Petri Net Analyses via Action Planning

Stefan Edelkamp

Computer Science Department
University of Bremen, Germany

Overview

- Directed Model Checking for Petri Nets
  - Heuristics for the Analysis of Petri Nets
  - Planning for the Directed Analysis of Petri Nets
- From Petri Nets to Graphs and Back:
  - Graph Transformation for Planning
  - Graph Transformation via Planning
  - Approximation via Petri Nets & Planning
- Extending Expressiveness of Petri Nets:
  - Colored Petri Nets [Jensen]
  - Predicate/Transition & Administration Nets [Hartmann et al.,Wedde]

Motivation

- Can reachability analysis in Petri nets be accelerated by exploiting heuristic estimates ?

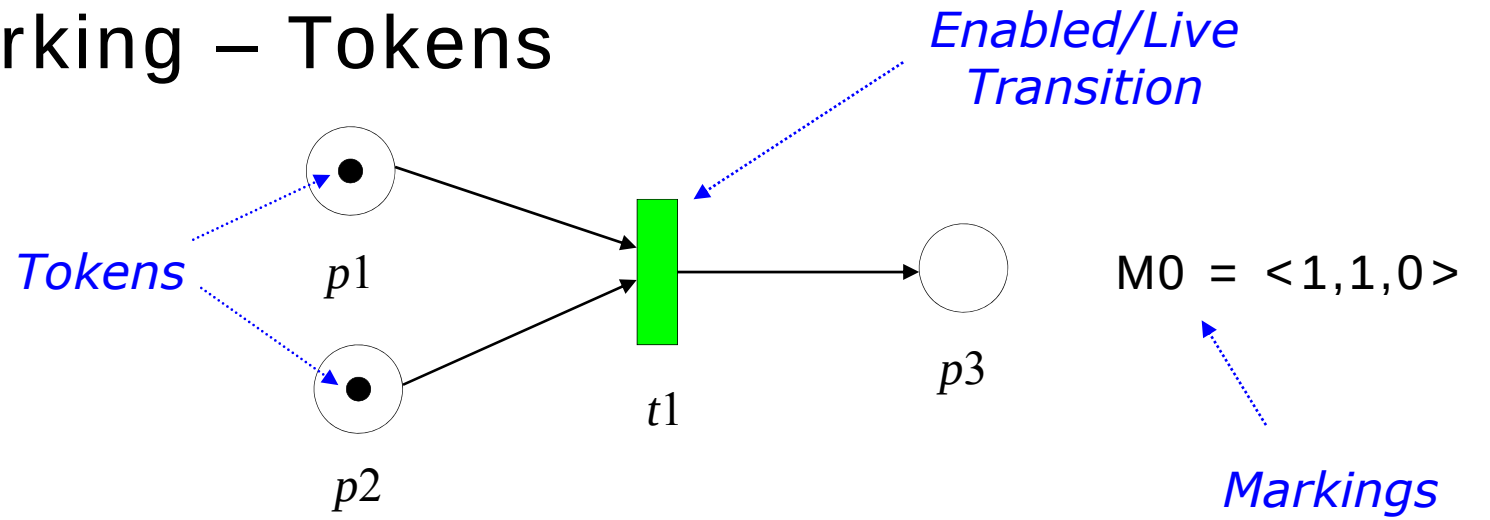
Petri Nets

- A Bipartite Directed Graph with two nodes set – places and transitions.

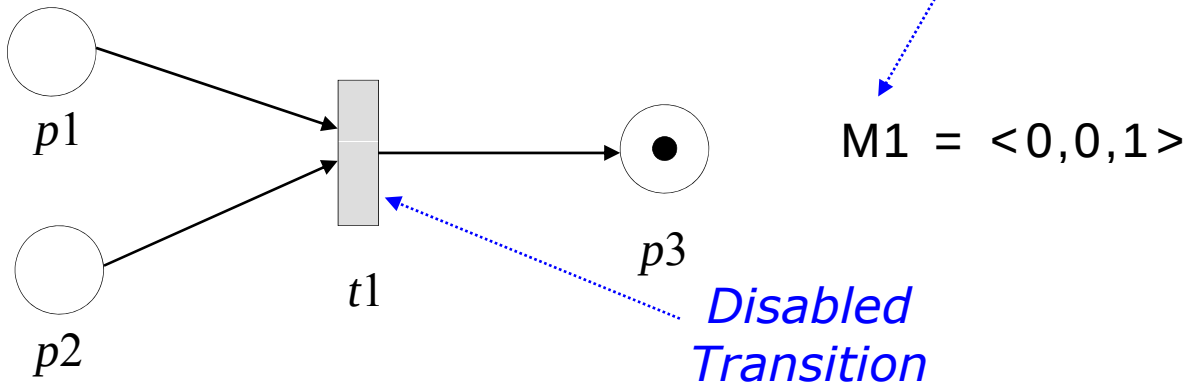
- Formally, a 4-tuple $(P, T, I-, I+)$ where,
  - P : *Places*
  - T : *Transitions*
  - $I- : P \times T \rightarrow N$ – *Backward incidence matrix*
  - $I+ : T \times P \rightarrow N$ – *Forward incidence matrix*

Execution

- Marking – Tokens



- Firing

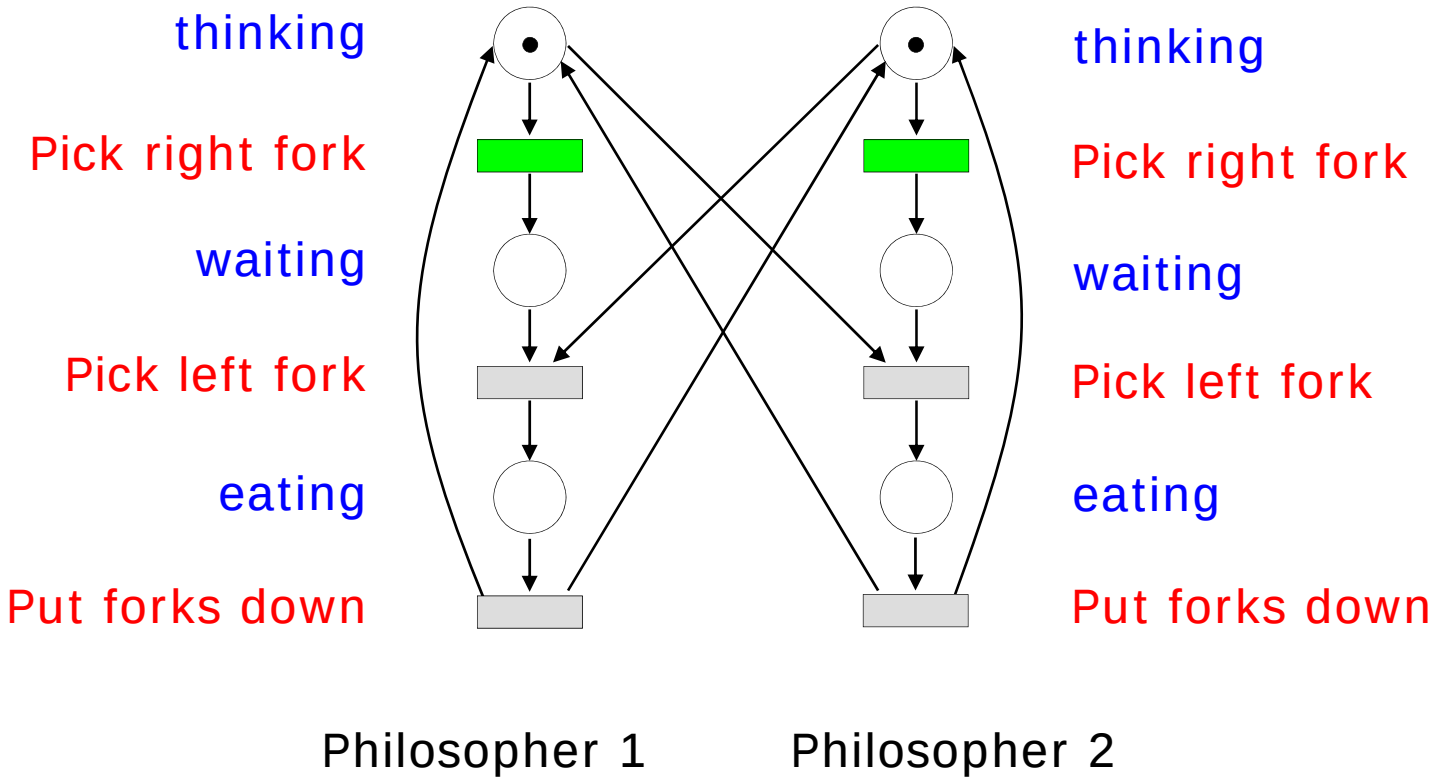


Goal Condition

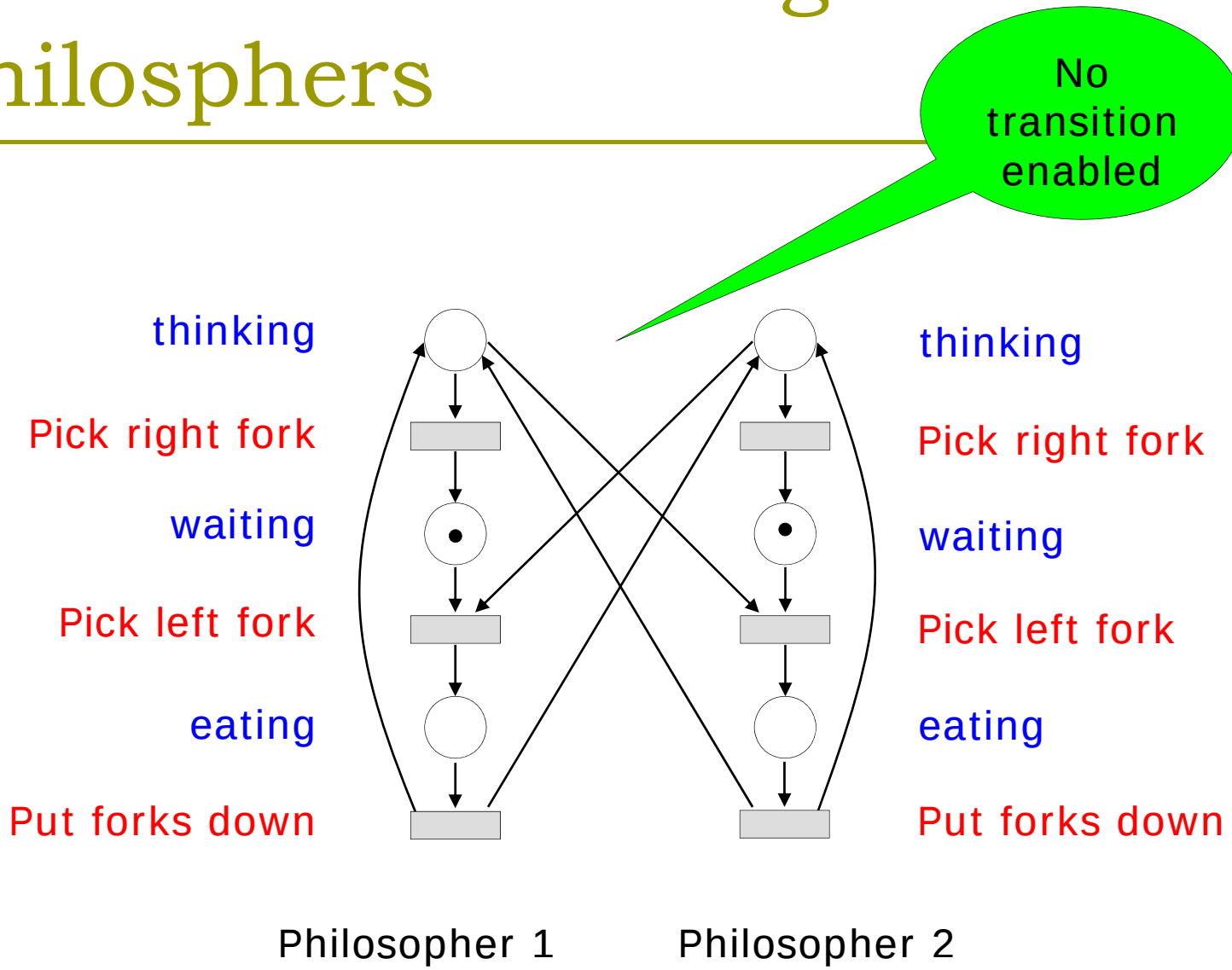
- Specific Goal Condition: An explicit marking.

- General Goal Condition: A set of different markings satisfying a particular property.
  E.g., A deadlock in the system – no transition is enabled.

Example: Dinning Philosphers



Deadlock in Dinning Philosphers



Distance Heuristics for Petri Nets – Basics

- Heuristics: Evaluation functions that estimate the number of transitions necessary to achieve a goal condition.
- Goal condition: ψ
- Shortest path between two markings M and M' is the minimum number of firings necessary to reach M' from M.
- Shortest path to the goal:

$$\delta(M,\psi) = \min\{\delta(M,M') \mid M' \models \psi\}$$

- Admissible: if $h(M) \leq \delta(M,\psi)$
- Monotone: if $h(M) - h(M') \leq 1$ for $M \rightarrow M'$

1. Hamming Distance Heuristics

$$h_H(M, M') = \sum_{p \in P} [M(p) \neq M'(p)]$$

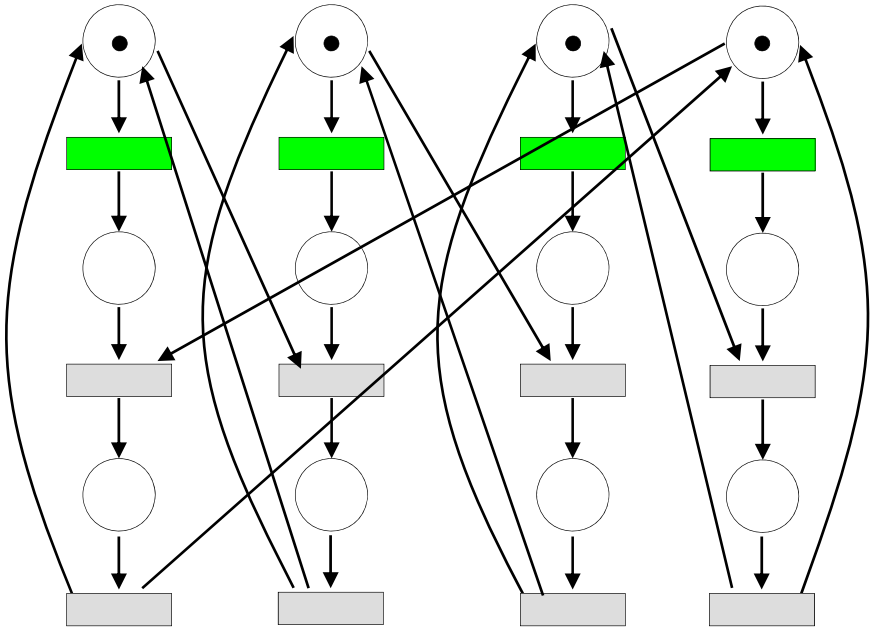
- where $[M(p) \neq M'(p)]$ evaluates to $\{0,1\}$

- ~~Admissible~~
- ~~Consistent~~

Since a transition can add/delete more than one tokens

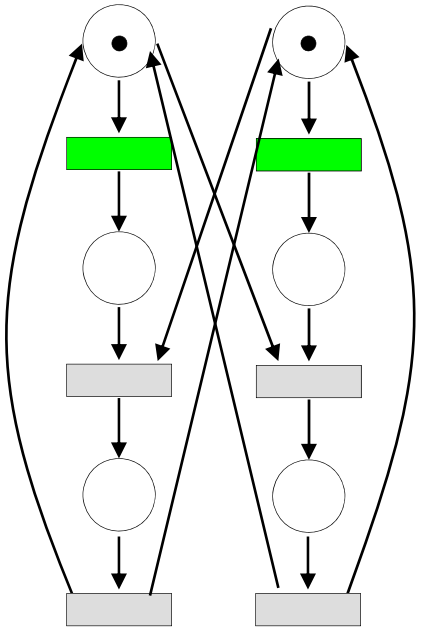
2. Subnet Distance Heuristics (Abstraction)

4-Philosophers



$\Rightarrow \varphi \Rightarrow$

2-Philosophers



- Admissible
- Consistent

Abstraction preserves triangular property

3. Activeness Heuristic

- Specialized heuristic for Deadlock detection.
- Deadlock => No enabled transition.
- Prioritize the marking that has the minimum number of diabled transitions.

$$h_A(M) = \sum_{t \in T} enabled(t)$$

- ~~Admissible~~
- ~~Consistent~~

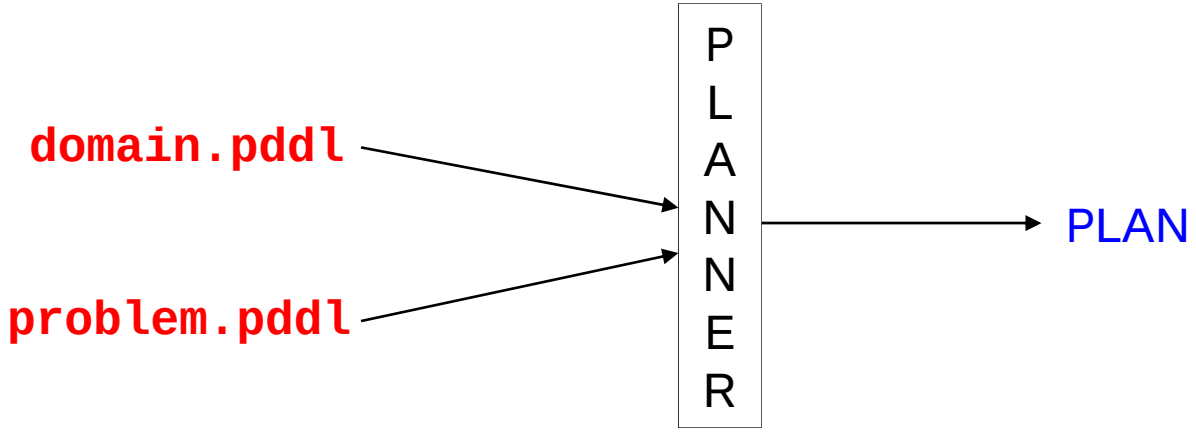
Since a single firing can effect the enableness of two or more transitions

Planning as Directed Model Checking

- **Motivation:** Can we utilize planning heuristics by modeling Petri nets analysis problem as a planning problem ?

PDDL Modeling for Petri Nets

- PDDL provides a modeling formalism for planning domains and problems.



- PDDL Level 2 allows numerical predicates => Necessary to model number of tokens in a Petri net

Modeling of Topology:

- Places :`(?p – place)`
- Transitions :`(?t – transition)`
- Incoming arcs to transitions
  - `(`**`incoming`** `?p – place  ?t – transition)`
- Outgoing arcs from transitions
  - `(`**`outgoing`** `?t – transition ?p – place )`
- Number of tokens
  - `(`**`number-of-tokens`** `?p – place)`

Modeling of Goal conditions

- Blocked Transition

```
(:derived block (?t - transition)
        (exists (?p - place)
                (and (incoming ?p ?t)
                        (= (number-of-tokens ?p) 0))))
```

- Deadlock:

```
(:derived deadlock
        (forall  (?t - transition)
                (blocked ?t)))
```

Propositional / ADL Encoding

- ADL provides a flexible planning formalism providing support for
  - Negation
  - Disjunctive preconditions
  - Conditional effects
  - Universal/existensial quantification of objects
- Transformation of Petri net model to ADL
  - Unary encoding of tokens `(?n – number)`
    - `zero`, `one`, `two`, `three`, .. Etc.
  - Predicates for their manipulations
    - `(`**`is-not-zero`** `?n – number)`
    - `(`**`inc`** `?n1 ?n2 – number)`

Propositional Planning Operator for Transition Firing

```
(:action fire-transition
 :parameters (?t - transition)
 :precondition
  (forall (?p - place)
     (or (not (incoming ?p ?t))
         (exists (?n - number)
                 (and (number-of-tokens ?p ?n) (is-not-zero ?n)))))
 :effect
  (and
   (forall (?p - place ?n1 ?n2 - number)
    (when
     (and (incoming ?p ?t) (inc ?n1 ?n2) (number-of-tokens ?p ?n2))
     (and (not (number-of-tokens ?p ?n2)) (number-of-tokens ?p ?n1))))
   (forall (?p - place ?n1 ?n2 - number)
    (when
     (and (outgoing ?t ?p) (inc ?n1 ?n2) (number-of-tokens ?p ?n1))
     (and (not (number-of-tokens ?p ?n1)) (number-of-tokens ?p ?n2)))))))
```

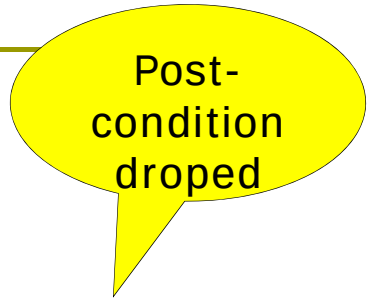
If all incoming places to t have tokens ?

Delete tokens from input places

Add tokens at output places

Planning Heuristic

- Action a = (pre(a), add(a), del(a))



- Relaxed action a^+ = (pre(a), add(a), Ø)

- *Heuristic* = length of the shortest plan that solves the relaxed problem.

Experiments

- Used FF Planner developed by Hoffmann.
- Relaxed Planning Heuristic.
- Extensive testing on deadlock checking benchmarks by Corbett.
- 1-safe Petri nets models.
  - A net is called 1-safe, if $M(p) \leq 1$ for all p
- Compared with the results by Heljanko and Niemelä on Bounded Model Checking.

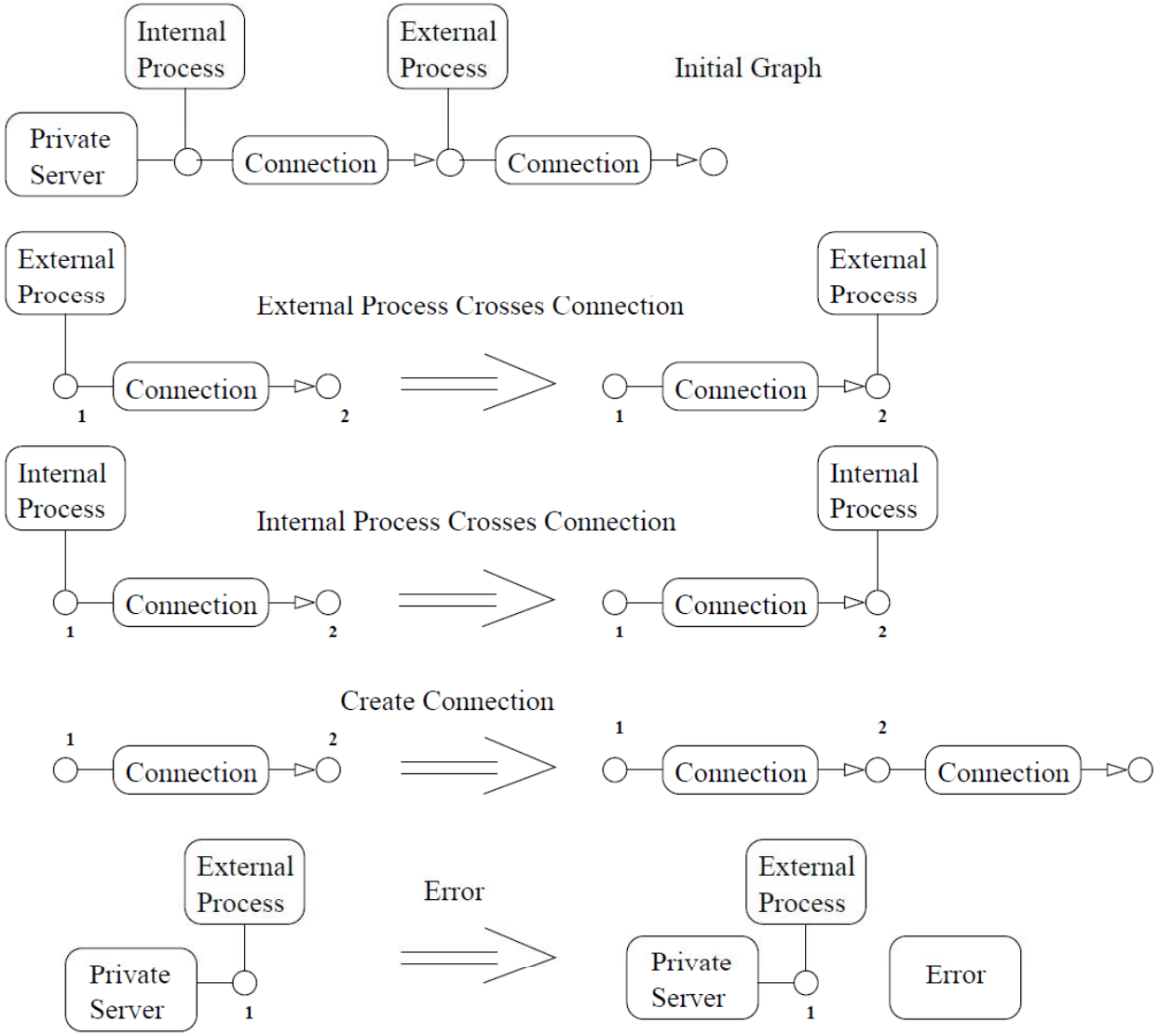
Experimental Results: Analysis of 1-safe petri nets with FF vs. Bounded Model Checking

				3.2 GHz		450 MHz
Prob.	P	T	Dep.	$Time_{FF}$	Expl.	$Time_{BMC}$
DARTES(1)	331	257	2	0.28	6	.5
DP(10)	60	40	10	0.08	19	3.3
DP(12)	72	48	12	0.08	23	617.4
ELEV(2)	146	299	16	0.2	74	3.9
ELEV(3)	327	783	18	2.08	106	139.0
HART(75)	377	227	76	0.71	77	15.5
HART(100)	502	302	101	1.45	102	45.9
Q(1)	163	194	21	0.25	258	2,733.7

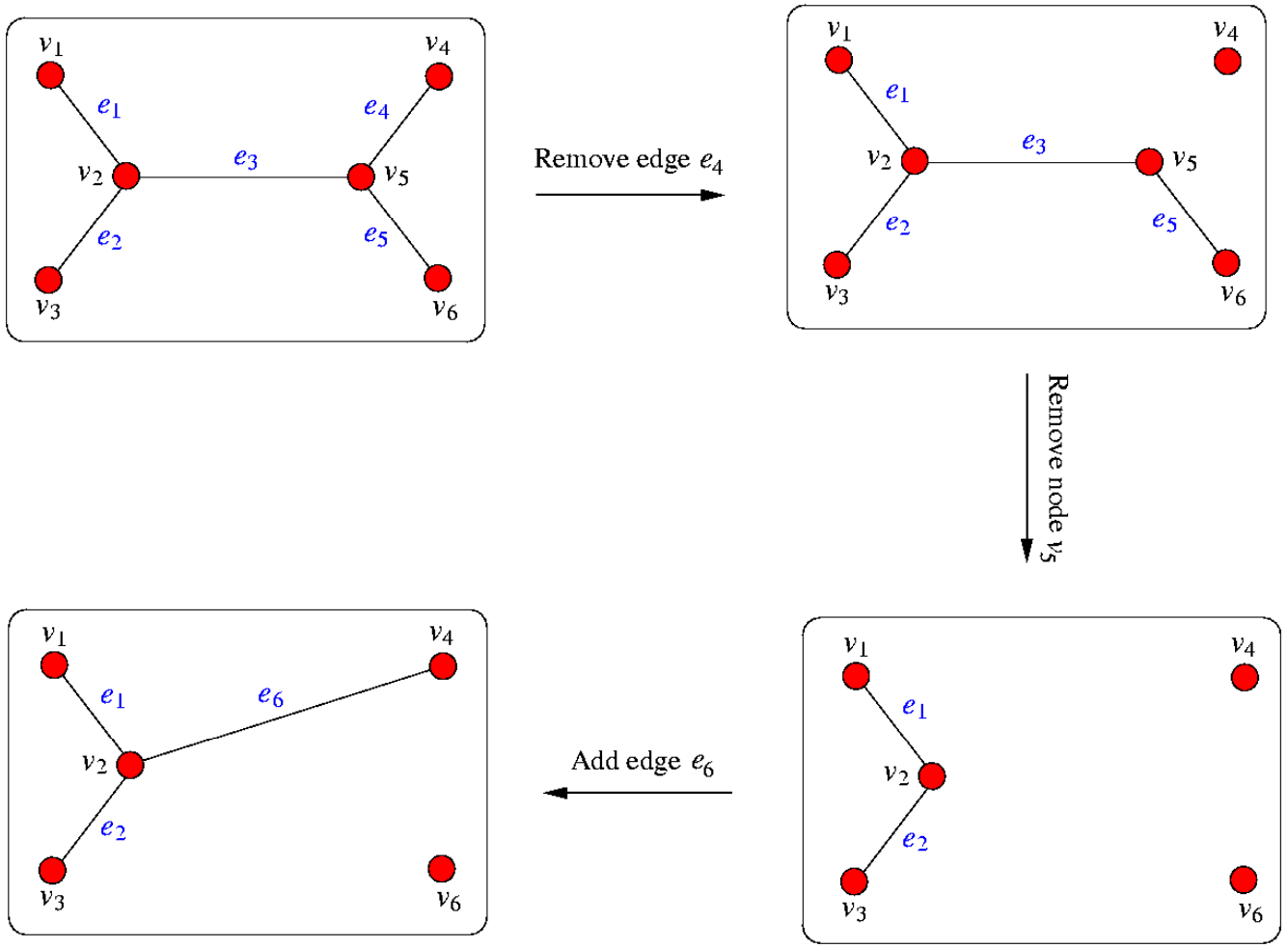
Wrap-Up

- Heuristics for analyzing Petri nets.
  - Hamming distance, abstraction, activeness.
- Modeling of a model checking problem as a planning problem.
  - Enable us to utilize planning heuristics for analysis of Petri nets.
- Experimental results show the potential of the approach.
- Can incorporate more complex goal conditions like assertions.
  - `(<= (number-of-tokens ?p) ?m)`

Graph Transformation System



Usual Operations



Directory Service Protocol

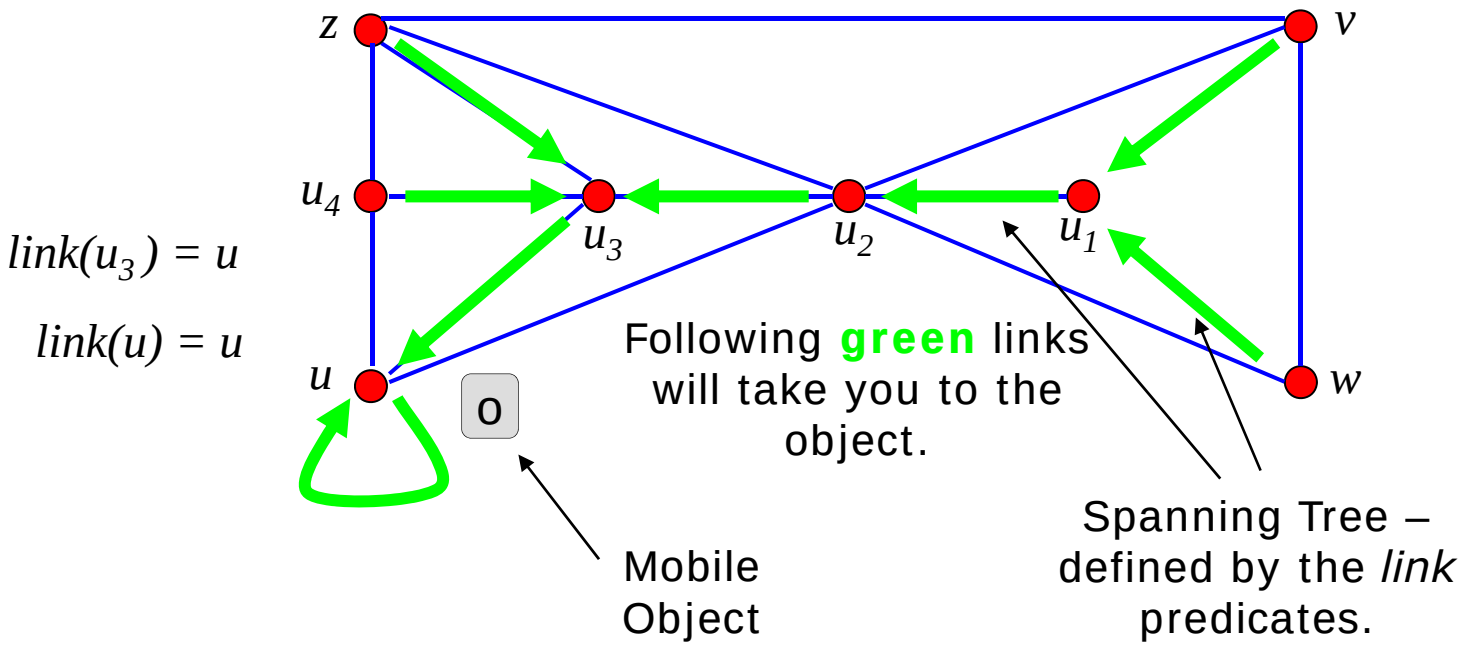
- Assume a distributed environment.
- **Clients:** The nodes in the distributed network e.g., different computers.
- **Mobile Objects:**
  - Could be a file, a process or any other data structure.
  - It can be transmitted over a network from one node to another.
  - It “lives” only on one node at a time.
- **Purpose of a Directory Service:**
  - Navigation: To provide the ability to locate a mobile object.
  - Synchronization: To ensure mutual exclusion in the presence of concurrent requests.

Usual Approach

- “home”-based structure.
- Each object has its own “home”.
- “home” keeps track of the object’s location.
- All requests are send to the “home”.
- “home” sends a message to the client currently holding the object.
- That client forwards the object to the requesting client.
- Bottleneck: Communication costs between “home” and clients.

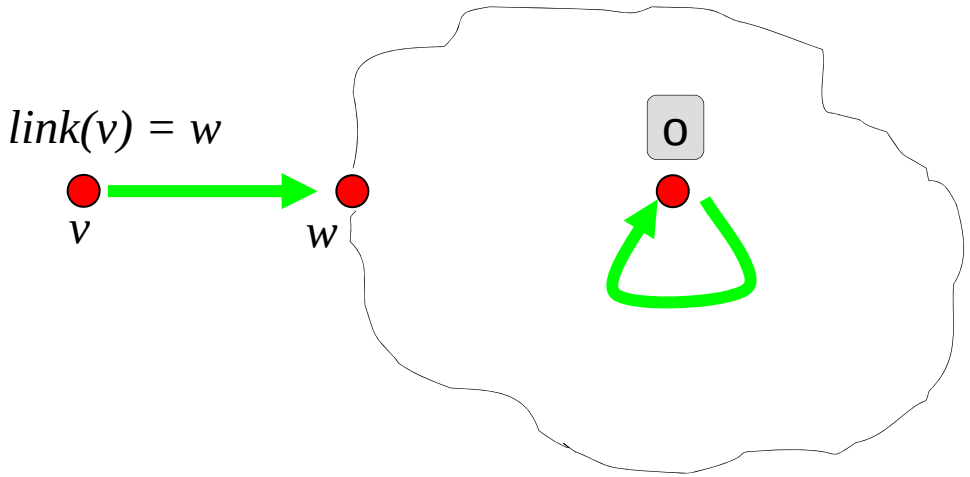
The Arrow Distributed Directory Protocol (Demmer and Herlihy)

- Based on the idea of a trail of pointers
- Distributed Network G = (V,E,w)



Properties of the Protocol

- **If** *link(v)* = *v* (self-loop) => The object either resides at *v*, or will soon reside at *v*.
- **Else**, the object resides some where in the region of the directory containing *link(v)*.

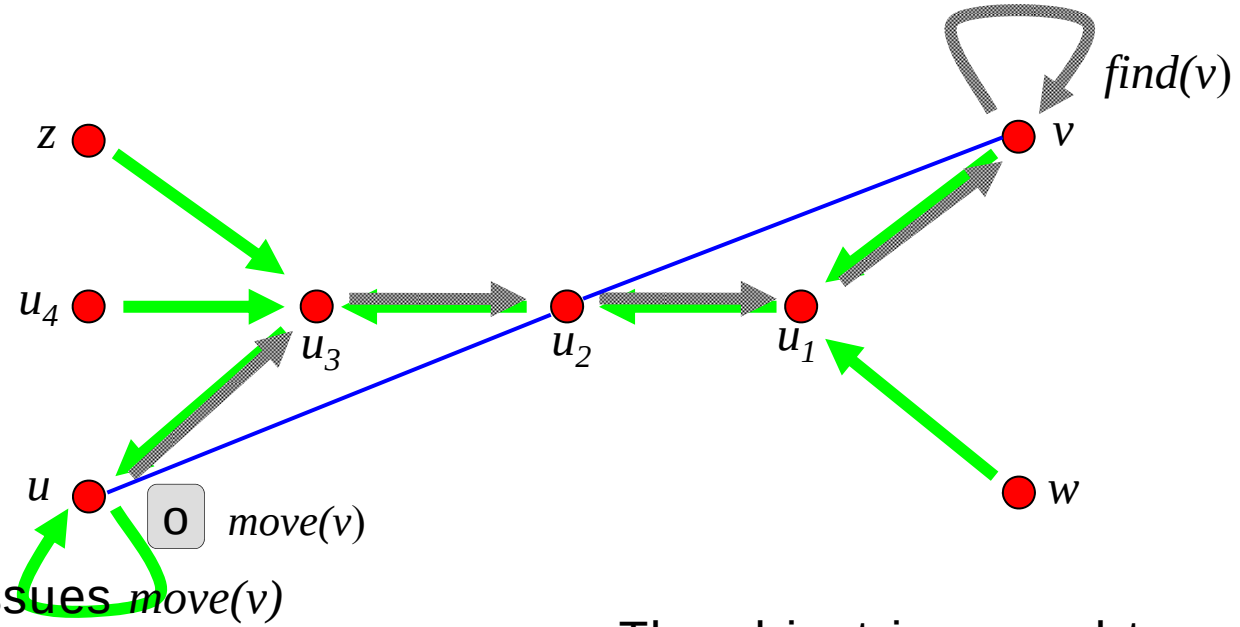


Messages and Constructs

- *link(u,v):* Defines the spanning tree.
- *find(v):* Request for the object issued by the node v.
- *move(v):* The object is free to be moved to v. It travels with the object, following the links in the original graph.
- *pending(u,v):* Every *link(u,v)* has a buffer that keeps the request. Not a FIFO, but reliable.
- *queue(u) = {v, NULL}:* A predicate attached with every node. Tells that u has to transfer the object to v when it is finished with the object.

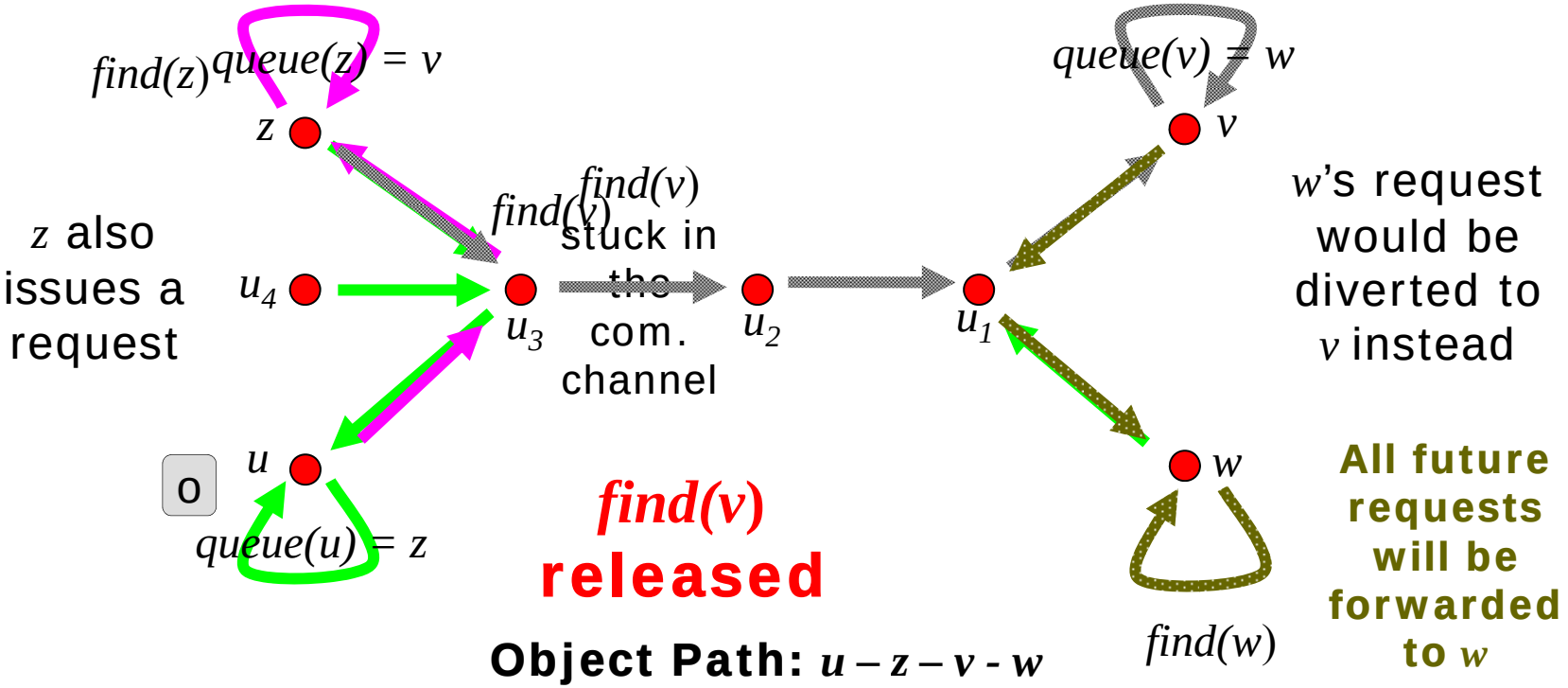
Working of the Protocol

- *v* issues a request *find(v)* for the object.



Concurrent Requests

- *find(v)* stuck in the communication channel.
- *w* also issues a request in the meanwhile.



Advantages

- A distributed queue structure.
- Object request messages travel the shortest path in the spanning tree and not in the original graph.
- The queue structure ensures locality: all requests will go directly to the object or to another terminal. Do not have to pass through a “home”.

Properties to Verify / Types of Goals

- Can a particular node u be a terminal? (Subgraph matching)
- Can a particular node u be a terminal and all arrow paths end at u? (Graph Matching)
- Can an arbitrary node u_i be a terminal? (Subgraph isomorphism)
- Can an arbitrary node u_i be a terminal and all arrow paths end at u_i? (Graph isomorphism)

PDDL: Morphism as Actions

- A morphism operation that inverses an edge can easily be defined as a very simple action.
-
```
(:action morphism-inverse
    :parameters(?u ?v - node)
    :precondition
        (link ?u ?v)
    :effect
        (and
            (not (link ?u ?v))
            (link ?v ?u)))
```

PDDL Encoding of Goals: Graph and Subgraph Matching

- Subgraph and graph matching are easy to encode.
- Encode the goal graph with `(link u v)`
- and owner with `(owner w)` predicates.

PDDL Encoding of Goals: Subgraph Isomorphism

- Goals are strictly more expressive.
- Need an existential quantification over all the nodes to be described.
- ADL (Pednault 1989) ➔
  - `(:goal  <existential-expression>`
    `<goal-condition>)`
- Using ADL, subgraph isomorphism can be encoded as
  - `(:goal (exists (?n - node) (owner ?n)))`

PDDL Encoding of Goals: Graph Isomorphism

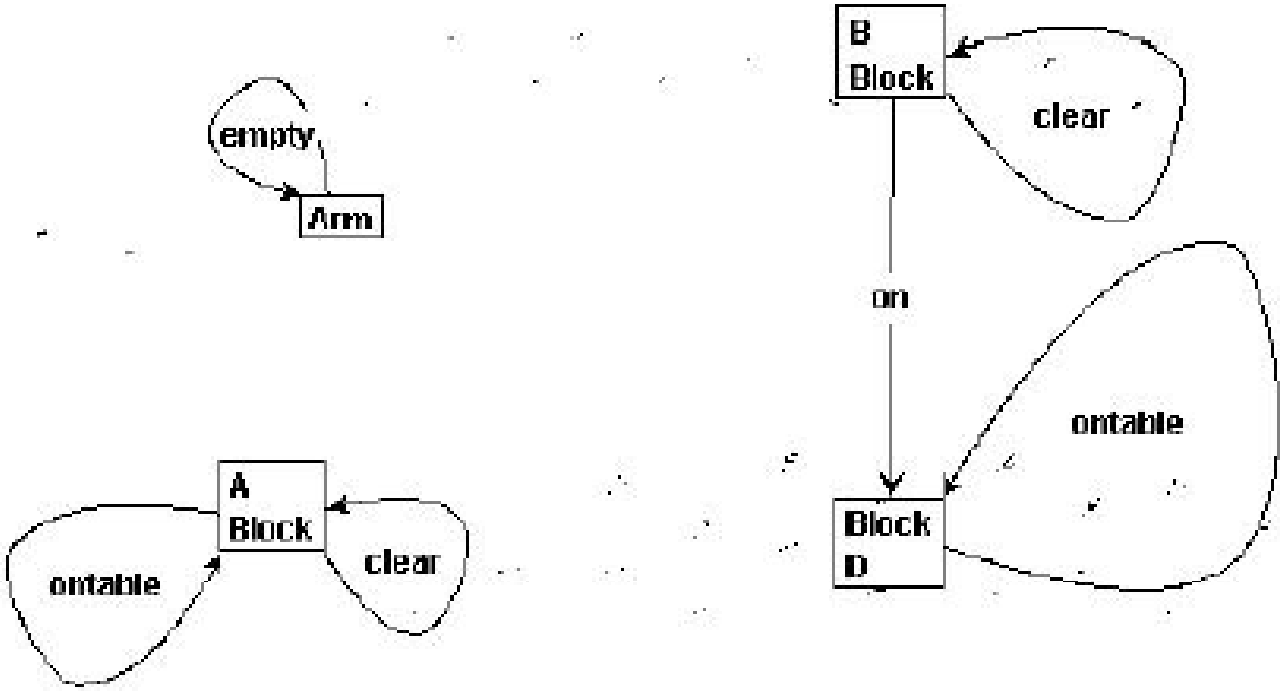
- Existential quantifier can again be used ..

```
(:goal (exists ?v0 ?v1 ?v2 ?v3 ?v4
                ?v5 - node)
  (and (link ?v0 ?v0)  (link ?v1 ?v0)
       (link ?v2 ?v0)  (link ?v3 ?v1)
       (link ?v4 ?v0)  (link ?v5 ?v4)
       (owner ?v3)))
```

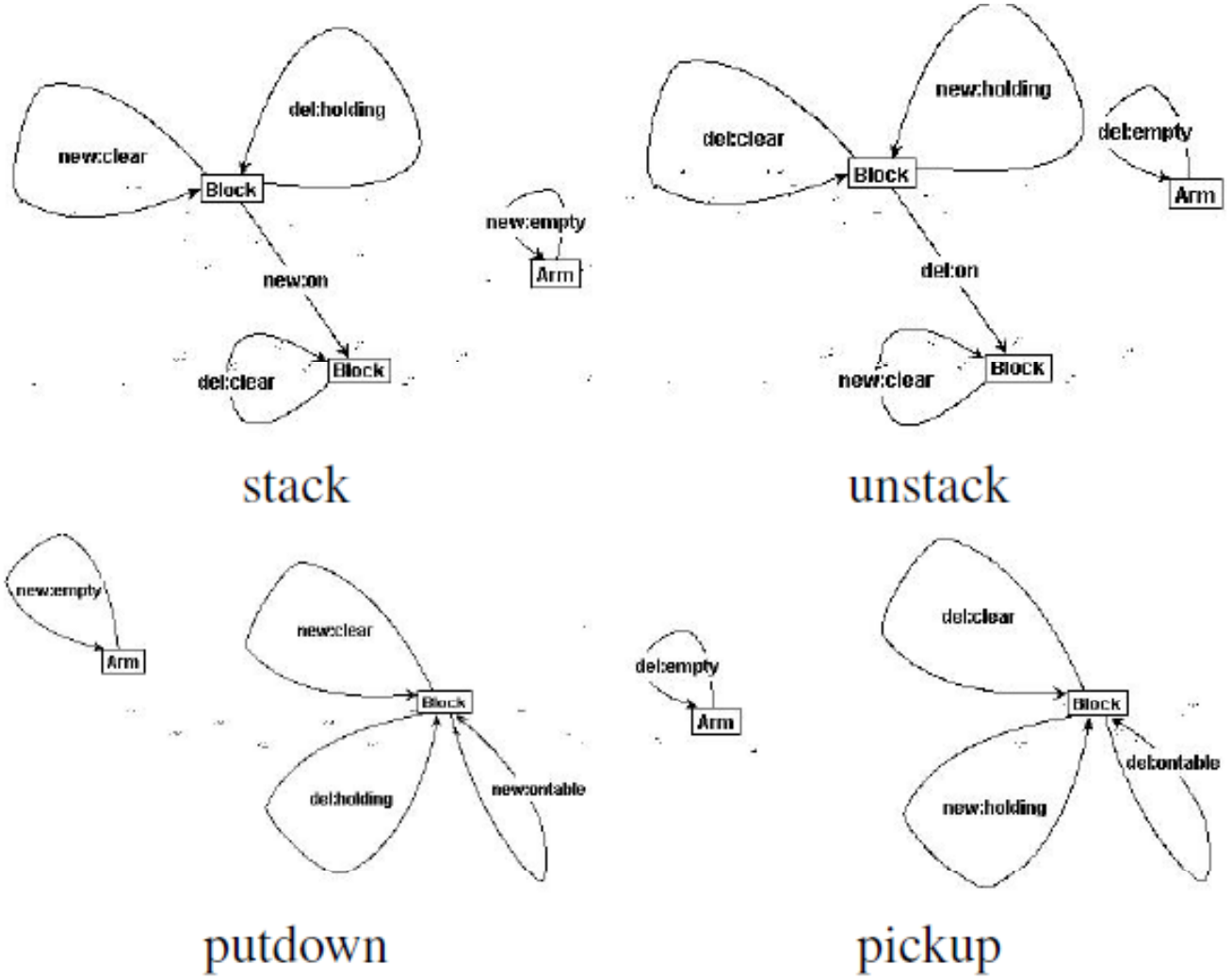
Performance: Model Checker vs. Planner– Subgraph Matching

Star	DFS	BFS + h_f	EHC + RPH
Stored nodes	6,253	30	6
Sol. length	134	58	5
Chain	DFS	BFS + h_f	EHC + RPH
Stored nodes	78,112	38	6
Sol. length	118	74	5
Tree	DFS	BFS + h_f	EHC + RPH
Stored nodes	24,875	34	6
Sol. length	126	66	5

Graph Transformation & Planning [E., Rensink ICKEPS-07-WS]



Graph Transformation & Planning [E., Rensink ICKEPS-07-WS]



GRaphs for Object-Oriented VErification

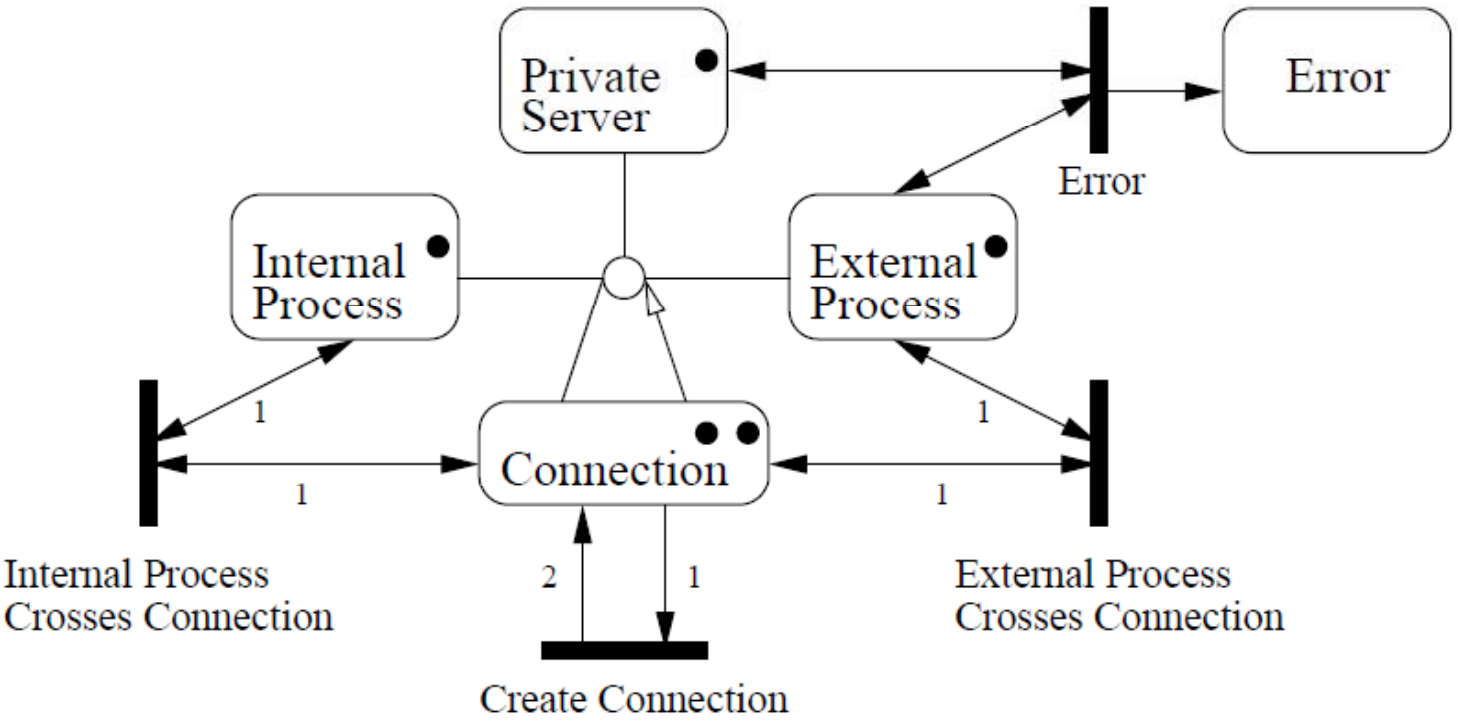
The GROOVE tool set includes an editor for creating graph production rules, a simulator for visually computing the graph transformations induced by a set of graph production rules, a generator for automatically exploring state spaces, and an imaging tool for converting graphs to images.

Arend Rensink, University of Twente
http://groove.cs.utwente.nl/groove-home/

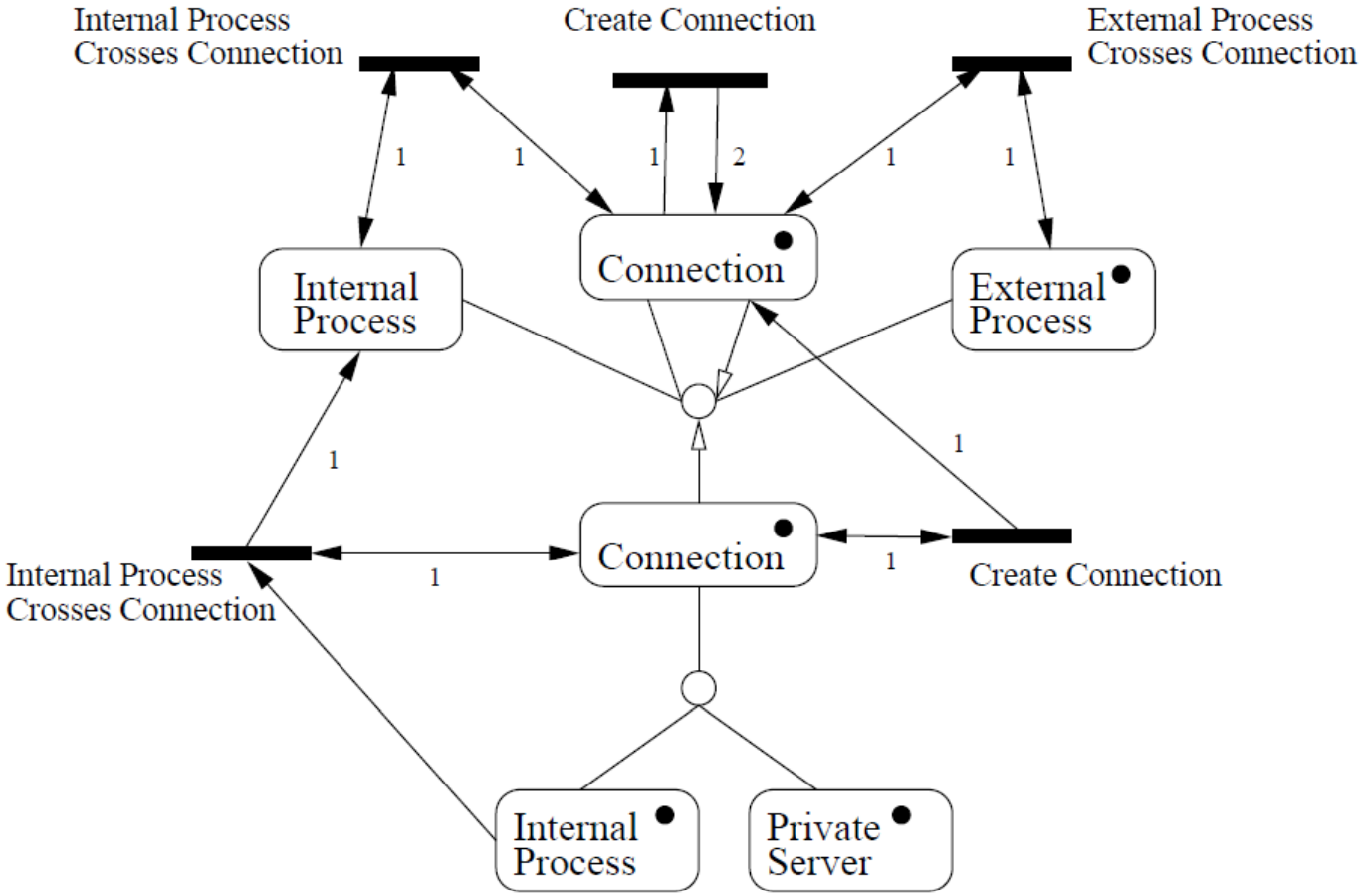
AUGUR (B. König et al.)

- Tool for the verification of systems described by (attributed) graph transformations using approximated unfoldings.
- The obtained over-approximation consists of an underlying hypergraph and an Petri net.
- Properties of graph transformation systems can be verified by analyzing the approximation, using regular expressions, first order logic and coverability checking techniques for Petri nets.

Petri Net Approximation



After Abstraction Refinement



Graph Transformation via Planning for Petri Nets

- Zaks (2007) showed that finding counterexample in Petri-Net approximations within a graph transformation refinement loop is seemingly faster when exporting the domain to PDDL and use a from-shelf action planner
- Analysis Algorithms for Petri Nets

[S. Turan, Bachelor Thesis, Stuttgart 2004]

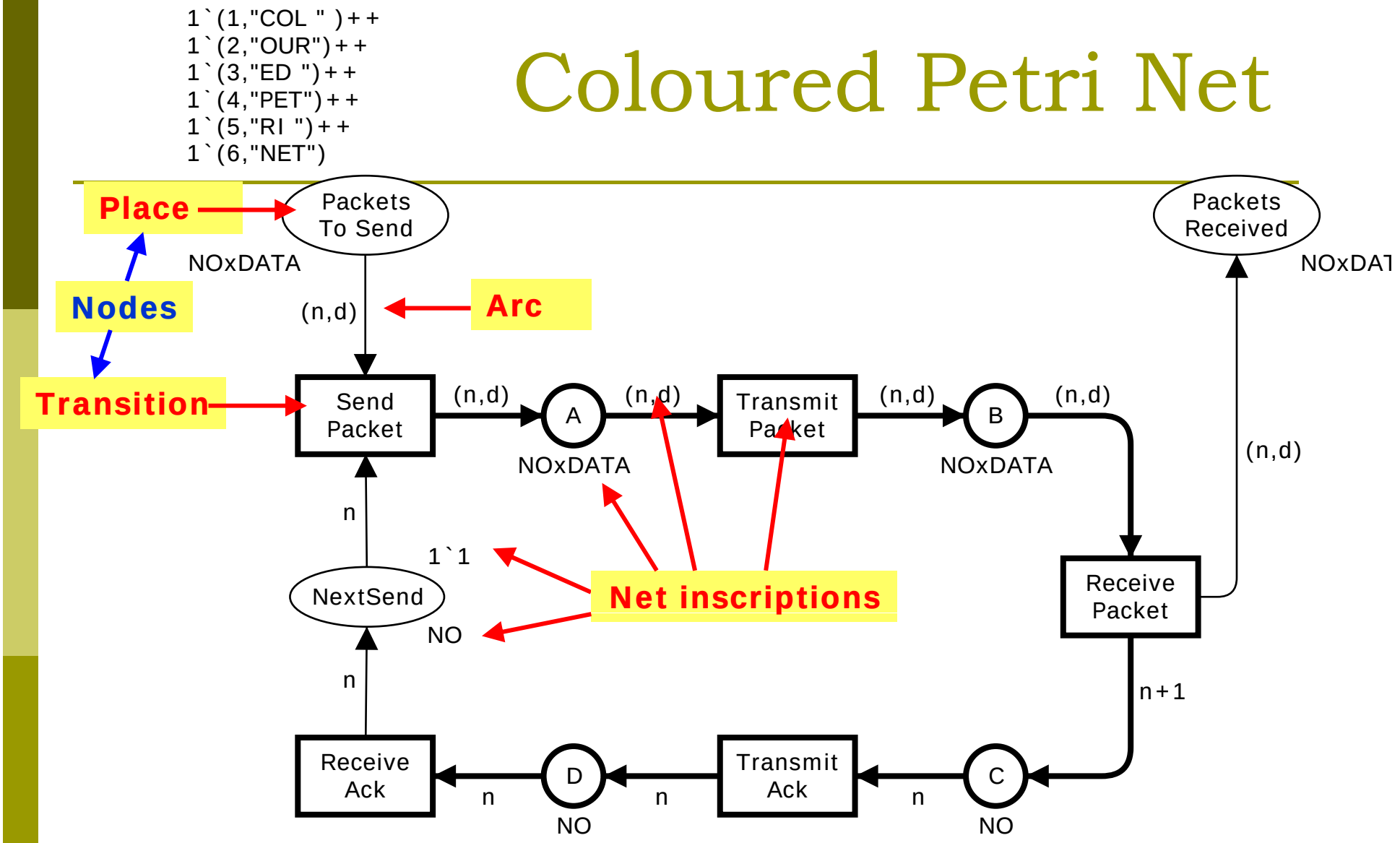
- Optionally one can use MetricFF for checking coverability of non-attributed PNs

Results

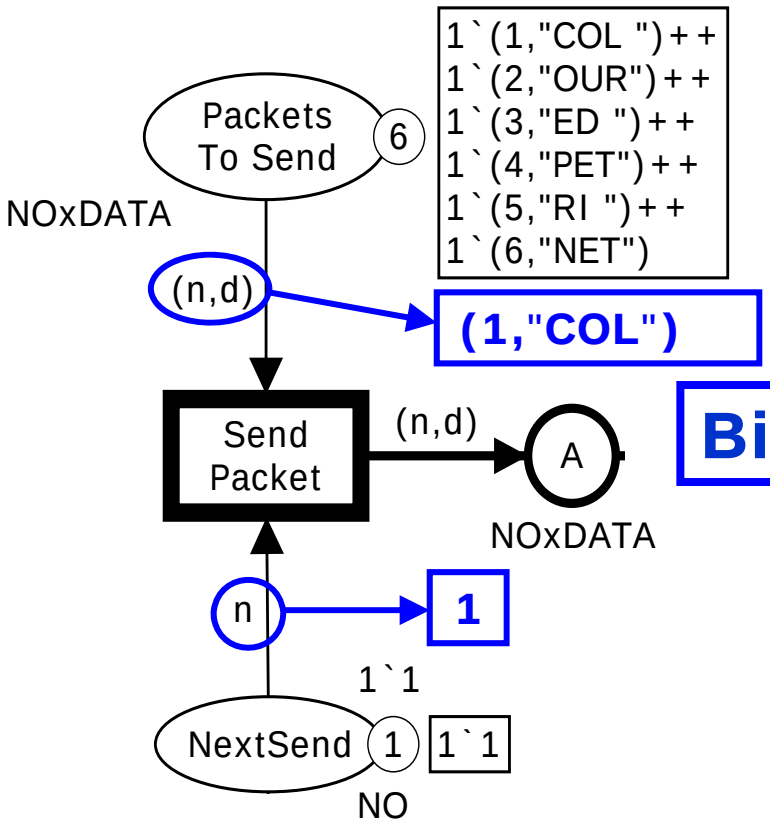
- MetricFF vs. Petri Net analyzer BWRA.

Problem	***BWRA***	***MetricFF***	***Error***
red-black	947s	1s	yes
red-black	948s	1s	yes
firewall2	7s	1s	yes
firewall2	338s	1s	yes
firewall2	6s	1s	yes
server2	1s	–	no
server2	6s	–	no
server2	545s	1s	yes

Coloured Petri Net



Enable Transition

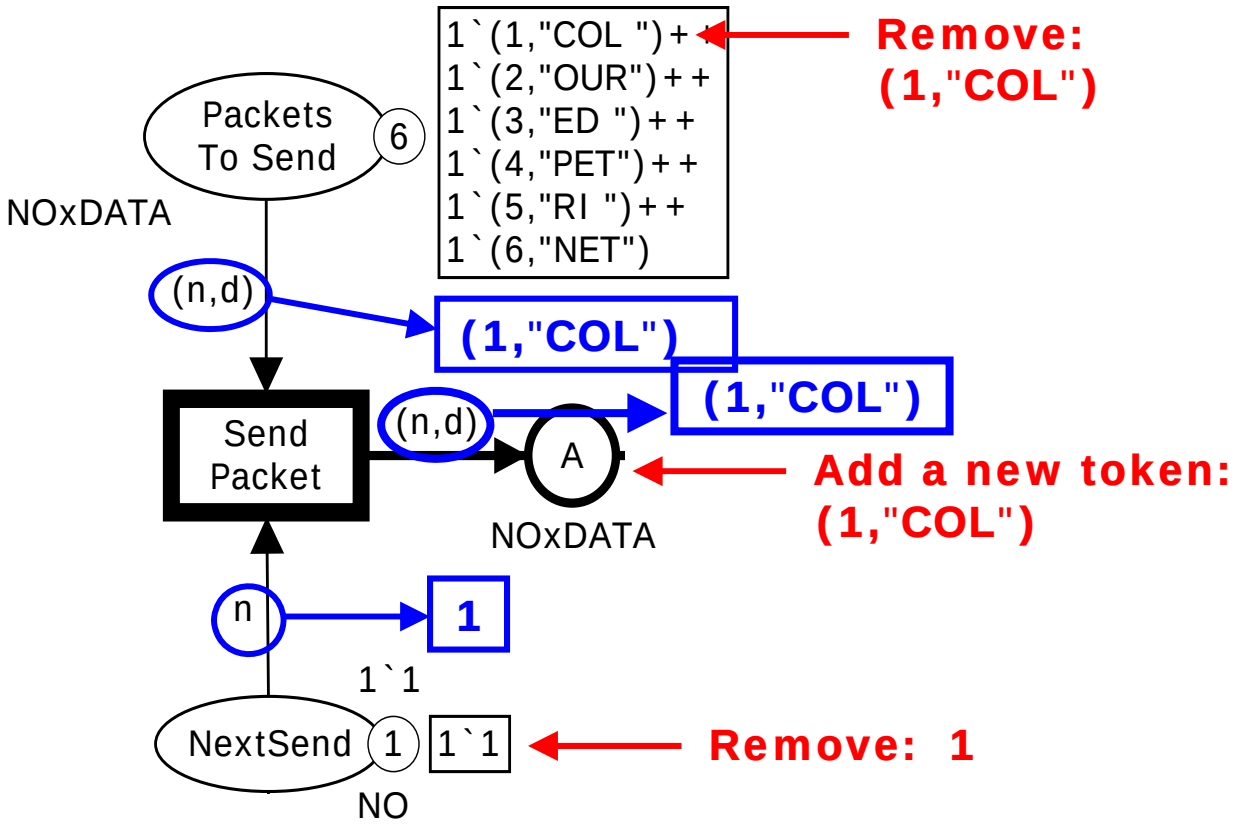


We have found a binding so that each input arc expression evaluates to a colour that is present on the corresponding input place

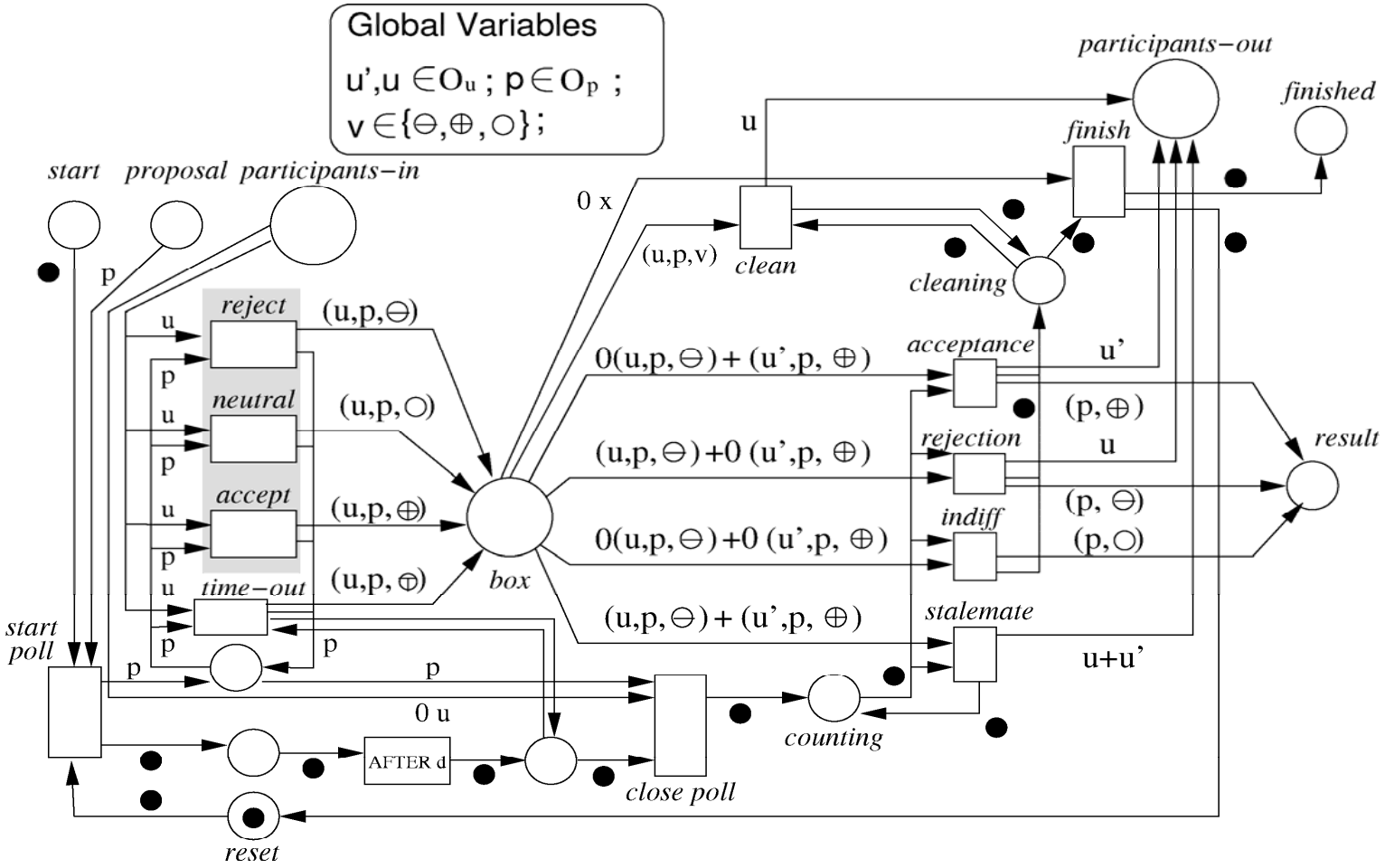
Binding: < n = 1 , d = "COL" >

Transition is enabled (ready to occur)

Fire Transition



Propositional and Administration Nets



Propositions

(incoming ?p - place ?t - transition ?c - class)

denoting input arcs connecting places with transitions and the class that is associated with it.

(incoming-inh ?p - place ?t - transition ?c - class)

denoting inhibitor arcs and the class that is associated with it.

(outgoing ?t - transition ?p - place ?c - class)

denoting output arcs and the class that is associated with it.

(of-type ?m - marking ?c - class)

denoting to which class a token element belongs to.

(at-place ?p - place ?m - marking)

denoting the current place of a token.

(selected ?p - place ?c - class ?m - marking)

denoting if a token of a certain class at a certain position is selected for firing.

Planning for Extended PN

```
(:action fire-transition
:parameters (?t - transition)
:precondition
(forall (?p - place)
(forall (?c - class)
(and (or (not (incoming-inh
   ?p ?t ?c))
(forall (?m2 - marking)
(or (not (of-type ?m2 ?c))
(not (at-place ?p ?m2)))))
(or (not (incoming ?p ?t
   ?c))
(exists (?m1 - marking)
(and (selected ?p ?c ?m1)
(of-type ?m1 ?c)
(at-place ?p ?m1)))))))
:effect
(forall (?c - class)
(forall (?pin - place)
(forall (?pout - place)
(forall (?m - marking)
(when
(and (selected ?pin ?c ?m)
(incoming ?pin ?t ?c)
(outgoing ?t ?pout ?c))
(and (not (selected ?pin ?c
   ?m))
(not (at-place ?pin ?m))
(at-place ?pout ?m)))))))
```