Searching for middle ground with PEPA

Jane Hillston
Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland

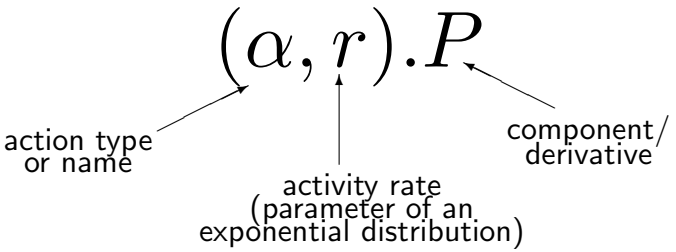
29th June 2003

Introduction

- The PEPA formalism
- Integration via tools
  - Solvers
  - Specification
- Integration at the formalism level
- Outlook and on-going work

Performance Evaluation Process Algebra (PEPA)

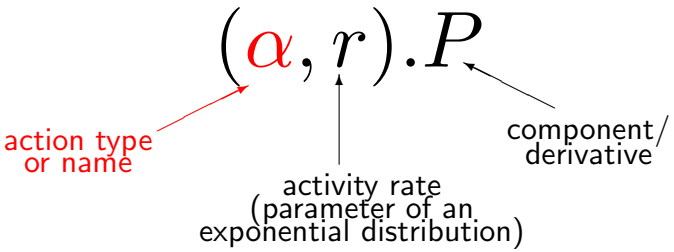
- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

Performance Evaluation Process Algebra (PEPA)

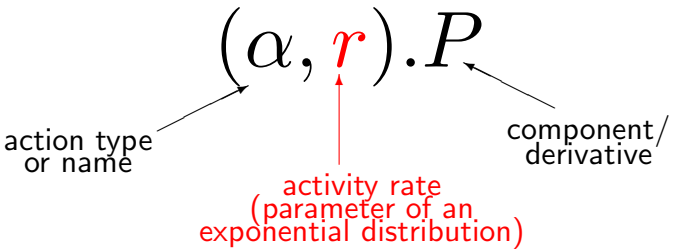
- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

Performance Evaluation Process Algebra (PEPA)

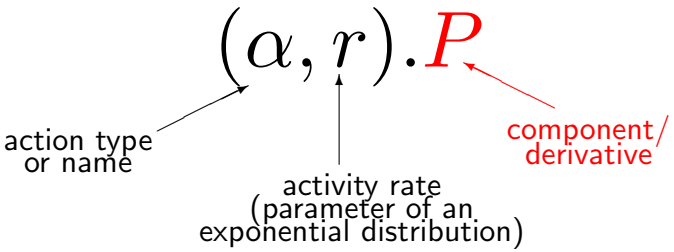
- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

Performance Evaluation Process Algebra (PEPA)

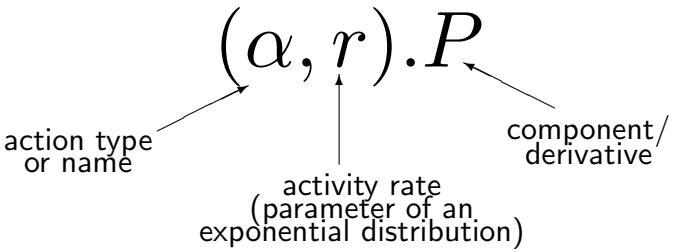
- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

Performance Evaluation Process Algebra (PEPA)

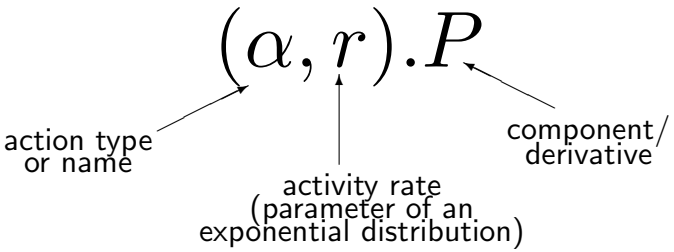
- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

Performance Evaluation Process Algebra (PEPA)

- Models are constructed from components which engage in activities.

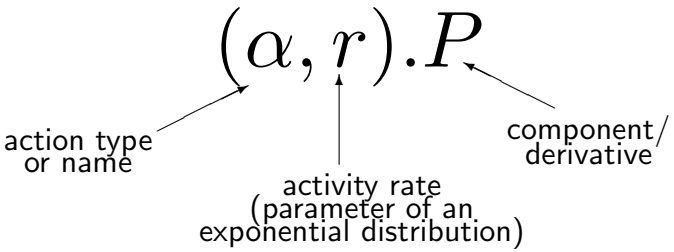


- The language is used to generate a CTMC for performance modelling.

PEPA MODEL

Performance Evaluation Process Algebra (PEPA)

- Models are constructed from components which engage in activities.

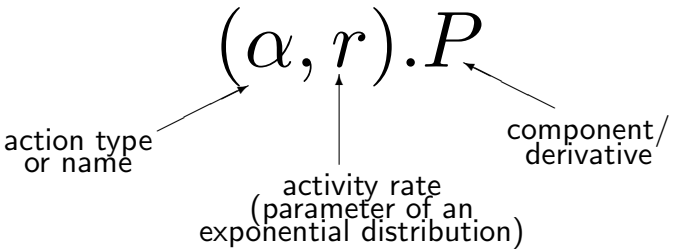


- The language is used to generate a CTMC for performance modelling.

PEPA MODEL $\xrightarrow{\text{SOS rules}}$

Performance Evaluation Process Algebra (PEPA)

- Models are constructed from components which engage in activities.

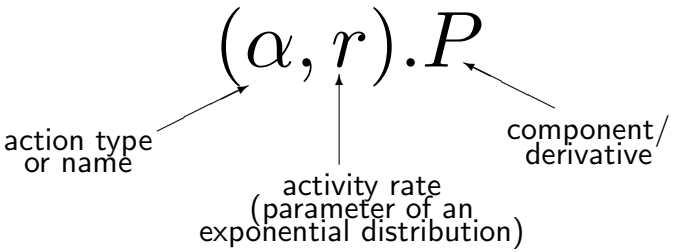


- The language is used to generate a CTMC for performance modelling.

PEPA MODEL —SOS rules→ LABELLED MULTI-TRANSITION SYSTEM

Performance Evaluation Process Algebra (PEPA)

- Models are constructed from components which engage in activities.

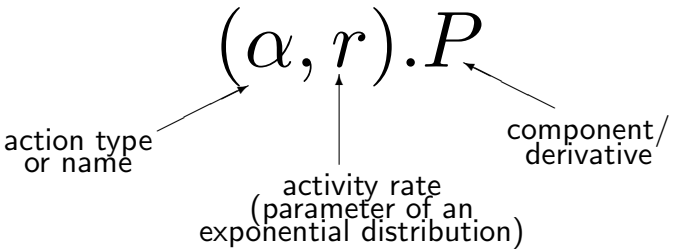


- The language is used to generate a CTMC for performance modelling.

PEPA MODEL —SOS rules→ LABELLED MULTI-TRANSITION SYSTEM —state transition diagram→

Performance Evaluation Process Algebra (PEPA)

- Models are constructed from components which engage in activities.



- The language is used to generate a CTMC for performance modelling.

PEPA MODEL —SOS rules→ LABELLED MULTI-TRANSITION SYSTEM —state transition diagram→ CONTINUOUS TIME MARKOV CHAIN $\mathbf{Q}$

The syntax of PEPA

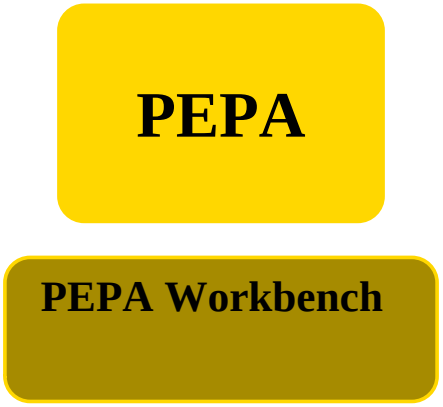
$$
\begin{array}{lll}
S & ::= & \quad (\textit{sequential components}) \\
 & & (\alpha, r).S \quad (\textit{prefix}) \\
 & | & S + S \quad (\textit{choice}) \\
 & | & I \quad (\textit{identifier})
\end{array}
$$

$$
\begin{array}{lll}
P & ::= & \quad (\textit{model components}) \\
 & & P \underset{L}{\bowtie} P \quad (\textit{cooperation}) \\
 & | & P/L \quad (\textit{hiding}) \\
 & | & I \quad (\textit{identifier})
\end{array}
$$

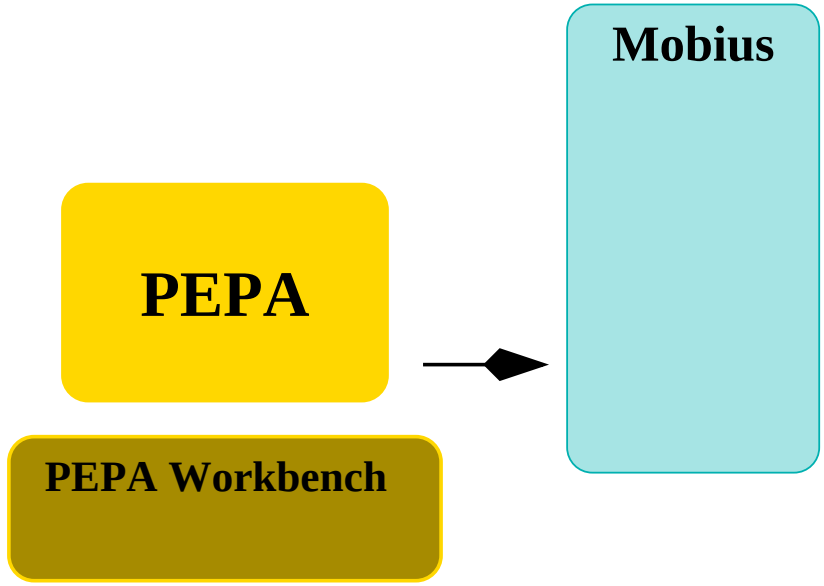
Tools and Integration

PEPA

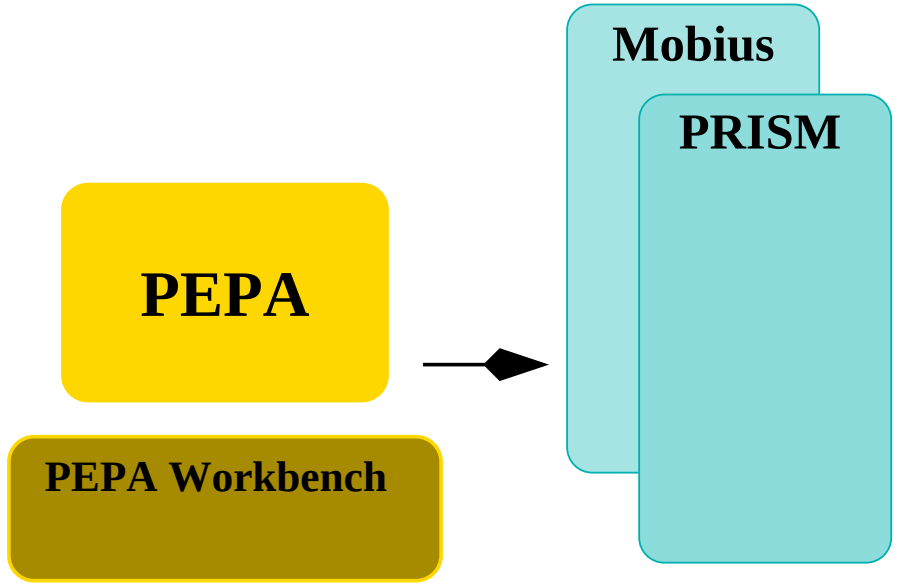
Tools and Integration



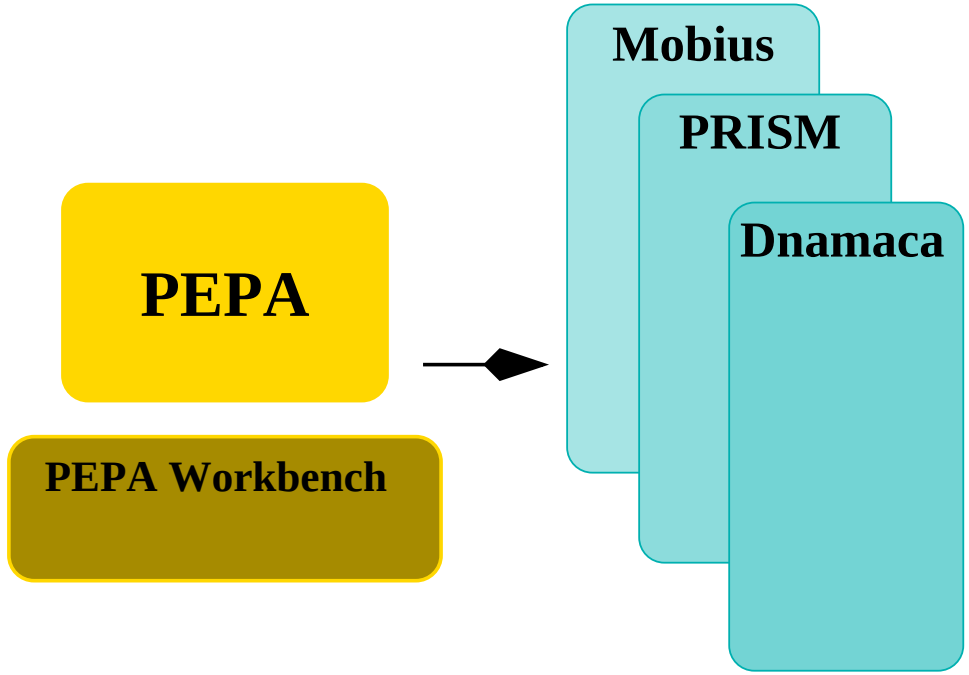
Tools and Integration



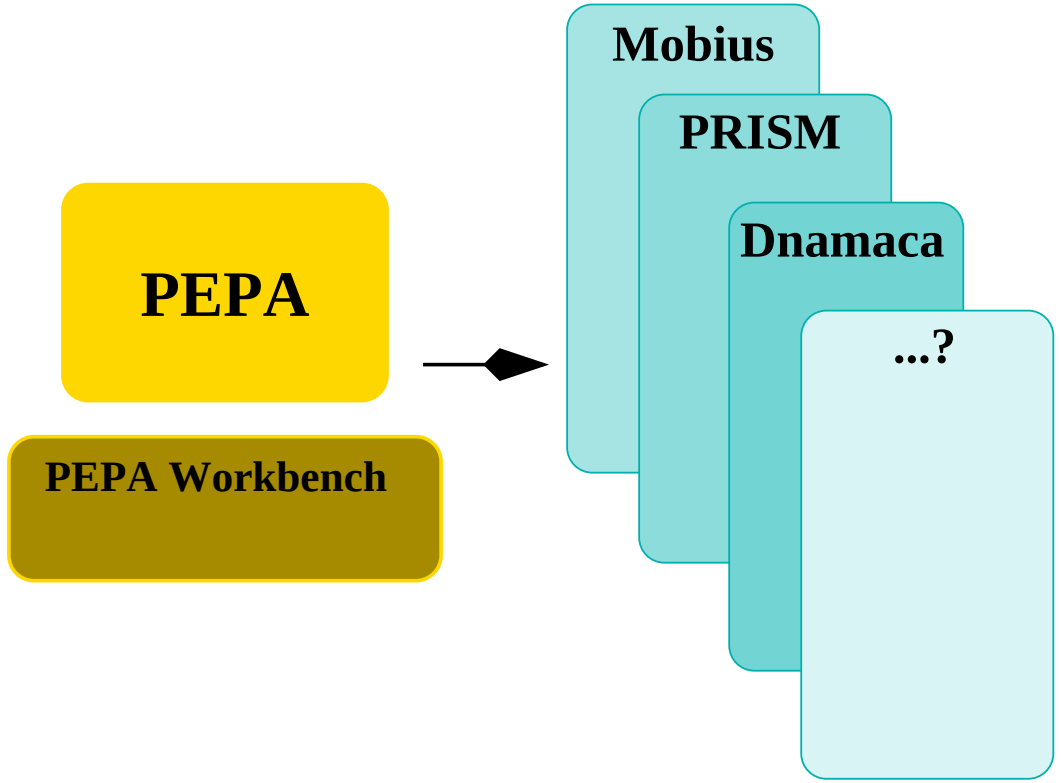
Tools and Integration



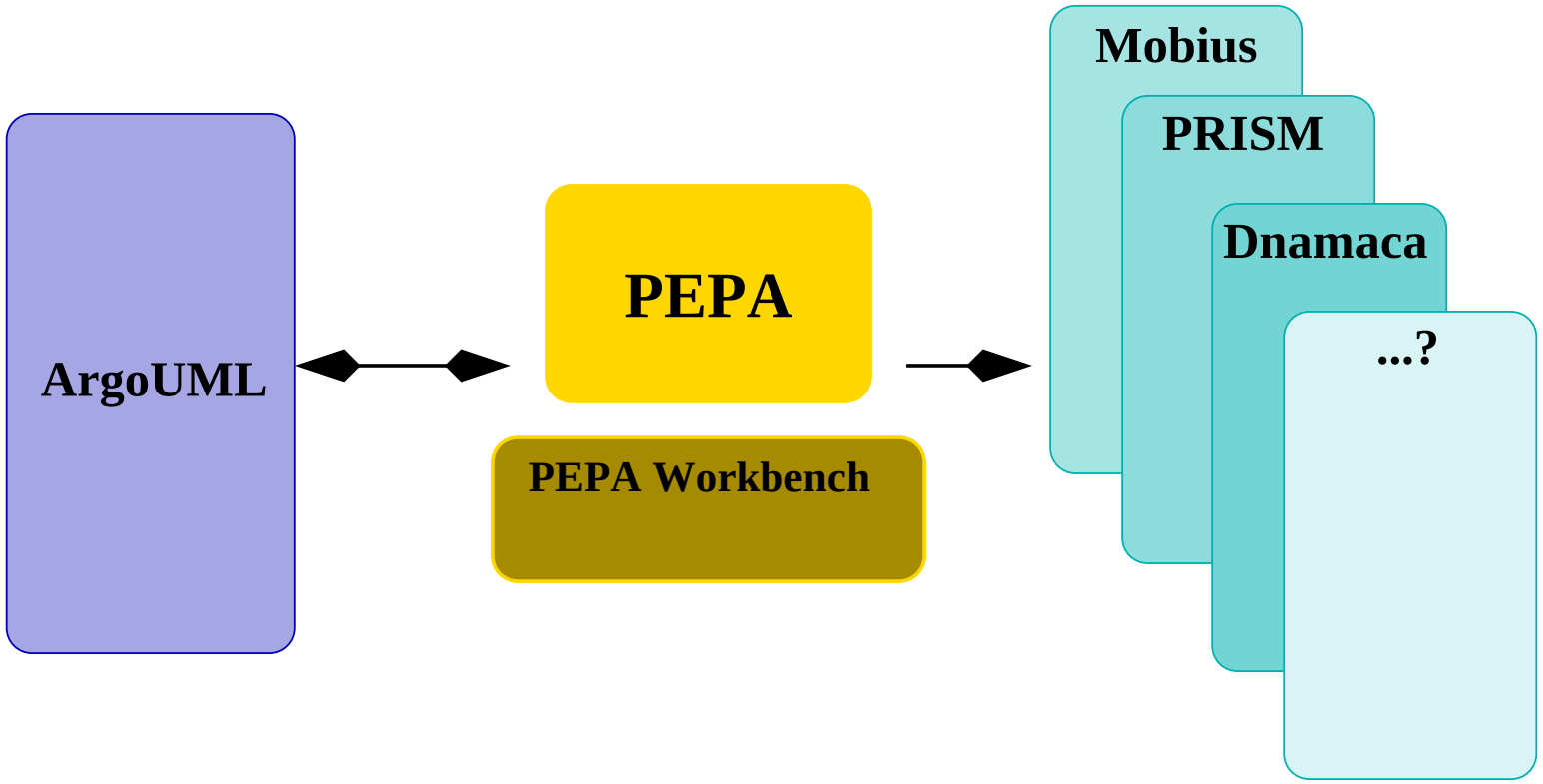
Tools and Integration



Tools and Integration



Tools and Integration



A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

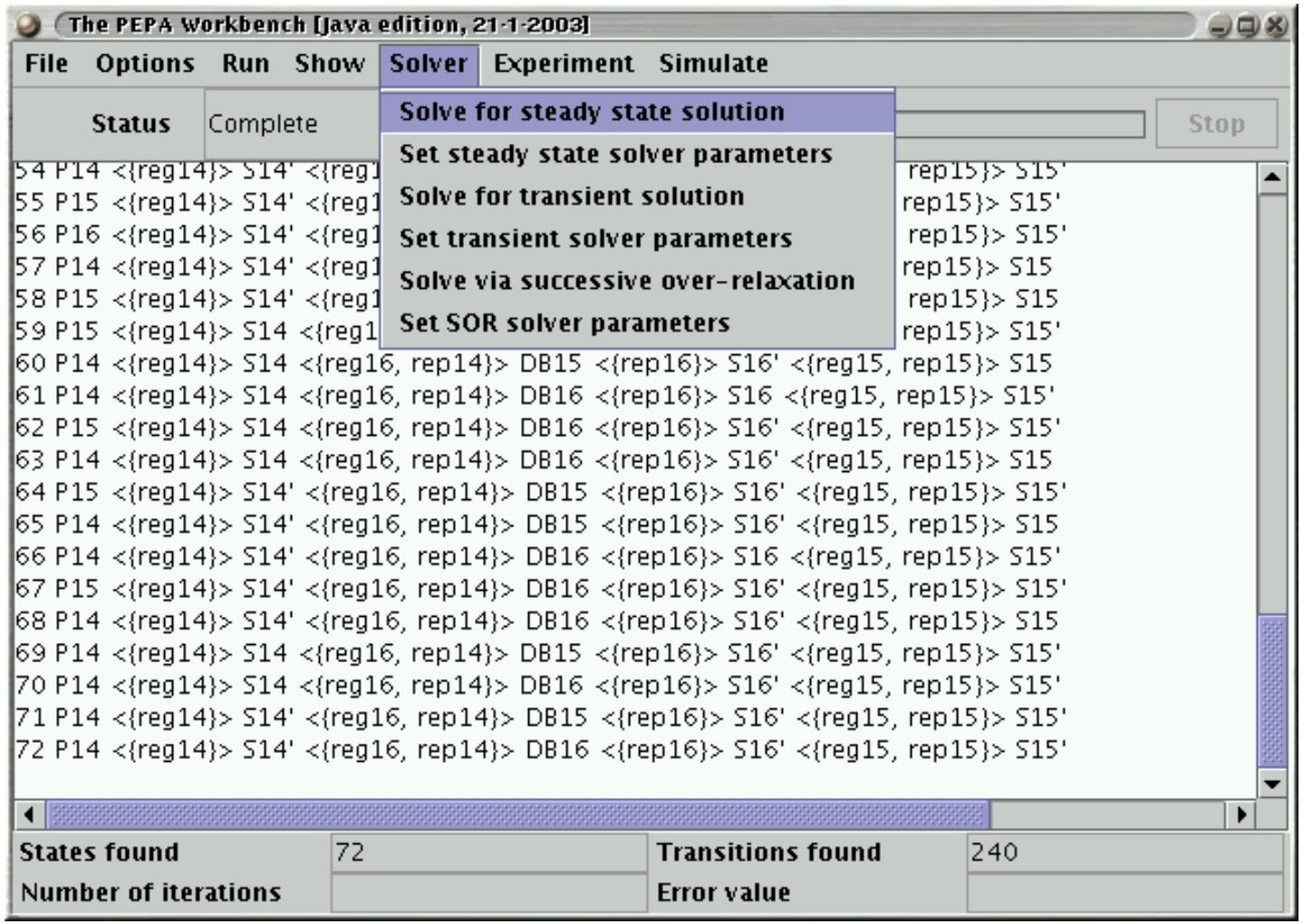
A first PEPA tool: The PEPA Workbench

- Our first PEPA tool was the PEPA Workbench, implemented in Standard ML.
- The PEPA syntax can be represented simply as an ML datatype.

```
datatype Component =
     PREFIX of (Activity * Rate) * Component          (* . *)
   | CHOICE of Component * Component                  (* + *)
   | COOP of Component * Component * Activity list    (* ⋈ *)
   | HIDING of Component * Activity list              (* / *)
   | VAR of Identifier                                (* X *)
   | DEF of Identifier * Component * Component        (* =def= *)
```

Beyond ML

- With the ML edition of the PEPA Workbench it was possible to solve small models using exterior solvers such as Maple and Matlab.
- However, users of the workbench wanted to make more detailed models (with larger state spaces).
- The ML edition of the PEPA Workbench could not solve Robert Holton's robotic workcell model efficiently enough so we interfaced it with an external solver written in C.
- Other users wanted to run the workbench on Solaris, Windows and Linux machines so we ported the Workbench and the solver to Java.


The PEPA Workbench [Java edition, 21-1-2003]
File Options Run Show Solver Experiment Simulate
Status Complete
Stop
Solve for steady state solution
Set steady state solver parameters
Solve for transient solution
Set transient solver parameters
Solve via successive over-relaxation
Set SOR solver parameters
60 P14 <{reg14}> S14 <{reg16, rep14}> DB15 <{rep16}> S16' <{reg15, rep15}> S15
61 P14 <{reg14}> S14 <{reg16, rep14}> DB16 <{rep16}> S16 <{reg15, rep15}> S15'
62 P15 <{reg14}> S14 <{reg16, rep14}> DB16 <{rep16}> S16' <{reg15, rep15}> S15'
63 P14 <{reg14}> S14 <{reg16, rep14}> DB16 <{rep16}> S16' <{reg15, rep15}> S15
64 P15 <{reg14}> S14' <{reg16, rep14}> DB15 <{rep16}> S16' <{reg15, rep15}> S15'
65 P14 <{reg14}> S14' <{reg16, rep14}> DB15 <{rep16}> S16' <{reg15, rep15}> S15
66 P14 <{reg14}> S14' <{reg16, rep14}> DB16 <{rep16}> S16 <{reg15, rep15}> S15'
67 P15 <{reg14}> S14' <{reg16, rep14}> DB16 <{rep16}> S16' <{reg15, rep15}> S15'
68 P14 <{reg14}> S14' <{reg16, rep14}> DB16 <{rep16}> S16' <{reg15, rep15}> S15
69 P14 <{reg14}> S14 <{reg16, rep14}> DB15 <{rep16}> S16' <{reg15, rep15}> S15'
70 P14 <{reg14}> S14 <{reg16, rep14}> DB16 <{rep16}> S16' <{reg15, rep15}> S15'
71 P14 <{reg14}> S14' <{reg16, rep14}> DB15 <{rep16}> S16' <{reg15, rep15}> S15'
72 P14 <{reg14}> S14' <{reg16, rep14}> DB16 <{rep16}> S16' <{reg15, rep15}> S15'
States found 72
Transitions found 240
Number of iterations
Error value

PEPA in Möbius

- Graham Clark implemented an editor for PEPA in the Möbius multi-paradigm modelling framework, extending PEPA to PEPA$_k$ with guards and parameters.

```
Consume[a,b] =    [a > 0] => (outa, ar).Consume[a-1,b]
 + [(b > 0) && (a == 0)] => (outb, br).Consume[a,b-1];

Breakdown = (outa, T).Breakdown
     + (fail,fr).(recover,RecoverRate).Breakdown ;

System = Consume[5,4] <outa> Breakdown;
```

PEPA in Möbius

- Graham Clark implemented an editor for PEPA in the Möbius multi-paradigm modelling framework, extending PEPA to PEPA$_k$ with guards and parameters.

```
Consume[a,b] =    [a > 0] => (outa, ar).Consume[a-1,b]
 + [(b > 0) && (a == 0)] => (outb, br).Consume[a,b-1];

Breakdown = (outa, T).Breakdown
     + (fail,fr).(recover,RecoverRate).Breakdown ;

System = Consume[5,4] <outa> Breakdown;
```

PEPA in Möbius

- Graham Clark implemented an editor for PEPA in the Möbius multi-paradigm modelling framework, extending PEPA to PEPA$_k$ with guards and parameters.

```
Consume[a,b] =    [a > 0] => (outa, ar).Consume[a-1,b]
 + [(b > 0) && (a == 0)] => (outb, br).Consume[a,b-1];

Breakdown = (outa, T).Breakdown
     + (fail,fr).(recover,RecoverRate).Breakdown ;

System = Consume[5,4] <outa> Breakdown;
```

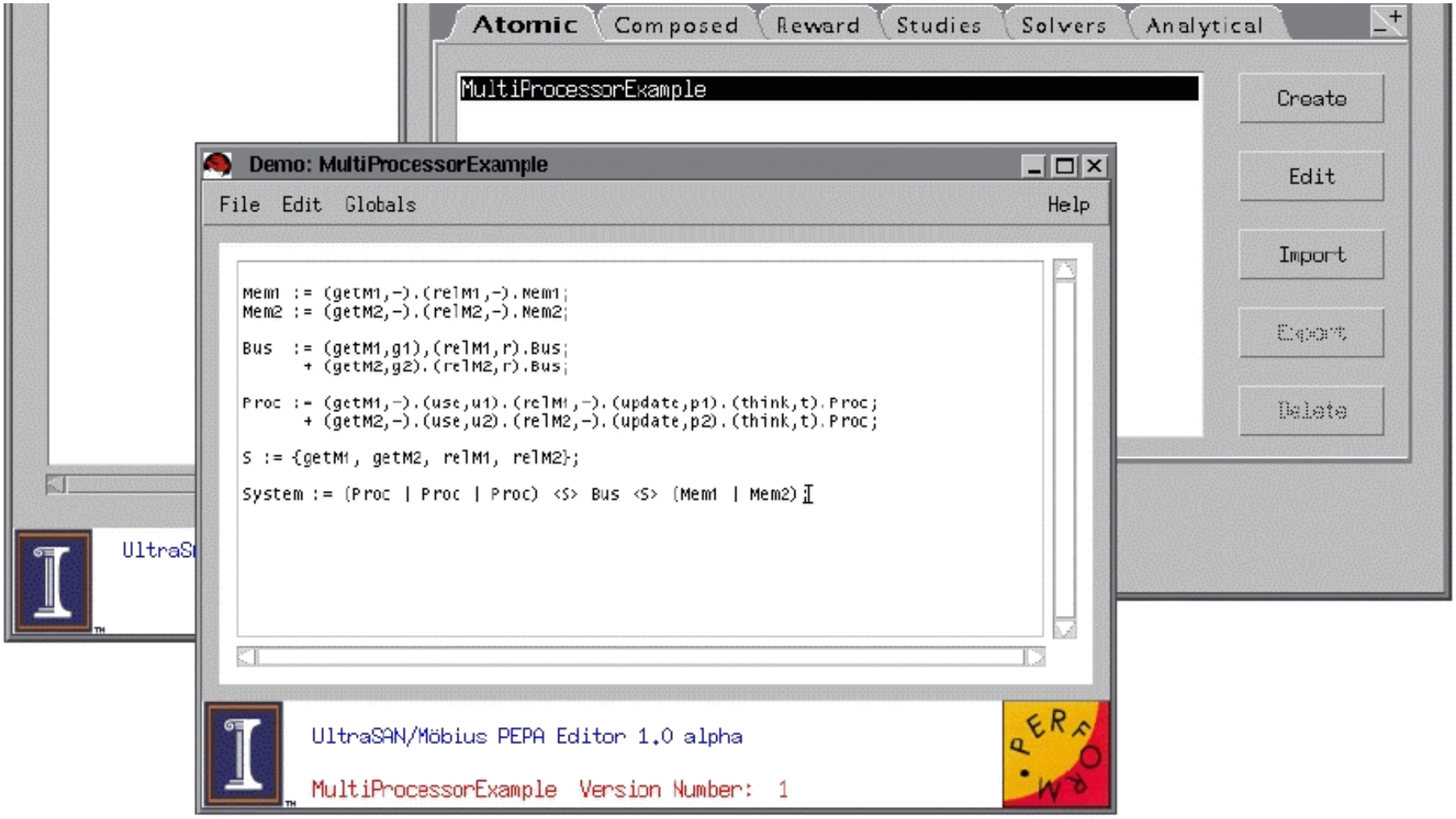
PEPA in Möbius

- Graham Clark implemented an editor for PEPA in the Möbius multi-paradigm modelling framework, extending PEPA to PEPA$_k$ with guards and parameters.

```
Consume[a,b] =    [a > 0] => (outa, ar).Consume[a-1,b]
 + [(b > 0) && (a == 0)] => (outb, br).Consume[a,b-1];

Breakdown = (outa, T).Breakdown
     + (fail,fr).(recover,RecoverRate).Breakdown ;

System = Consume[5,4] <outa> Breakdown;
```


Atomic
Composed
Reward
Studies
Solvers
Analytical
MultiProcessorExample
Create
Edit
Import
Export
Delete
Demo: MultiProcessorExample
File
Edit
Globals
Help
Mem1 := (getM1,-).(relM1,-).Mem1;
Mem2 := (getM2,-).(relM2,-).Mem2;
Bus := (getM1,g1),(relM1,r).Bus;
+ (getM2,g2).(relM2,r).Bus;
Proc := (getM1,-).(use,u1).(relM1,-).(update,p1).(think,t).Proc;
+ (getM2,-).(use,u2).(relM2,-).(update,p2).(think,t).Proc;
S := {getM1, getM2, relM1, relM2};
System := (Proc | Proc | Proc) <S> Bus <S> (Mem1 | Mem2)
UltraSAN/Möbius PEPA Editor 1.0 alpha
MultiProcessorExample Version Number: 1
PERFORM

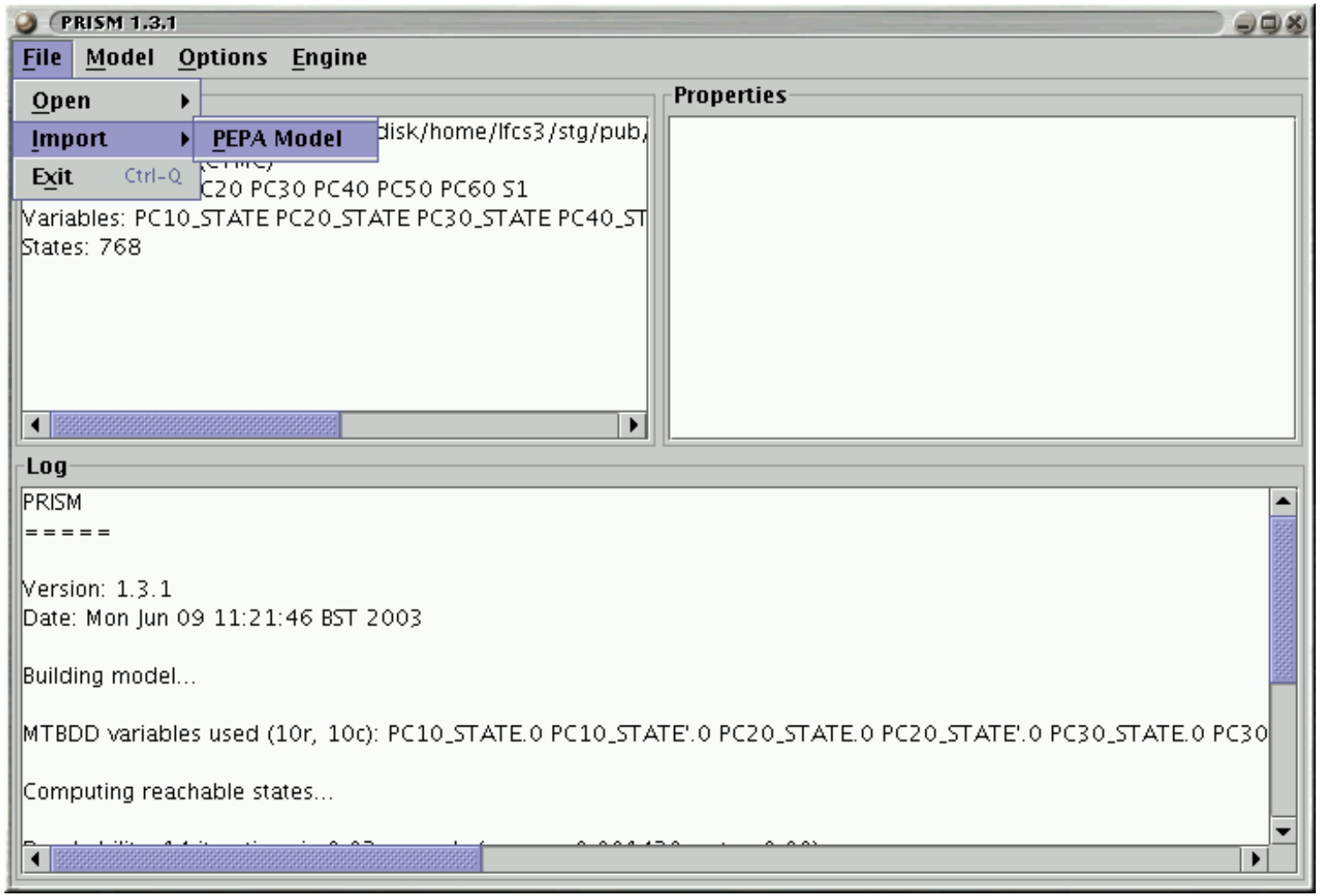
PEPA and PRISM

- PRISM is a probabilistic model checker which supports modelling in DTMCs, CTMCs and MDPs with PTCL and CSL model checking.

- The matrix storing the state space of the system is expressed as an MTBDD built using the CUDD package.

- Support for the PEPA language in PRISM was provided in two steps:

  1. extending the PRISM input language with a new **system** construct providing the PEPA composition operators for synchronisation over activity sets and hiding; and
  2. compiling the PEPA language into the extended PRISM language.


PRISM 1.3.1
File
Model
Options
Engine
Open
Import
Exit
Ctrl-Q
PEPA Model
disk/home/lfcs3/stg/pub/
C20 PC30 PC40 PC50 PC60 S1
Variables: PC10_STATE PC20_STATE PC30_STATE PC40_ST
States: 768
Properties
Log
PRISM
=====
Version: 1.3.1
Date: Mon Jun 09 11:21:46 BST 2003
Building model...
MTBDD variables used (10r, 10c): PC10_STATE.0 PC10_STATE'.0 PC20_STATE.0 PC20_STATE'.0 PC30_STATE.0 PC30
Computing reachable states...

PEPA modelling with PRISM

- PEPA modelling with PRISM has proved to be very effective in practice. The largest PEPA model so far solved has been solved with PRISM.

- However, there are a number of places where the user needs to understand the tool chain thoroughly:
  - The PEPA-to-PRISM compiler rejects (valid) PEPA models which use active/active synchronisation or anonymous components;
  - The compiler can fail during compilation with Java stack overflow;
  - PRISM can reject models which the PEPA-to-PRISM compiler outputs;
  - The CUDD package can fail with out-of-memory errors and need to be reconfigured.

PEPA and IPC/Dnamaca

The Imperial PEPA compiler (IPC) compiles PEPA models into Petri nets which are solved with the Dnamaca solver. Dnamaca provides a number of numerical solvers and outperforms PRISM on small PEPA models.

```
P1 = (start, r1).P2; =
```

```
\transition{P1_start} {
    %% PEPA action type { start }
    \condition{ P1 > 0 }
    \action {
        next -> P1 = P1 - 1;
        next -> P2 = P2 + 1;
    }
    \priority{1}
    \rate{ PEPA_r1 }
}
```

Synchronisation in Dnamaca

Suppose that two copies of P synchronise on the run activity.

```
P2 = (run, r2).P3; =
```

```
\transition{P2_run__P2_1_run} {
    %% PEPA action type { run }
    \condition{ P2 > 0 && P2_1 > 0 }
    \action {
        next -> P2_1 = P2_1 - 1;
        next -> P3_1 = P3_1 + 1;
        next -> P2 = P2 - 1;
        next -> P3 = P3 + 1;
    }
    \priority{1}
    \rate{ PEPA_r2 }
}
```

PEPA modelling with IPC and Dnamaca

- More of the PEPA language is supported by IPC/Dnamaca than by PRISM. Active/active synchronisation and anonymous components are supported.

- However, there are still a number of places where the user needs to understand the tool chain thoroughly:
  - The IPC compiler can fail during compilation with Haskell memory exhaustion;
  - Dnamaca can reject models which IPC outputs; and
  - Dnamaca's numerical procedures can fail to converge.

Dnamaca features and PEPA extensions

- Because Dnamaca supports non-Markovian modelling, beyond the models which are expressible in PEPA, it would be possible to support PEPA extensions with Dnamaca:
  - PEPA$_k$ guards and parameters; *[Clark, Sanders, '01]*
  - Weighted (WSCCS-style) PEPA; *[Bradley, '02]*
  - PEPA nets with priorities; *[Gilmore, Hillston, Ribaudo, Kloul, '03]*
  - Semi-Markov PEPA; *[Bradley, '03]*
  - . . .

PEPA and Stochastic Petri nets

Since the initial development, the relationship between PEPA and stochastic Petri nets has been studied and investigated.

- Marina Ribaudo developed an SPN semantics for PEPA in her thesis [1994].
- She also investigated the relationship between the aggregation offered within PEPA and that offered by Stochastic Well-formed Nets [1995].
- Donatelli, Hermanns, Hillston and Ribaudo informally investigated the expressiveness of the two formalisms via case study [1995].
- Hillston, Recalde, Ribaudo and Silva developed a compositional mapping from bounded SPNs to PEPA models [2001].

Petri nets and process algebras

- Petri nets provide a graphical presentation of a model which has an easily accessible interpretation and they also have the advantage of being supported by an unambiguous formal interpretation.

- Stochastic process algebras lack the attractive graphical presentation of Petri nets. In contrast though, an explicit compositional structure is imposed on the model. This structure can be exploited for both qualitative and quantitative analysis.

- The present work considers using both Petri nets and process algebras together as a single, structured performance modelling formalism.

PEPA nets

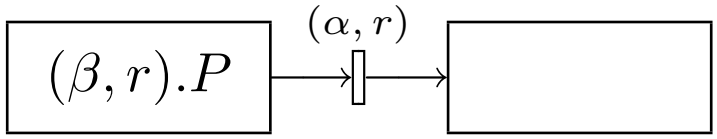
- Coloured Petri nets are a high-level form of classical Petri nets. The plain (indistinguishable) tokens of a classical Petri net are replaced by arbitrary terms which are distinguishable.

- In stochastic Petri nets the transitions from one marking to another are associated with a random variable drawn from an exponential distribution. Here we consider coloured stochastic Petri nets where the colours used as the tokens of the net are PEPA components. We refer to these as PEPA nets.

- Petri nets have previously been combined with other performance modelling formalisms e.g. Bause's Queueing Petri nets and Haverkort's Dynamic Queueing Networks. Other extensions of (non-stochastic) Petri nets have programmable tokens, e.g. Valk's Elementary Object systems.

Transitions in a PEPA net

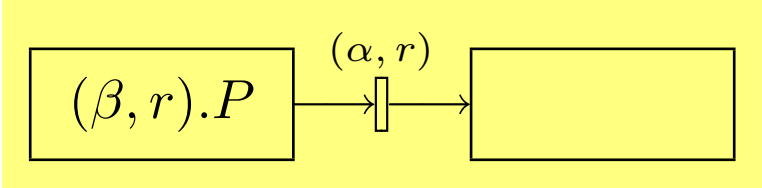
- A transition in a PEPA net takes place whenever a transition of a PEPA component can occur (either individually, or in co-operation with another component at the same place).



- Transitions of PEPA components are used to model small-scale changes of state as components undertake activities. The PEPA net formalism does not allow components at different places in the net to co-operate on a shared activity so transitions have only local effect.

Transitions in a PEPA net

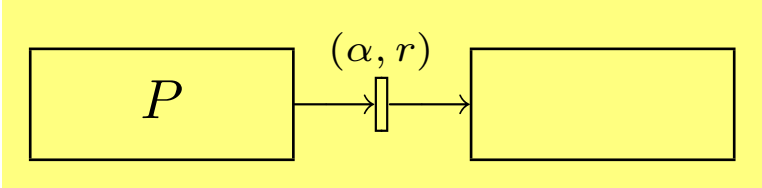
- A transition in a PEPA net takes place whenever a transition of a PEPA component can occur (either individually, or in co-operation with another component at the same place).



- Transitions of PEPA components are used to model small-scale changes of state as components undertake activities. The PEPA net formalism does not allow components at different places in the net to co-operate on a shared activity so transitions have only local effect.

Transitions in a PEPA net

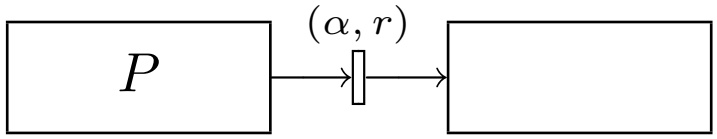
- A transition in a PEPA net takes place whenever a transition of a PEPA component can occur (either individually, or in co-operation with another component at the same place).



- Transitions of PEPA components are used to model small-scale changes of state as components undertake activities. The PEPA net formalism does not allow components at different places in the net to co-operate on a shared activity so transitions have only local effect.

Transitions in a PEPA net

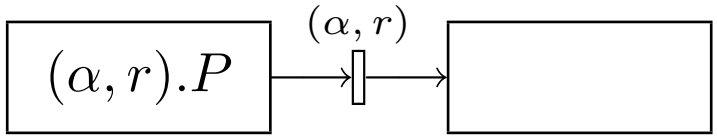
- A transition in a PEPA net takes place whenever a transition of a PEPA component can occur (either individually, or in co-operation with another component at the same place).



- Transitions of PEPA components are used to model small-scale changes of state as components undertake activities. The PEPA net formalism does not allow components at different places in the net to co-operate on a shared activity so transitions have only local effect.

Firings in a PEPA net

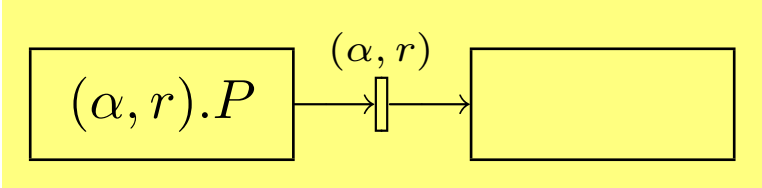
- Firings in a PEPA net are used to model macro-step changes of state such as breakdowns and repairs, one thread yielding to another, or a mobile software agent moving from one network host to another.



- A firing causes the transfer of one token from one place to another. The token which is moved is a PEPA component, which causes a change in both the input place (where existing co-operations now can no longer take place) and the output place (where previously disabled co-operations are now enabled).

Firings in a PEPA net

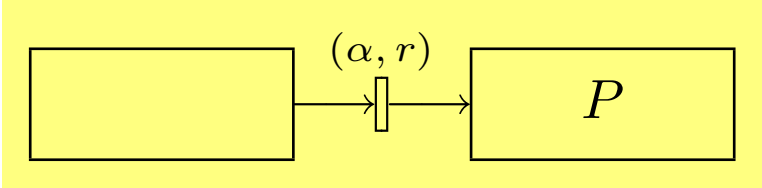
- Firings in a PEPA net are used to model macro-step changes of state such as breakdowns and repairs, one thread yielding to another, or a mobile software agent moving from one network host to another.



- A firing causes the transfer of one token from one place to another. The token which is moved is a PEPA component, which causes a change in both the input place (where existing co-operations now can no longer take place) and the output place (where previously disabled co-operations are now enabled).

Firings in a PEPA net

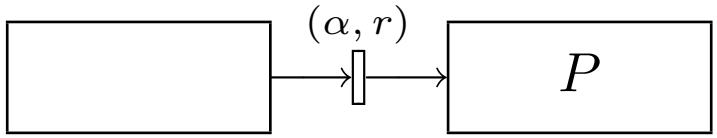
- Firings in a PEPA net are used to model macro-step changes of state such as breakdowns and repairs, one thread yielding to another, or a mobile software agent moving from one network host to another.



- A firing causes the transfer of one token from one place to another. The token which is moved is a PEPA component, which causes a change in both the input place (where existing co-operations now can no longer take place) and the output place (where previously disabled co-operations are now enabled).

Firings in a PEPA net

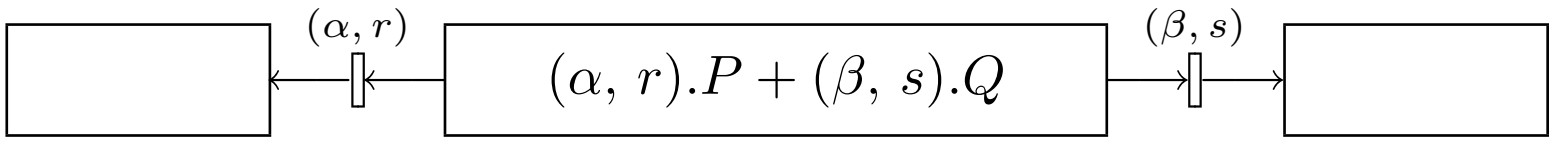
- Firings in a PEPA net are used to model macro-step changes of state such as breakdowns and repairs, one thread yielding to another, or a mobile software agent moving from one network host to another.



- A firing causes the transfer of one token from one place to another. The token which is moved is a PEPA component, which causes a change in both the input place (where existing co-operations now can no longer take place) and the output place (where previously disabled co-operations are now enabled).

Choice in a PEPA net

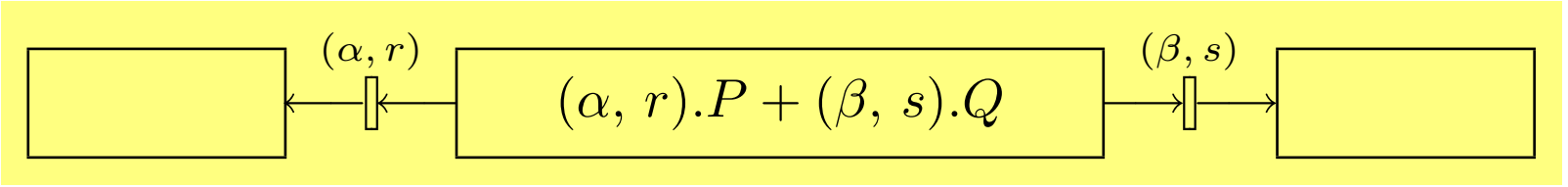
- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.



- The outcome of a choice is governed by a race condition.

Choice in a PEPA net

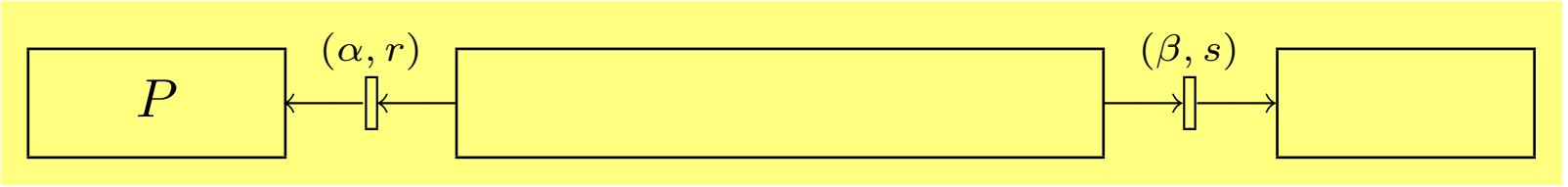
- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.



- The outcome of a choice is governed by a race condition.

Choice in a PEPA net

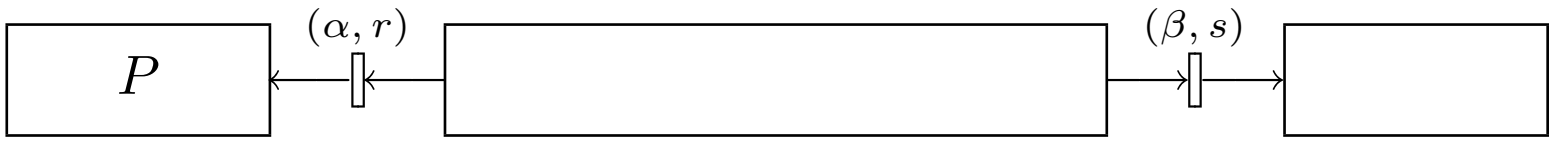
- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.



- The outcome of a choice is governed by a race condition.

Choice in a PEPA net

- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.



- The outcome of a choice is governed by a race condition.

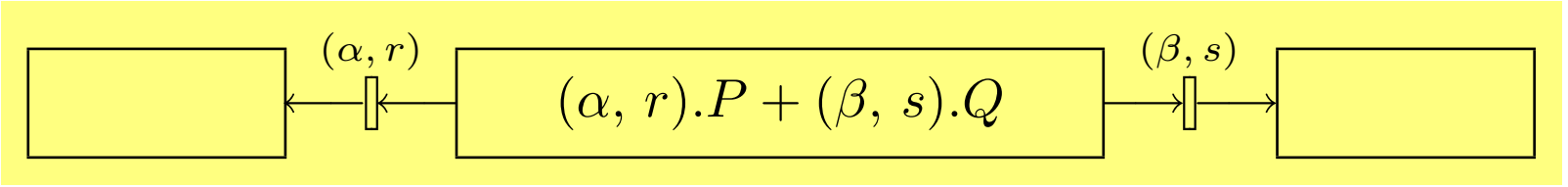
Choice in a PEPA net

- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.

- The outcome of a choice is governed by a race condition.

Choice in a PEPA net

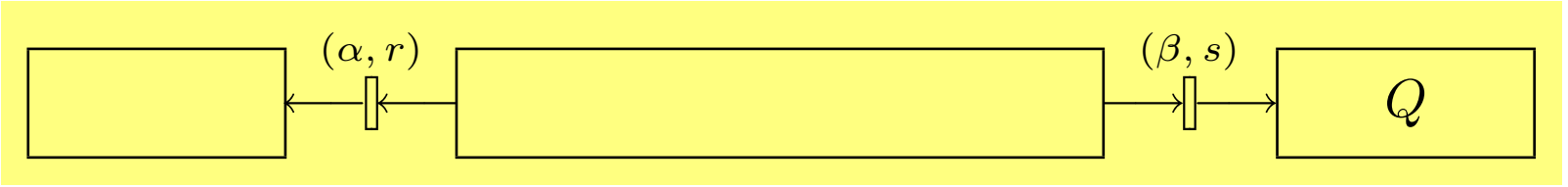
- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.



- The outcome of a choice is governed by a race condition.

Choice in a PEPA net

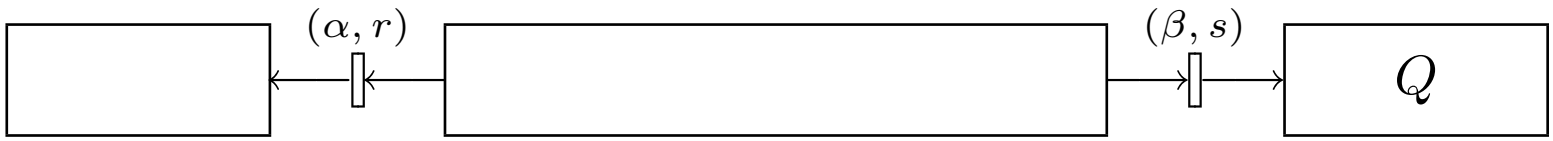
- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.



- The outcome of a choice is governed by a race condition.

Choice in a PEPA net

- Choices in a PEPA net occur when the token has a choice of possible behaviours and a choice of possible output places. Choices are used to model decisions in a system.



- The outcome of a choice is governed by a race condition.

Markings in a PEPA net

- The marking of a PEPA net is made up of a list of PEPA contexts, one at each place in the net, where the system descriptions can also contain component cells.

- A cell is a slot to be filled by a component of a particular type.
  - Components which fill these cells can circulate as the tokens of the net.
  - Components which are not in a cell are static and cannot move.

- We use the notation $P[\_]$ to denote a context or a place which could be filled by the PEPA component P or one with the same alphabet.

Markings in a PEPA net

- The marking of a PEPA net is made up of a list of PEPA contexts, one at each place in the net, where the system descriptions can also contain component cells.

- A cell is a slot to be filled by a component of a particular type.
  - Components which fill these cells can circulate as the tokens of the net.
  - Components which are not in a cell are static and cannot move.

- We use the notation $P[\_]$ to denote a context or a place which could be filled by the PEPA component P or one with the same alphabet.

Markings in a PEPA net

- The marking of a PEPA net is made up of a list of PEPA contexts, one at each place in the net, where the system descriptions can also contain component cells.

- A cell is a slot to be filled by a component of a particular type.
  - Components which fill these cells can circulate as the tokens of the net.
  - Components which are not in a cell are static and cannot move.

- We use the notation $P[\_]$ to denote a context or a place which could be filled by the PEPA component P or one with the same alphabet.

Markings in a PEPA net

- The marking of a PEPA net is made up of a list of PEPA contexts, one at each place in the net, where the system descriptions can also contain component cells.

- A cell is a slot to be filled by a component of a particular type.
  - Components which fill these cells can circulate as the tokens of the net.
  - Components which are not in a cell are static and cannot move.

- We use the notation $P[\_]$ to denote a context or a place which could be filled by the PEPA component P or one with the same alphabet.

The syntax of PEPA with cells

$$
\begin{array}{lll}
S & ::= & \quad (\textit{sequential components}) \\
 & & (\alpha, r).S \qquad (\textit{prefix}) \\
 & | & S + S \qquad (\textit{choice}) \\
 & | & I \qquad (\textit{identifier}) \\
 & & \\
P & ::= & \quad (\textit{model components}) \\
 & & P \underset{L}{\bowtie} P \qquad (\textit{cooperation}) \\
 & | & P/L \qquad (\textit{hiding}) \\
 & | & I \qquad (\textit{identifier}) \\
 & | & P[C] \qquad (\textit{cell})
\end{array}
$$

The syntax of PEPA with cells

$$
\begin{array}{lll}
S & ::= & \textit{(sequential components)} \\
 & (\alpha, r).S & \textit{(prefix)} \\
 & \mid \quad S + S & \textit{(choice)} \\
 & \mid \quad I & \textit{(identifier)}
\end{array}
$$

$$
\begin{array}{lll}
P & ::= & \textit{(model components)} \\
 & P \underset{L}{\bowtie} P & \textit{(cooperation)} \\
 & \mid \quad P/L & \textit{(hiding)} \\
 & \mid \quad I & \textit{(identifier)} \\
 & \mid \quad P[C] & \textit{(cell)}
\end{array}
$$

$$
\begin{array}{lll}
C & ::= & \textit{(cell terms)} \\
 & \text{`\_'} & \textit{(empty)} \\
 & \mid \quad P & \textit{(component)}
\end{array}
$$

Example: a mobile agent system

- In this example *a roving agent* visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where *a network probe* is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.
- When it returns to the central co-ordinating site it dumps the data which it has harvested to *the master probe*. The master probe performs a computationally expensive statistical analysis of the data.
- The structure of the system allows this computation to be overlapped with the agent's communication and data gathering.

Example: a mobile agent system

- In this example *a roving agent* visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where *a network probe* is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.
- When it returns to the central co-ordinating site it dumps the data which it has harvested to *the master probe*. The master probe performs a computationally expensive statistical analysis of the data.
- The structure of the system allows this computation to be overlapped with the agent's communication and data gathering.

Example: a mobile agent system

- In this example *a roving agent* visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where *a network probe* is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.
- When it returns to the central co-ordinating site it dumps the data which it has harvested to *the master probe*. The master probe performs a computationally expensive statistical analysis of the data.
- The structure of the system allows this computation to be overlapped with the agent's communication and data gathering.

Example: a mobile agent system

- In this example *a roving agent* visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where *a network probe* is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.
- When it returns to the central co-ordinating site it dumps the data which it has harvested to *the master probe*. The master probe performs a computationally expensive statistical analysis of the data.
- The structure of the system allows this computation to be overlapped with the agent's communication and data gathering.

Example: a mobile agent system

- In this example *a roving agent* visits three sites. It interacts with static software components at these sites and has two kinds of interactions.
- When visiting a site where *a network probe* is present it interrogates the probe for the data which it has gathered on recent patterns of network traffic.
- When it returns to the central co-ordinating site it dumps the data which it has harvested to *the master probe*. The master probe performs a computationally expensive statistical analysis of the data.
- The structure of the system allows this computation to be overlapped with the agent's communication and data gathering.

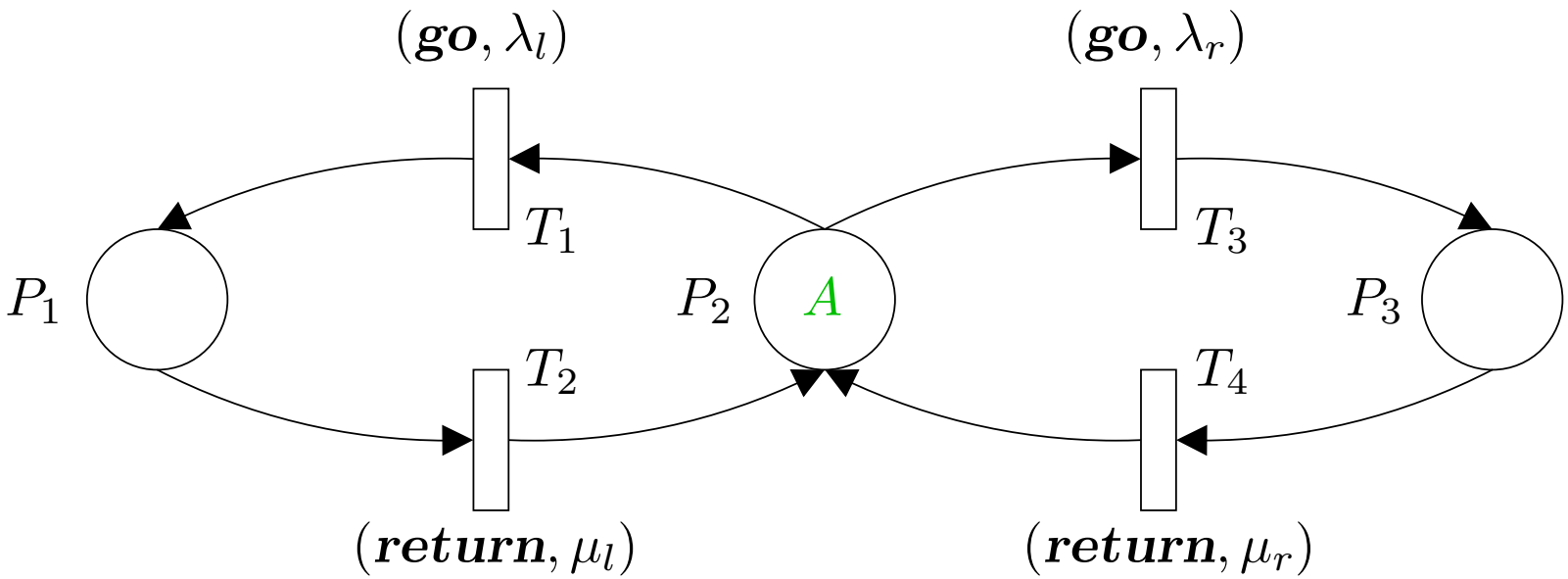
PEPA components

$$
\begin{aligned}
Agent &\stackrel{def}{=} (\mathbf{go}, \lambda).Agent' \\
Agent' &\stackrel{def}{=} (interrogate, r_i).Agent'' \\
Agent'' &\stackrel{def}{=} (\mathbf{return}, \mu).Agent''' \\
Agent''' &\stackrel{def}{=} (dump, r_d).Agent
\end{aligned}
$$

$$
\begin{aligned}
Master &\stackrel{def}{=} (dump, \top).Master' \\
Master' &\stackrel{def}{=} (analyse, r_a).Master
\end{aligned}
$$

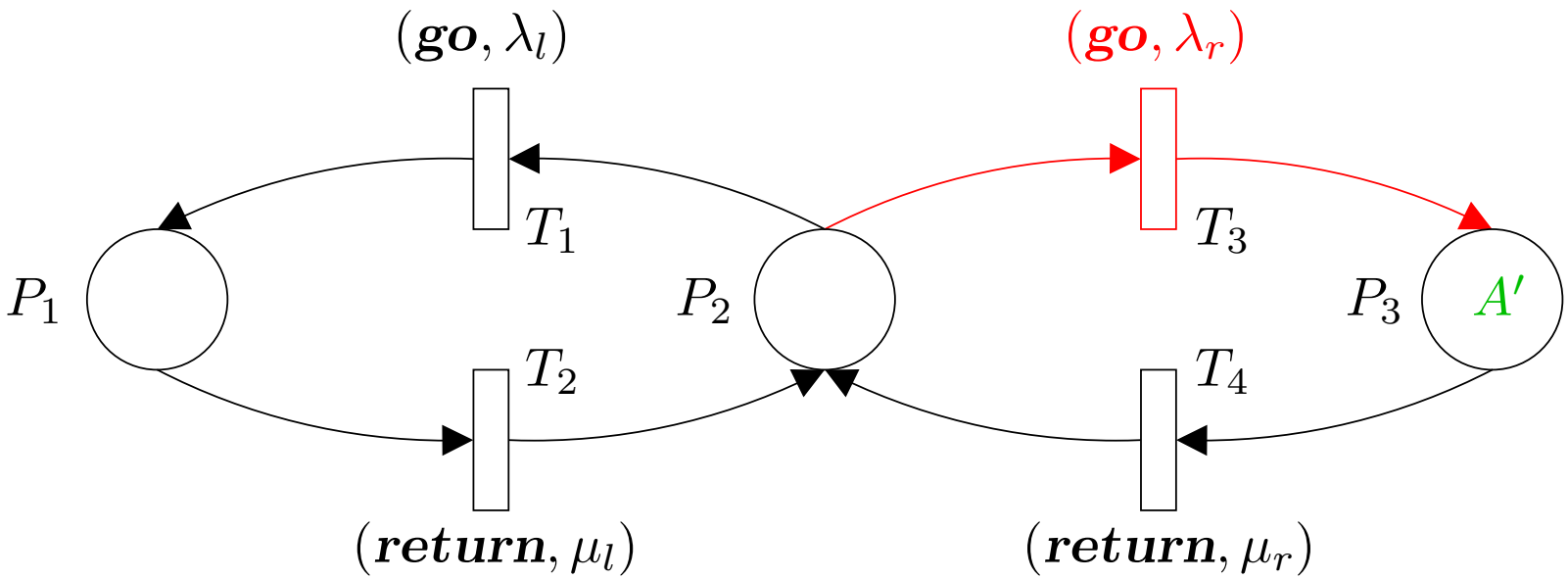
$$
\begin{aligned}
Probe &\stackrel{def}{=} (monitor, r_m).Probe + \\
&\quad\ (interrogate, \top).Probe
\end{aligned}
$$

A mobile agent system (1)



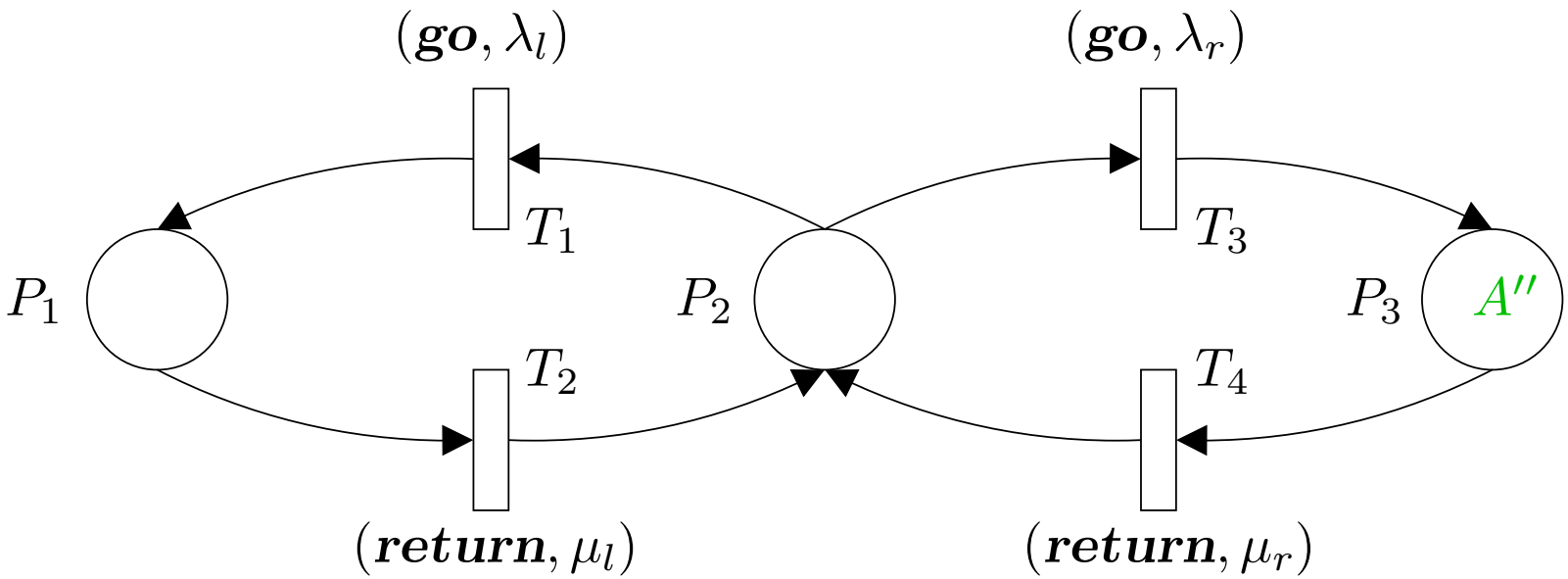
$$Agent \stackrel{def}{=} (\boldsymbol{go}, \lambda).Agent'$$

A mobile agent system (2)



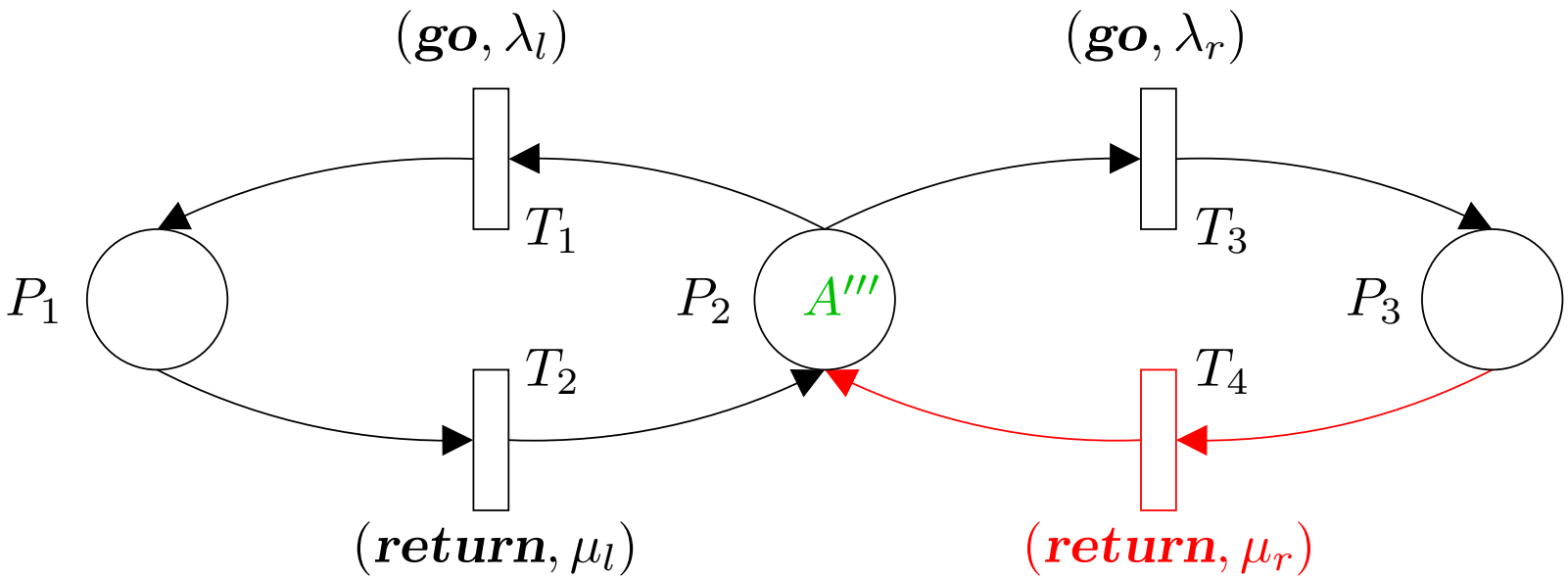
$$Agent' \stackrel{def}{=} (interrogate, r_i).Agent''$$

A mobile agent system (3)



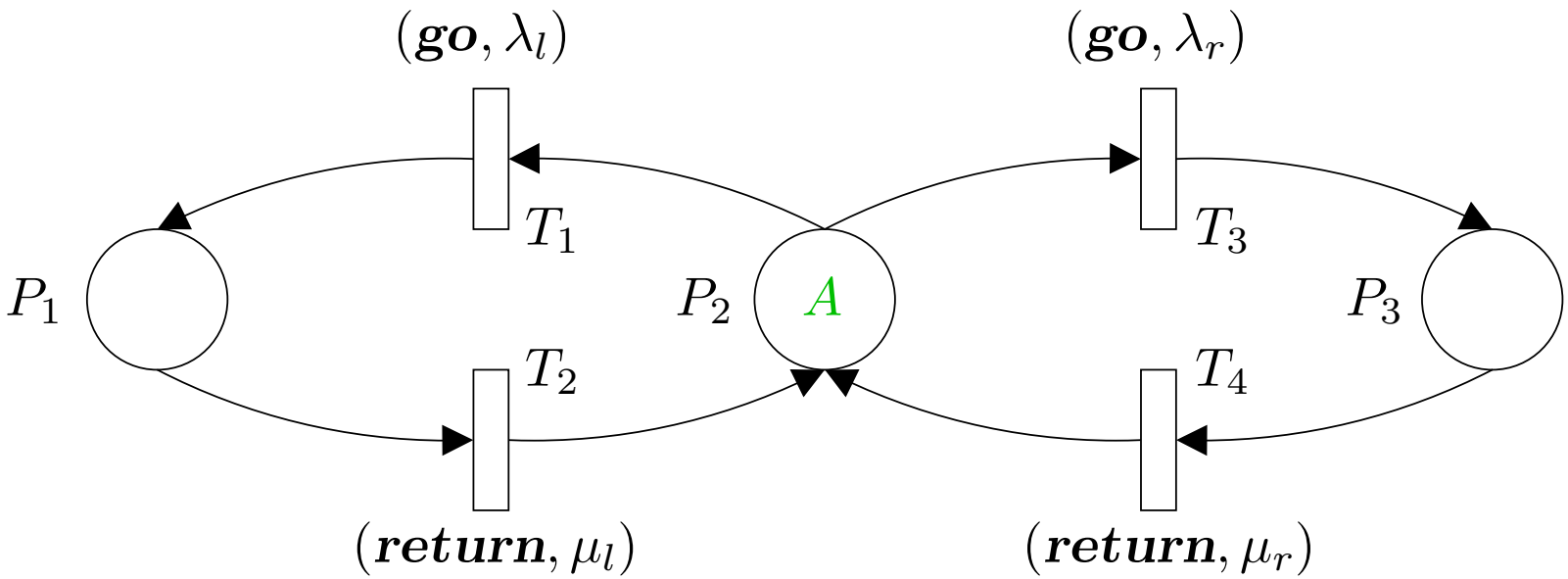
$$Agent'' \stackrel{def}{=} (\mathbf{return}, \mu).Agent'''$$

A mobile agent system (4)



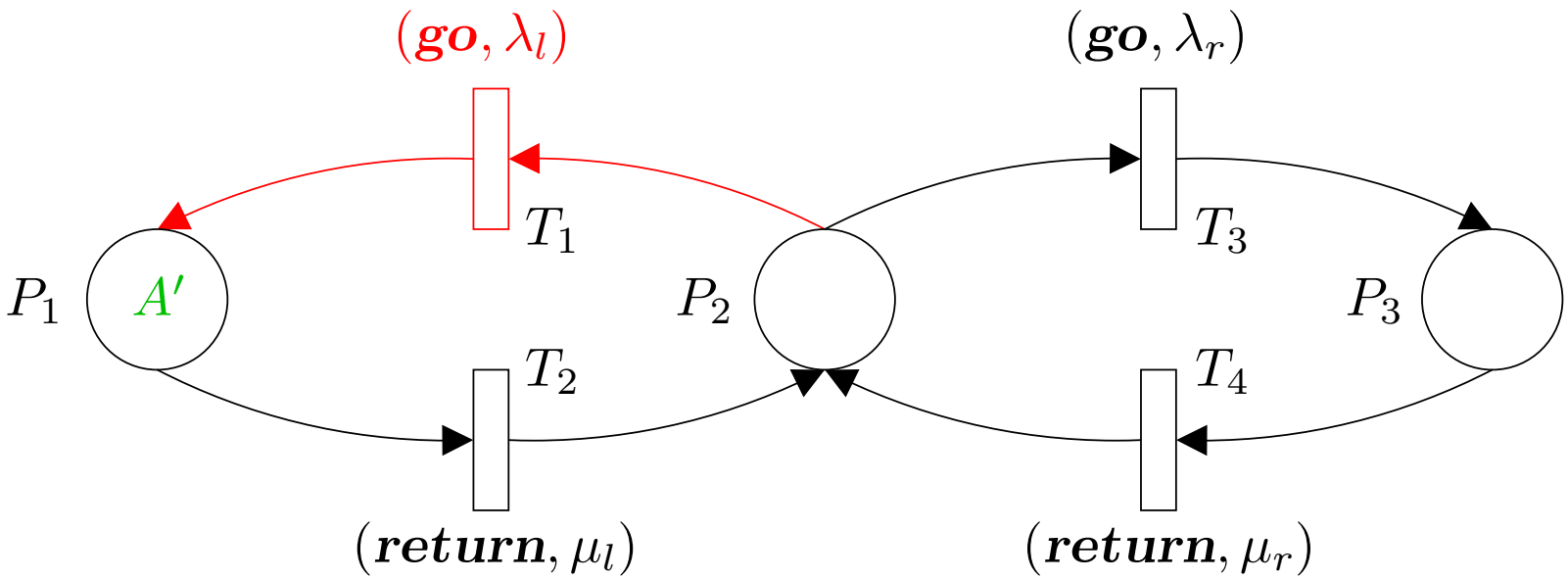
$$Agent''' \stackrel{def}{=} (dump, r_d).Agent$$

A mobile agent system (5)



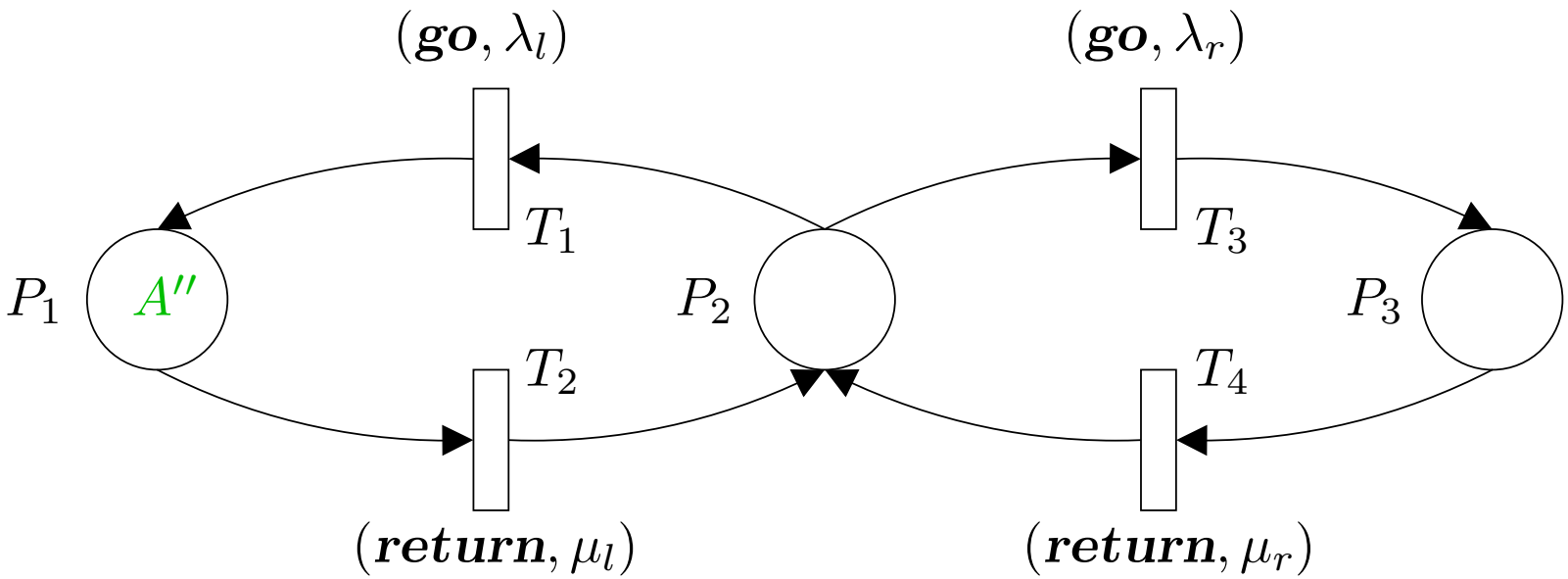
$$Agent \stackrel{def}{=} (\mathbf{go}, \lambda).Agent'$$

A mobile agent system (6)



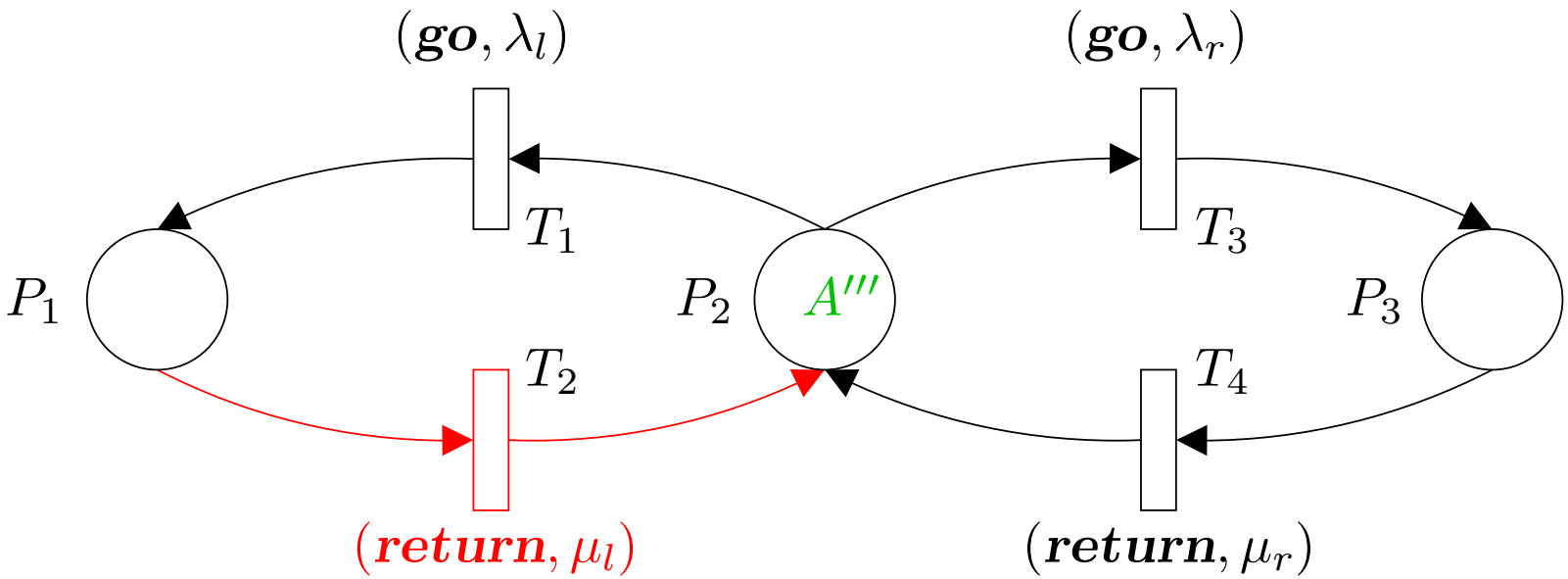
$$Agent' \stackrel{def}{=} (interrogate, r_i).Agent''$$

A mobile agent system (7)



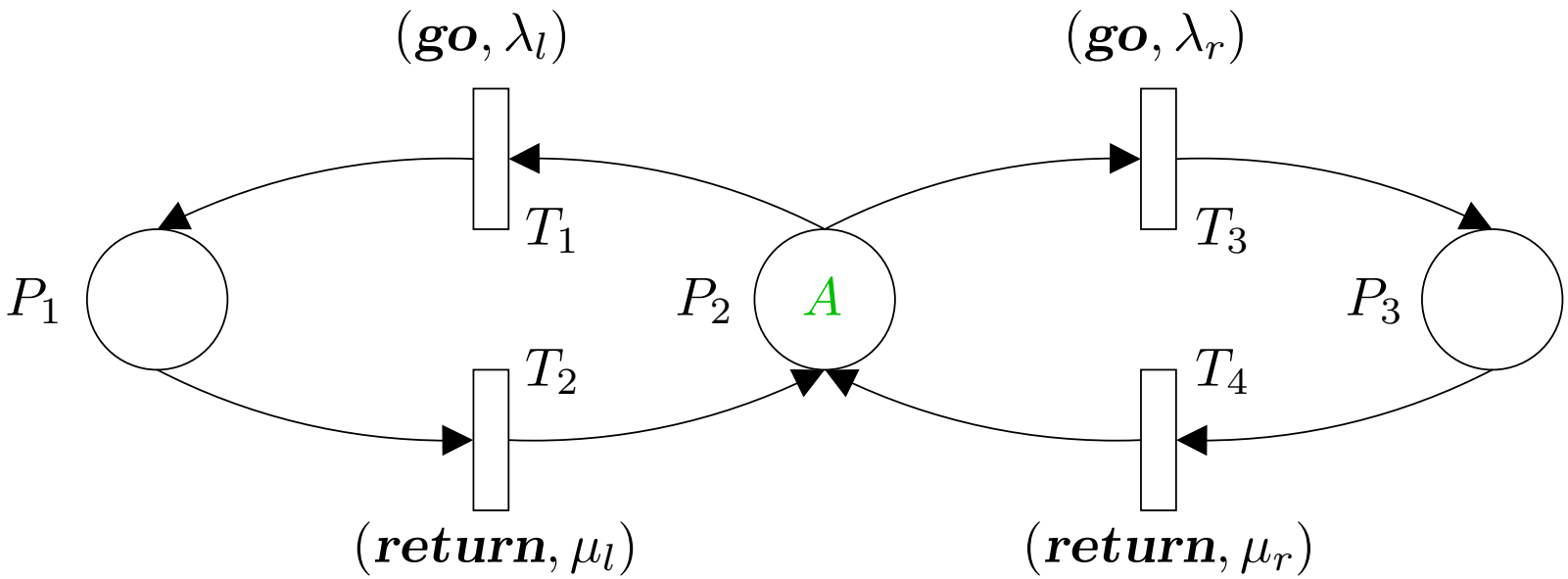
$$Agent'' \stackrel{def}{=} (\mathbf{return}, \mu).Agent'''$$

A mobile agent system (8)



$$Agent''' \stackrel{def}{=} (dump, r_d).Agent$$

A mobile agent system (9)



$$Agent \stackrel{def}{=} (\boldsymbol{go}, \lambda).Agent'$$

Conclusions

- Having a small, stable core language has facilitated many possibilities for us — compiling PEPA and PEPA net models to other formalisms seems to be a very profitable activity.
- However, there are typically many small details in the translation which need to be taken care of.
- It is tempting to lift features of the host tool back to the PEPA level but sometimes desirable properties of the PEPA language are lost.
- It is important to strike a balance between exploiting opportunities and losing theoretical properties.

Acknowledgements

This work has been supported by the DEGAS (Design Environments for Global ApplicationS) project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing.