of pi



[Milner et al. ECS-LFCS-89-86]

10 pages of proof appendix + 30 main text and proofs

(⇓): Trivial.
(⇑): From $y \notin \mathrm{fn}(P_2)$ and Lemma 1 we get that $x \neq y$. We cannot prove that $z \neq y$, but if $z = y$ then we use a fresh z' instead of z to get a simulating transition as follows: from Lemma 2 we get that $P_1 \xrightarrow{x(z')} P_1'\{z'/y\}$. The simulating transition then is:

$$(y)P_1 \mid P_2 \xrightarrow{\tau} ((y)P_1'\{z'/y\})\{v/z'\} \mid P_2' \qquad (*)$$

($\mathcal{S}$): From $v, z \neq y$ it follows that $((y)P_1')\{v/z\} \equiv (y)P_1'\{v/z\}$, and Lemma 1 with $y \notin \mathrm{fn}(P_2)$ gives that $y \notin \mathrm{fn}(P_2')$, so

$$((y)P_1')\{v/$$

as required. For the simulati
(since $v \neq y$ and z' is chosen

$$((y)P_1\{z'/y\})\{v/z'\} \mid P_2'$$

Case :

P_1
RES:
COM :

With cheats!

(⇓): Trivial.
(⇑): From $y \notin \mathrm{fn}(P_2)$ and Lemma 1 we g
$z \neq y$ but if $z = y$ then we use a fresh

In this Appendix we outline the proofs of some of the results stated in the text; most of the proofs are by case analysis, and we give the argument for a few crucial or typical cases. Full proofs may be found in [3].

(⇓): Trivial.
(⇑): From Lem
is treated in another case (see [3]).
($\mathcal{S}$): From Lemma 1 and $y \notin \mathrm{fn}(P_2)$ we get that $y = z$ or $y \notin \mathrm{fn}(P_2')$, so from $v \neq y$ it follows $y \notin \mathrm{fn}(P_2'\{v/z\})$. This proves as required

$$(y)P_1' \mid P_2'\{v/z\} \quad \mathcal{S} \quad (y)(P_1' \mid P_2'\{v/z\})$$

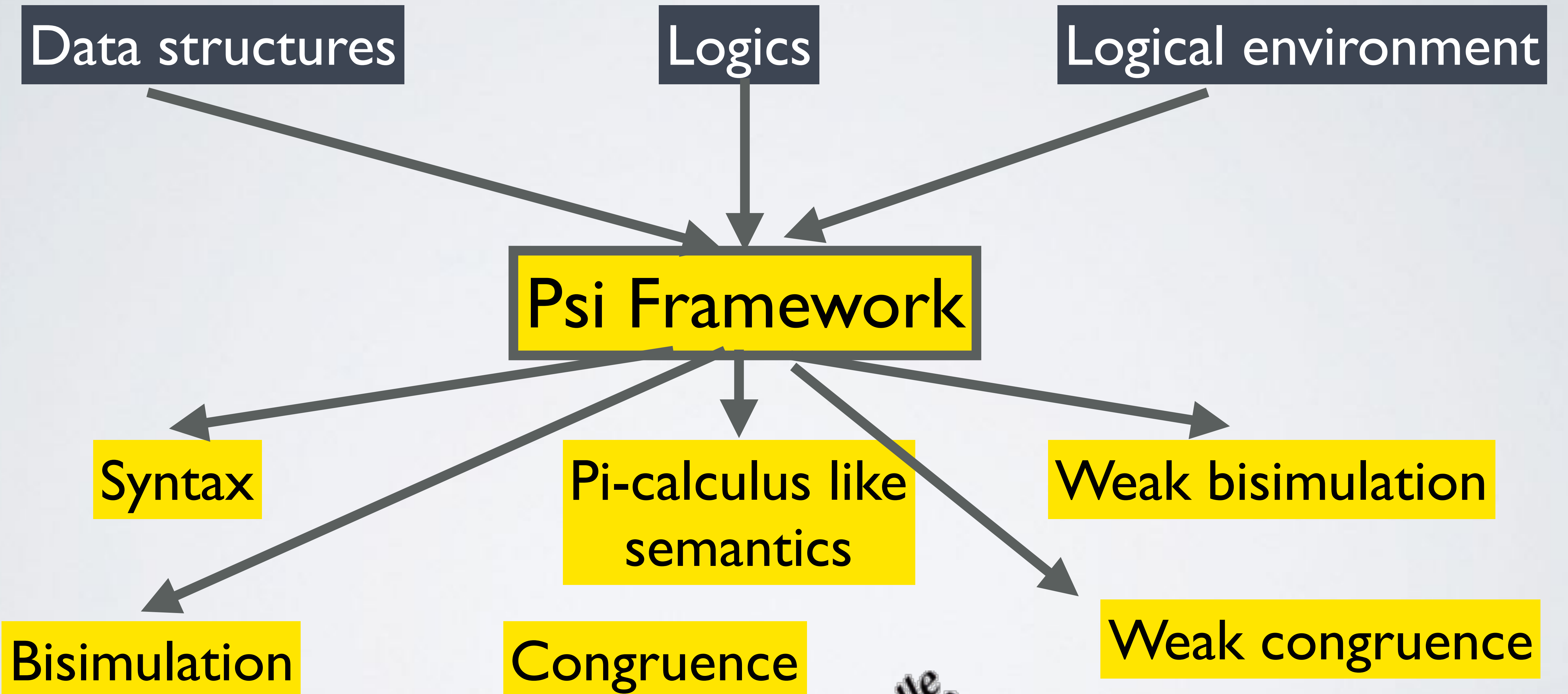
38

[3] is self-reference.

For examples of bugs in meta-theories see [Bengtson et al. 2011]

[Bengtson et al. 2011]

PSI-CALCULUS FRAMEWORK



Bengtson and Pohjola

Parameters

x	$\in$	$\mathcal{N}$	name
M, N	$\in$	$\mathbf{T}$	term
$\varphi_1, \ldots, \varphi_n$	$\in$	$\mathbf{C}$	condition
Ψ	$\in$	$\mathbf{A}$	assertion
$\dot{\leftrightarrow}$	$\in$	$\mathbf{T} \times \mathbf{T} \to \mathbf{C}$	channel equivalence
$\otimes$	$\in$	$\mathbf{A} \times \mathbf{A} \to \mathbf{A}$	assertion composition
$\vdash$	$\subseteq$	$\mathbf{A} \times \mathbf{C}$	entailment
$[\widetilde{x} := \widetilde{N}]$	$\in$	$\mathbf{T} \to \mathbf{T}$	substitution function

Syntax

P, Q	$::=$	$M(\lambda\widetilde{x})N.P$	input
	$\mid$	$\overline{M}\,N.P$	output
	$\mid$	$P \mid Q$	parallel
	$\mid$	$!P$	replication
	$\mid$	$(\nu x)P$	restriction
	$\mid$	$(\!\mid \Psi \mid\!)$	assertion
	$\mid$	$\mathbf{case}\ \varphi_1 : P_1 \,[\,]\, \cdots \,[\,]\, \varphi_n : P_n$	case
	$\mid$	$\mathbf{0}$	inaction

MAC

$$
\begin{array}{l}
(\nu\,\mathrm{secret})(\\
\qquad (\!|\mathsf{hash}(\langle \mathrm{secret}, \mathrm{message}\rangle) = x|\!) \mid \\
\qquad \overline{a}\langle \mathrm{message}, x\rangle \mid \\
\qquad a(y). \\
\qquad\qquad \textbf{case } \mathsf{hash}(\langle \mathrm{secret}, \mathsf{fst}(y)\rangle) = \mathsf{snd}(y) \quad : \overline{b}\,\mathrm{YES} \\
\qquad\qquad\quad [\!]\,\mathsf{hash}(\langle \mathrm{secret}, \mathsf{fst}(y)\rangle) \neq \mathsf{snd}(y) \quad : \overline{b}\,\mathrm{NO} \\
)
\end{array}
$$

generate a key

sign a message

send MAC

receive MAC

Verify

UPPSALA
UNIVERSITET

Languages, Logics, Types and Tools for Concurrent System Modelling

RAMŪNAS GUTKOVAS

Expanding generality of Psi-calculi with a type-system

Providing a verification calculus for psi-calculus, and others

Tool support for psi-calculi

SORT SYSTEM FOR PSI

[LMCS 2016]

REPRESENTATION

A direct encoding of a process calculus to a Psi-calculus

No elaborate encodings

No superfluous data terms

No superfluous behaviour

Many calculi were not representable

Unsorted polyadic pi-calculus   Sorted polyadic pi-calculus

LINDA pattern matching   Polyadic synchronisation pi-calculus

Value-passing CCS

Goal: extend psi-calculi to be capable of representing new calculi!

SYMMETRIC CRYPTO

Computation

$$\mathtt{dec}(\mathtt{enc}(M, K), K) \rightarrow M$$

makes sense when it is typed

$$(\nu a, k)(\overline{a}\,\text{“foobar”}.\mathbf{0} \quad | \quad \underline{a}(\lambda y)y.\,\overline{c}\,\mathsf{dec}(y, k).\mathbf{0})$$

$$\xrightarrow{\tau} \qquad (\nu a, k)(\mathbf{0} \quad | \quad \overline{c}\,\mathsf{dec}(\text{“foobar”}, k).\mathbf{0})$$

$$(\nu a, k)(\overline{a}\,\mathsf{enc}(M, k).\mathbf{0} \quad | \quad \underline{a}(\lambda y)y.\,\overline{c}\,\mathsf{dec}(y, k).\mathbf{0})$$

$$\xrightarrow{\tau} \qquad (\nu a, k)(\mathbf{0} \mid \overline{c}M)$$

SORT SYSTEM

Set of sorts $\mathcal{S}$

Sort assigning to params function $\textsc{Sort}(X) \in \mathcal{S}$

Sorting relations for substitution and processes:

can send **can receive** **can restrict**

can substituted

Consider only well sorted substitutions

Sanity check: A well-sorted substitution preserves well-sortedness of a process.

RESULTS

All the standard algebraic laws of bisimulation are preserved

Weak bisimulation   Weak congruence   Bisimulation

Congruence

All the mentioned calculi are **directly representable**

Unsorted polyadic pi-calculus   LINDA pattern matching

Sorted polyadic pi-calculus   Polyadic synchronisation pi-calculus

Value-passing CCS

MODAL LOGICS FOR PSI

MODAL LOGICS

Find grained properties of a system

== Process

Deadlock freedom

Eventually coffee machine produces **coffee**

A malicious message is eventually rejected

== Modal Logic Formula

Process

is a model

Modal logical formula

$$P \models \varphi$$

formula φ is true for P

MODAL LOGICS

Concurrent System Models	Logics
CCS	Hennessy, Milner 1985
Value-Passing CCS	Hennessy, Liu 1995
Spi-calculus	Frendrup et al.2002
Applied pi-calculus	Hüttel, Pedersen et al. 2007
Fusion calculus	Haugstad et al. 2006
Multi-labelled Nominal transition systems	De Nicola, Loreti 2008
Psi-calculi framework	???
Concurrent constraint calc.	???

Possibly others

[CONCUR 2015]

NOMINAL MODAL LOGIC

Formulas depend on finite number of names

$$P \vDash \varphi \quad \textbf{iff} \quad P \vdash \varphi$$

$$P \vDash \neg A \quad \textbf{iff} \quad \textbf{not } P \vDash A$$

$$P \vDash \bigwedge_{i \in I} A_i \quad \textbf{iff} \quad (\forall i \in I) \quad P \vDash A_i$$

$$P \vDash \langle \alpha \rangle A \quad \textbf{iff} \quad (\exists P') \quad P \xrightarrow{\alpha} P', \; P' \vDash A$$

Thm. Adequate for strong bisimilarity.

What's new: finitely supported formulas

EXPRESSIVENESS

Next step	for any action there is a state
Quantifiers	for every value of a domain
Fresh/New	for a state where a name does not appear

Recursion in Logic $\mathrm{rec}\, X.A$

Ex.

Eventually get **coffee** :=
rec X. <coffee>true ∨ next step, recurse on X

RESULTS

Adequate Modal Logic for many transition systems

The main proofs are machine checked

Adequate for many variants of bisimilarity:
hyper, open, early, late, weak

Provide an adequate modal logic for

psi-calculi, concurrent constraint calculus, and others

TOOL SUPPORT

AUTOMATED TOOLS

Small specification:

WSN secure aggr.

Small spec. in Pwb

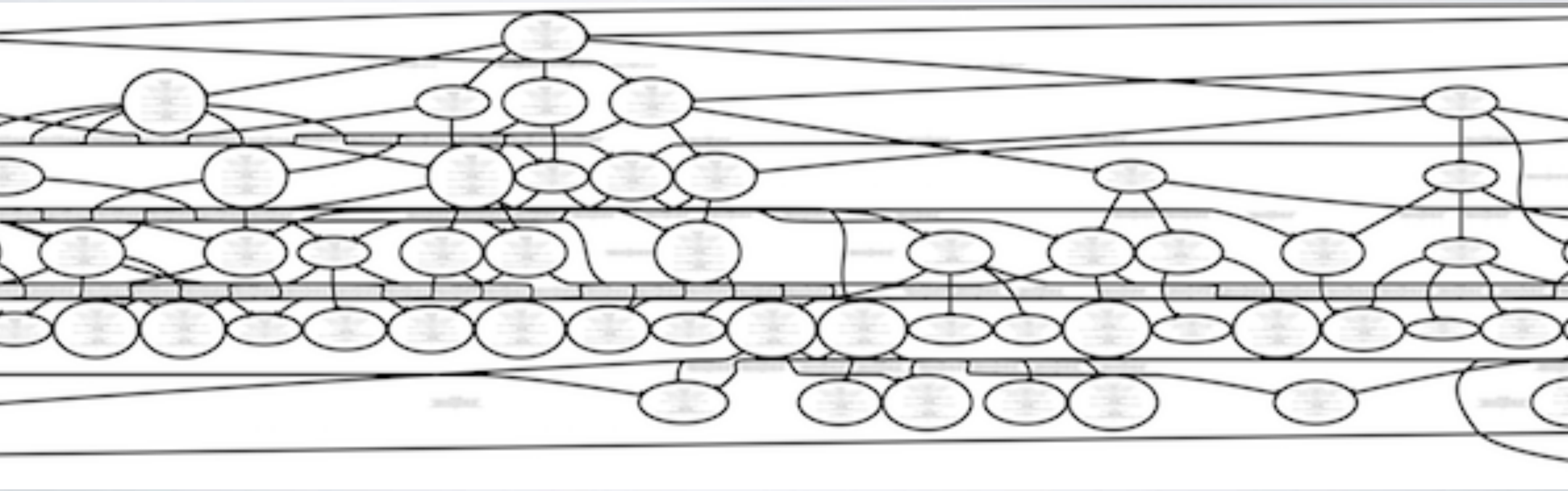
20 LOC

only 3 nodes

Property

There is no tempered data that the network accepts

Results in



[TECS 2015]

PSI-CALCULI WORKBENCH

Tool factory: define your own tool!

Based on the parametric psi-calculi framework

PARAMETRIC

Data Structures

e.g., Names, Bits, Vectors, ADTs, Trees, ...

Logics

e.g., EUF, FOL, Equational Theory, ...

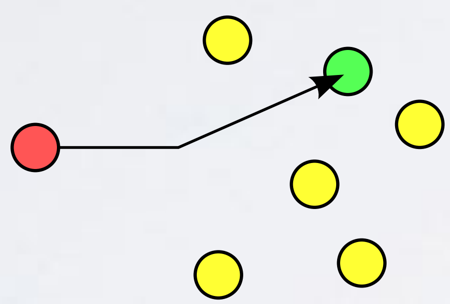
Logical Assertions

e.g., Knows a secret, Connectivity, …

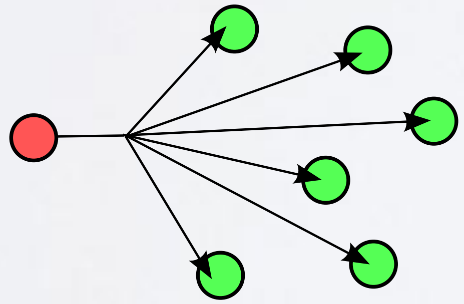
FEATURES

Communication Primitives

Unicast



Unreliable Broadcast



[Borgström et al. 2011]

Execution of Processes

(Weak) Bisimulation Checking

Pluggable Architecture

EXAMPLE: WSN AGGREGATION

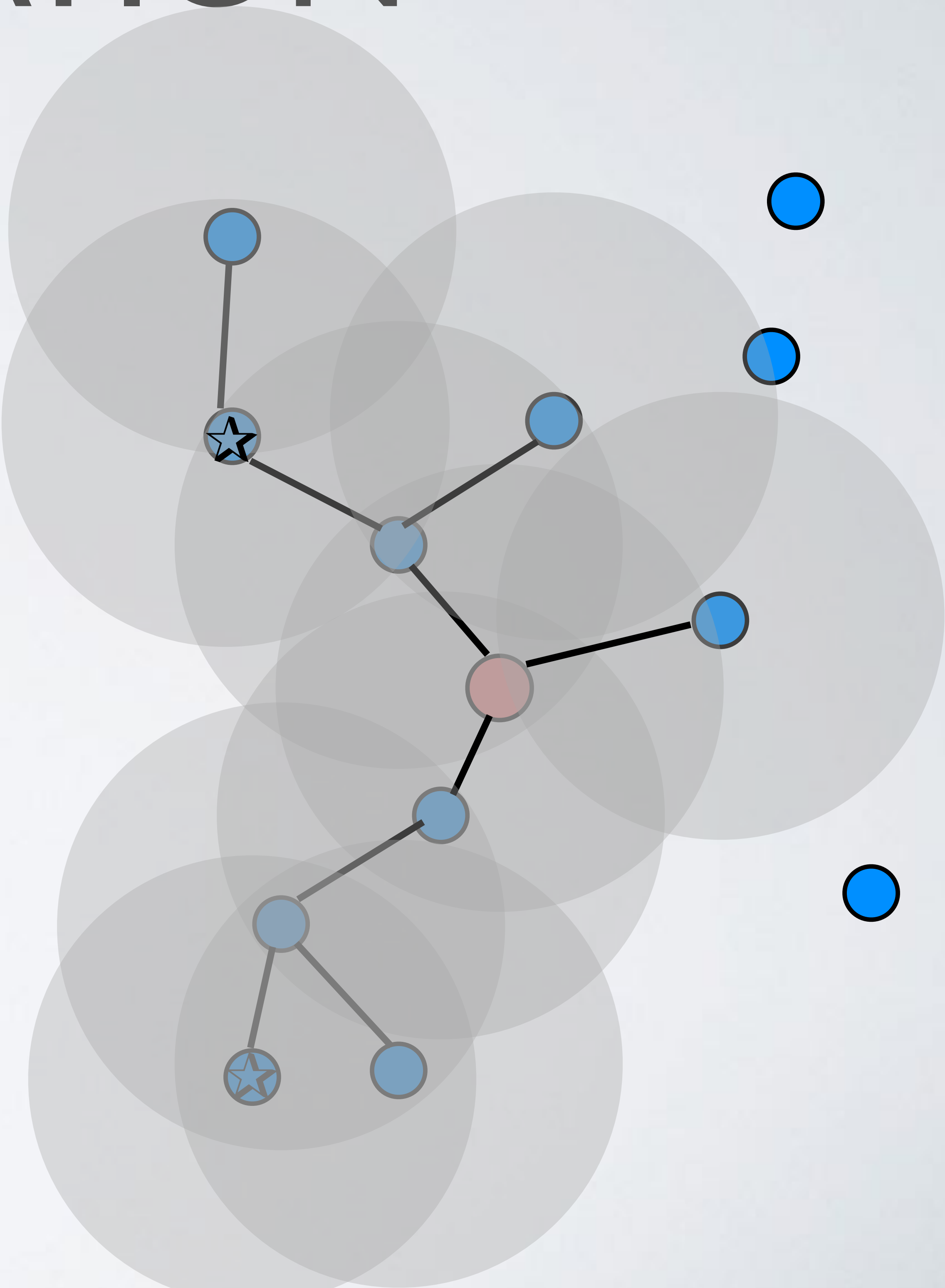
Spatially distr. nodes

Wireless communication

Protocol:

Establish routing tree

Forward data



WORKBENCH MODEL

```
Sink(nodeId, bsChan) <=
    ’”init(nodeId)”!<bsChan> .
    ! ”data(bsChan)”(x) ;

Node(nodeId, nodeChan, datum) <=
    ”init(nodeId)”?(pChan) .
    ’”init(nodeId)”!<nodeChan> .
    ’”data(pChan)”<datum> .
    NodeForwardData<nodeChan, pChan> ;

NodeForwardData(nodeChan, pChan) <=
    ! ”data(nodeChan)”(x). ’”data(pChan)”<x> ;
```

SYMBOLIC EXECUTION

generated action

```
--|gna!(new bsChan)bsChan|-->
Source:
  System3<d1, d2>
Constraint:
  (new chan1, chan2, chanS){| "init(0)<gna" |} ∧
  (new chanS, chan2, chan1){| "gna>init(1)" |} ∧
  (new chanS, chan1, chan2){| "gna>init(2)" |}
Solution:
  ([gna := "init(0)"], 1)
Derivative:
  (!("data(chanS)"(x))) |
    (((new chan1)(
      '"init(1)"!<chan1>.
        '"data(chanS)"<d1>.
          NodeForwardData<chan1, chanS>
    )) |
      ((new chan2)(
        '"init(2)"!<chan2>.
          '"data(chanS)"<d2>.
            NodeForwardData<chan2, chanS>
      )))
```

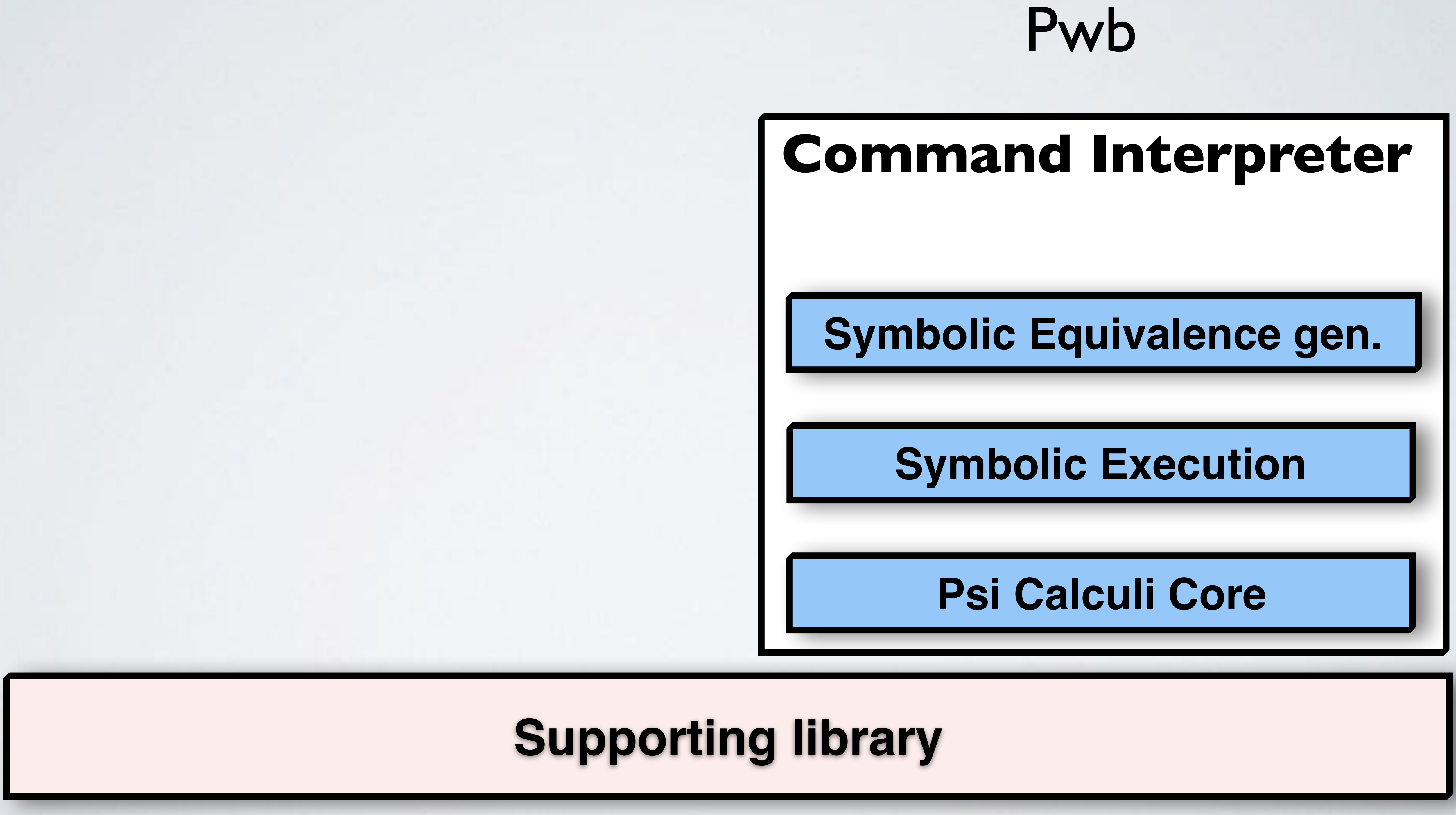
system with 3 nodes

constraints

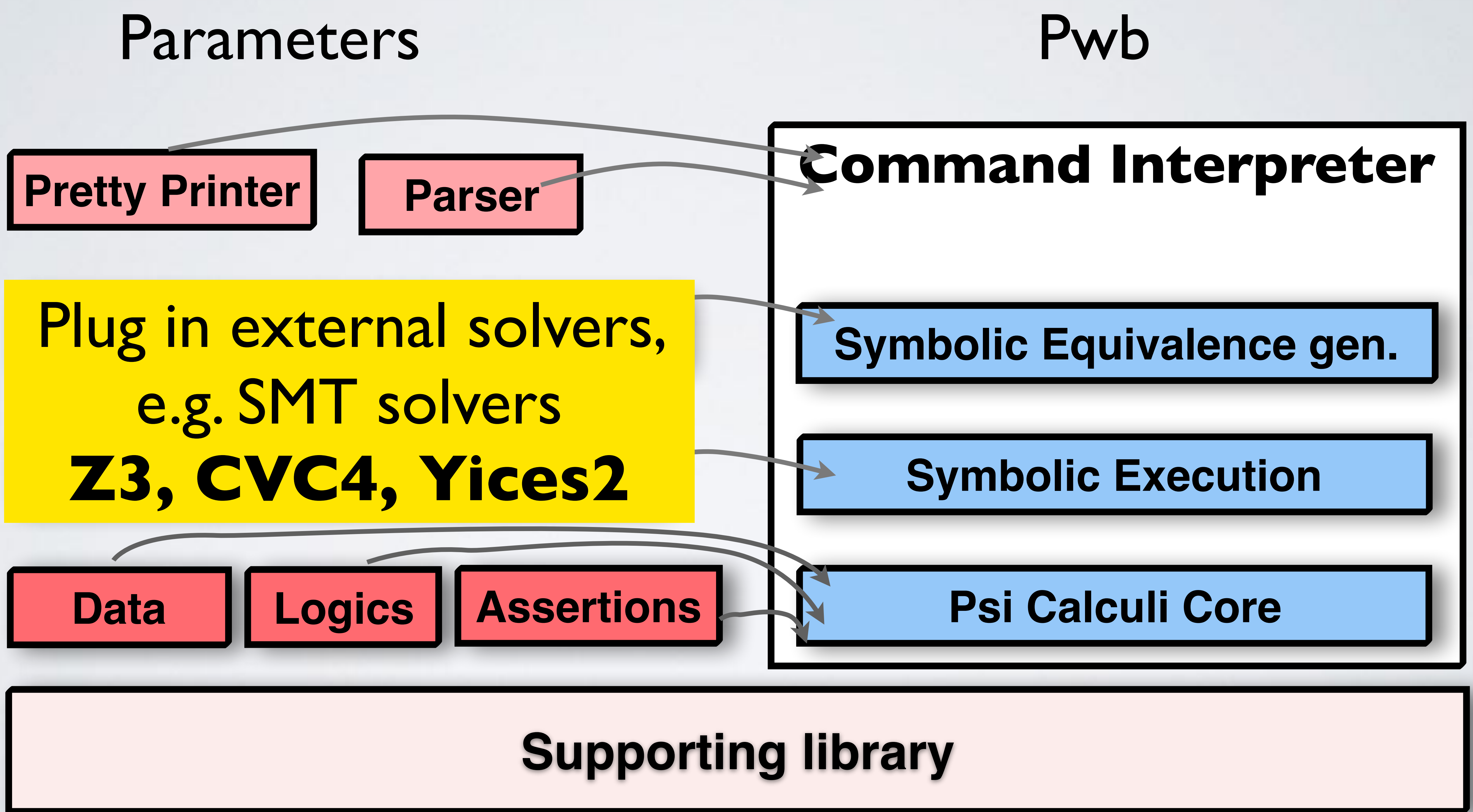
solution

Execution: derived process

ARCHITECTURE



ARCHITECTURE



CONCLUSION

A widened applicability of psi-calculi via a type system

A general and powerful modal logic that is applicable to systems such as psi-calculi

Tool support for psi

QUESTIONS