DSM-TP 2010 | Domain Specific Modeling Theory and Practice

# (Modelling) Semantics of Modelling Languages


**Hans Vangheluwe**

7 September 2010, Lisboa, Portugal

## Overview

1. Building DS(V)M Tools Effectively
   1. Specifying **syntax** of DS(V)Ls:
      - **abstract** (**meta-modelling**)
      - **concrete** (textual–visual)
   2. Specifying DS(V)L **semantics**: **transformations**
   3. Modelling (and executing) **transformations**: **(rule-based) transformation languages**
2. Delving into Semantics
3. DSL examples with a focus on semantics

## Syntax, **Semantics**, and all that Stuff

David Harel, Bernhard Rumpe.
*Meaningful Modeling: What's the Semantics of "Semantics"?*
IEEE Computer, vol. 37, no. 10, pp. 64-72, October, 2004.

## Information and its syntactic representation as data

- **data** is used to communicate
- understanding the **information** encoded in the data
  interpretation:
  a **mapping** that assigns a **meaning** to each (legal) piece of data

## Two pieces of data may encode the **same** information

- “June 20, 2000”
- “The last day of the first spring in the second millennium”

## Information and its syntactic representation as data

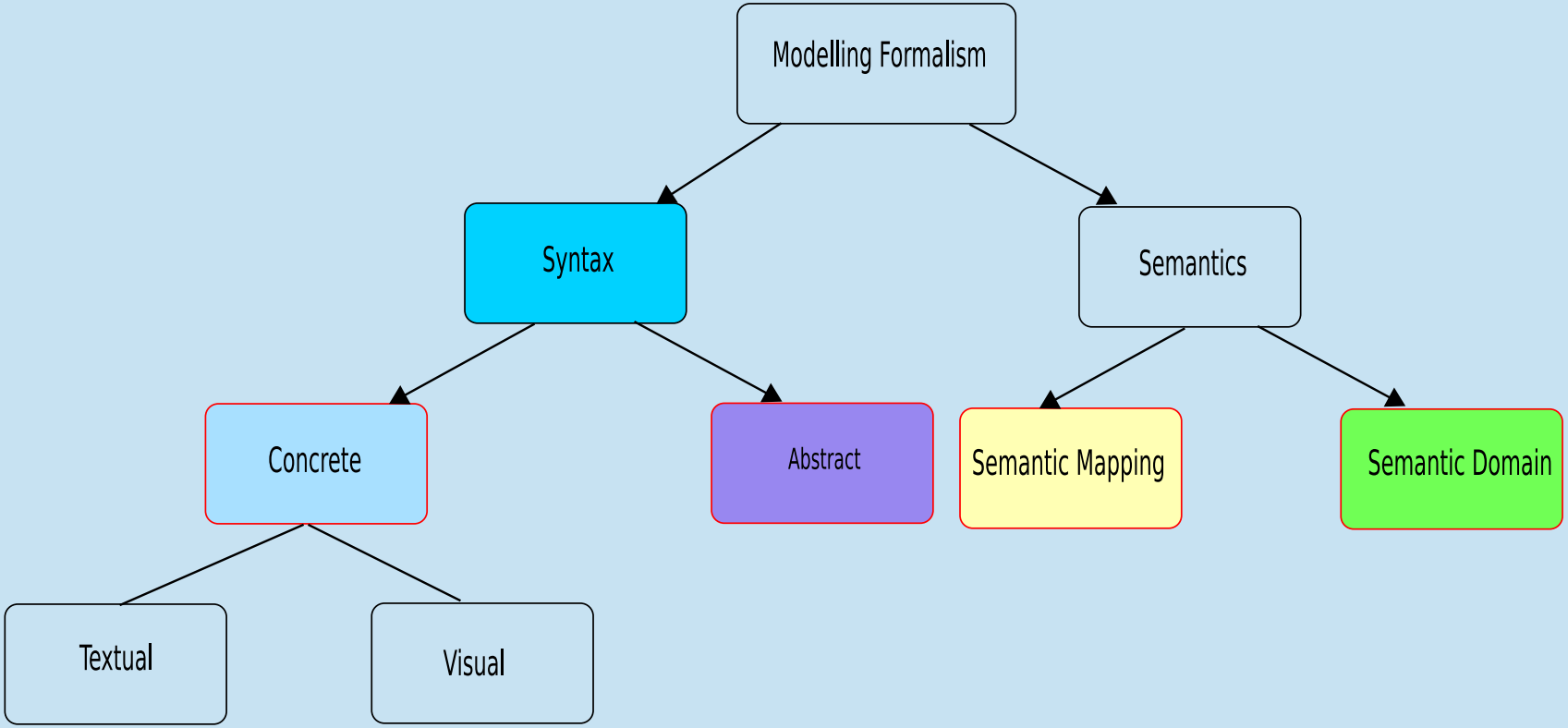
Same piece of data may have several meanings
i.e., denote **different** information for different people or applications

- "John's birthday" is **context-dependent**

## Different languages for different "users"

- people use natural languages to communicate
- machines use machine-readable languages for communication
- people use programming/modelling languages to communicate with machines

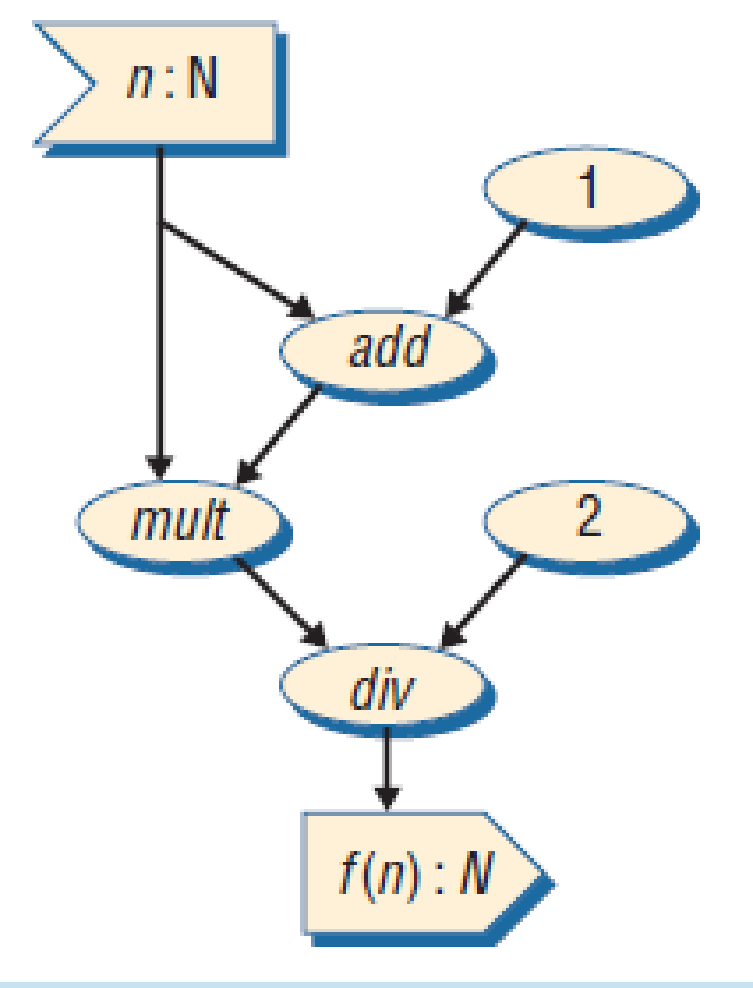
# Dissecting a Modelling Language (tool builder's view)



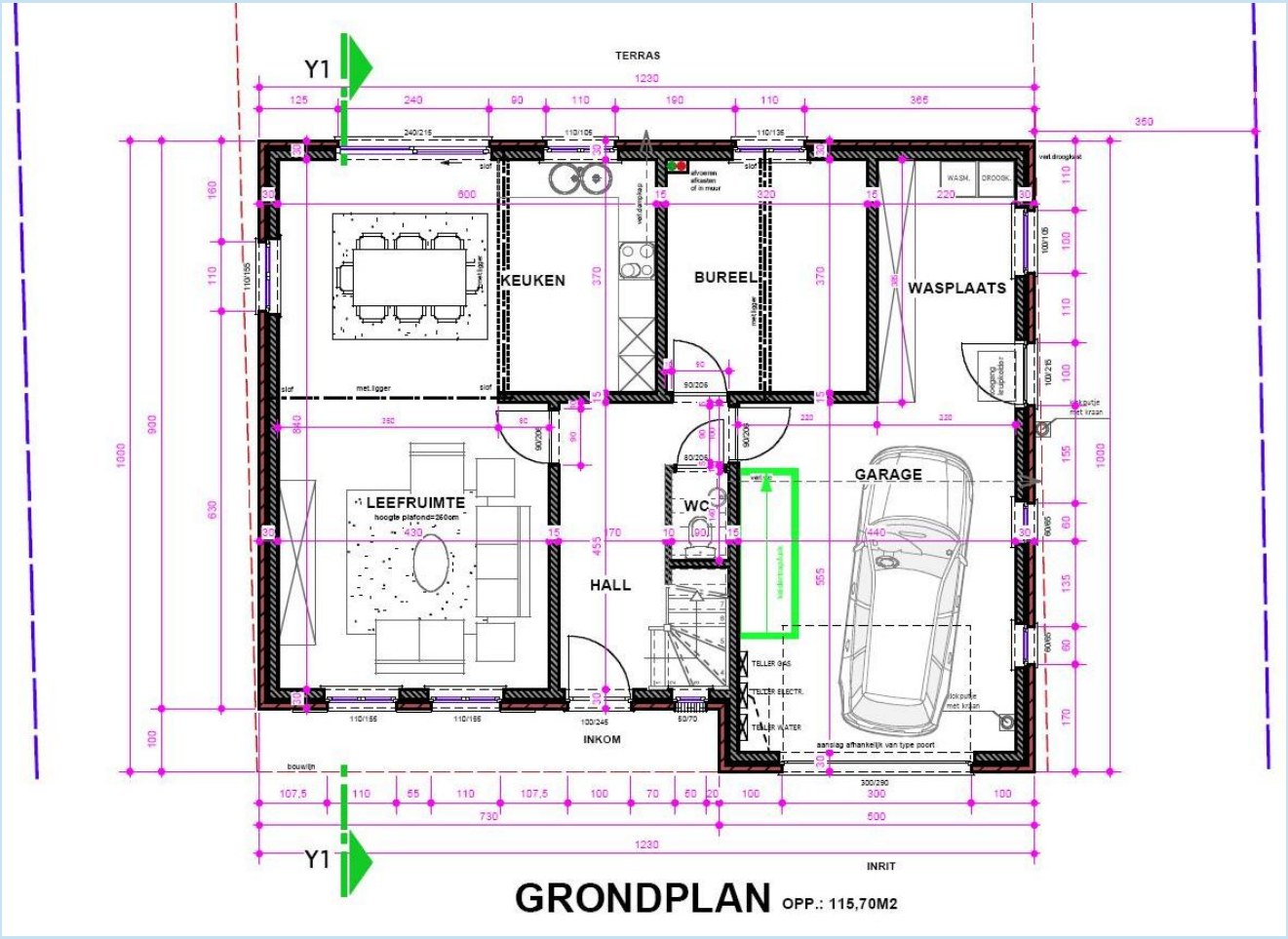
## Concrete Textual Syntax

```
define f(n: NaturalNumber): NaturalNumber
{
 return n*(n+1) DIV 2;
}
```

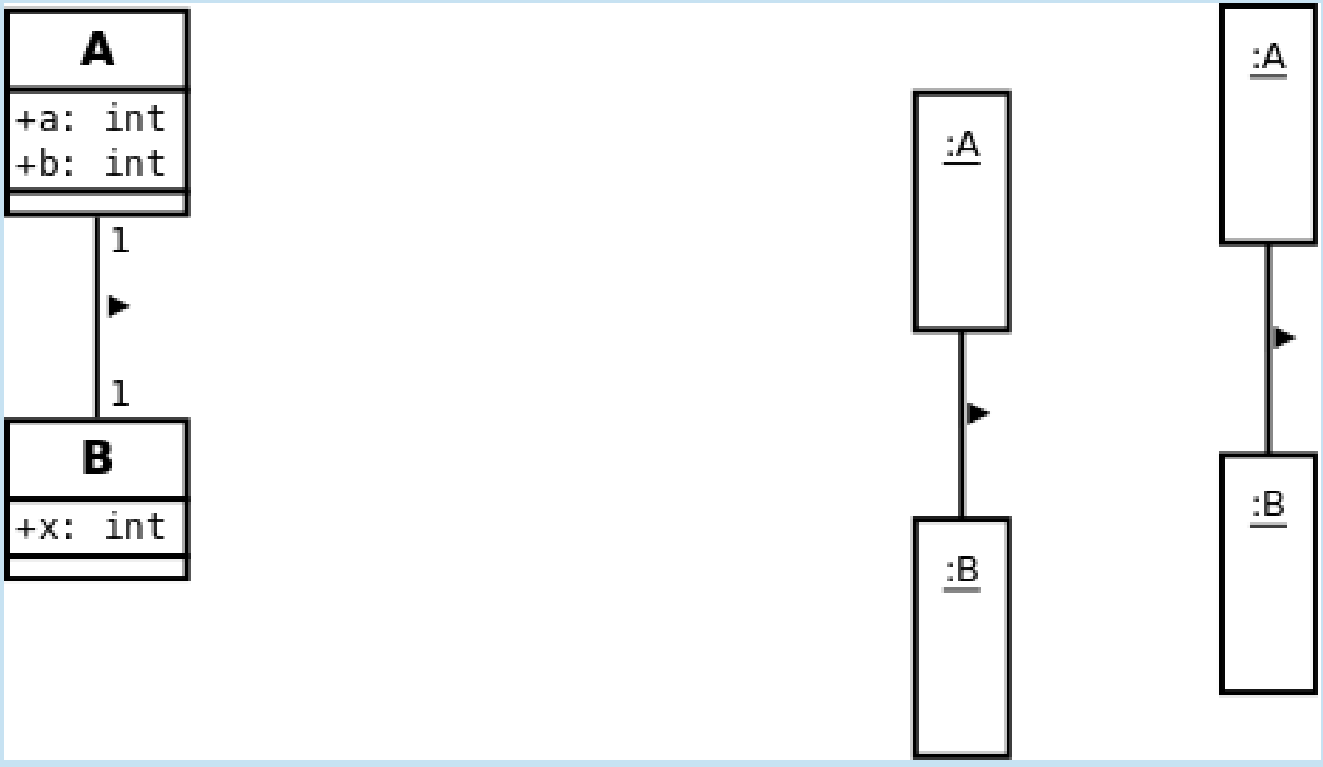
# Concrete Visual Syntax



# Semantics: not necessarily Behavioural, can be Structural



# Semantic Domain: Structural



## Sentence/Model in Language

```
define f(n: NaturalNumber): NaturalNumber
{
 return n*(n+1) DIV 2;
}
```

## Semantic Domain: Arithmetic expressions

A BNF-like **grammar** describes the abstract syntax for simplified arithmetic expressions.

$$
\begin{array}{lcl}
\langle Exp \rangle & ::= & \langle Number \rangle \quad | \quad \langle Variable \rangle \\
& | & \texttt{(}\ \langle Exp \rangle\ \texttt{)} \quad | \quad -\ \langle Exp \rangle \\
& | & \langle Exp \rangle\ [\ + \ | \ - \ | \ * \ | \ /\ ]\ \langle Exp \rangle \\
& | & \texttt{foo (}\ \langle Exp \rangle\ \texttt{,}\ \langle Exp \rangle\ \texttt{)}
\end{array}
$$

## Semantic Domain: Arithmetic expressions

for semantic domain S we choose all natural numbers

$$S < Exp >= Nat$$

and the semantic mapping M associates a number with each expression:

$$M < Exp >:< Exp > \rightarrow Nat$$

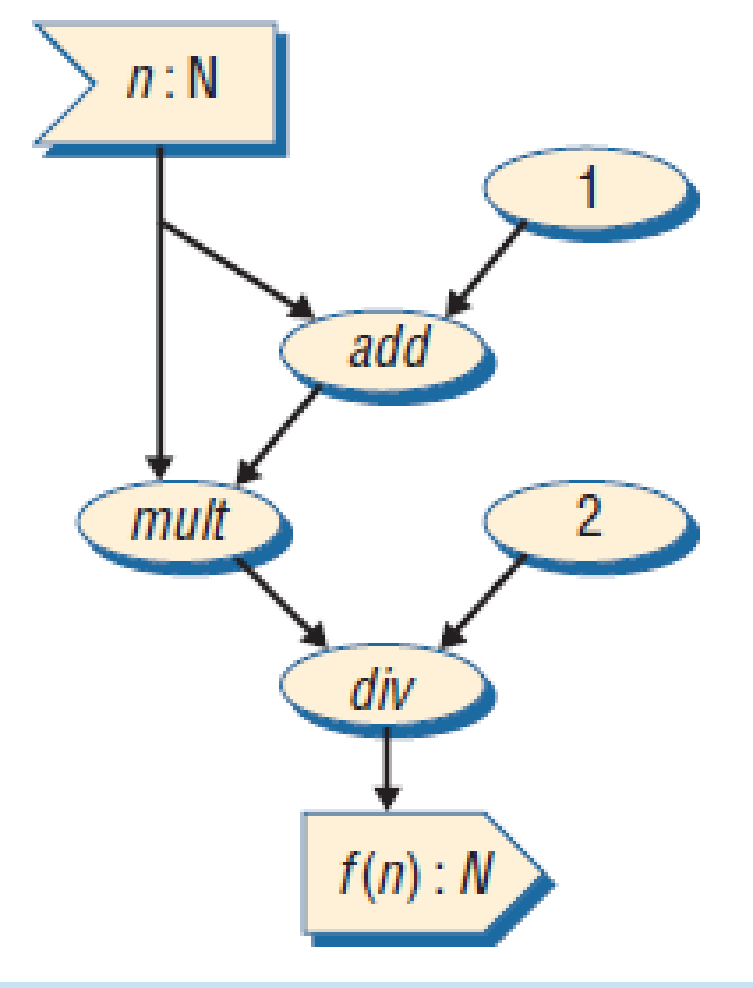
standard mathematics is a natural notation for describing the mapping.

$$M(''42'') = 42$$

inductive definition of M:

$$M(a'' + '' b) = M(a) + M(b); M('' foo('' a'')'') = M(a)xM(a)$$

# Semantic Domains for Dataflow Diagrams?



## Semantic Domains for Dataflow Diagrams

Dataflow diagrams consist of computational nodes equipped with input and output channels for communication.

Semantics: **structural** view or **behavioural** view.

Does a computational component have memory? Can it be nondeterministic? Can the component react to partial input by emitting a partial result? Can several results be sent as a reaction to a single input? Is there a need to track the causality between input and output or is a message trace sufficient? Do the components need to be greedy, and can they emit messages spontaneously? Is there a buffer along the communication lines between components for storing unprocessed messages, or are messages lost if unprocessed? Is the fairness of processing input from different sources guaranteed? Is feedback (looping) in the diagram allowed?

## Semantic Domains for Dataflow Diagrams

Different answers to such questions lead to a variety of different kinds of semantic domains for behaviour:

- traces
- input/output-relations
- streams and stream-processing functions
- ...

## Semantic Domain: Dataflow Diagrams

In the simplest case, the dataflow network

- is deterministic;
- reacts only to complete sets of inputs;
- has no memory

It is then sufficient to adopt a function from inputs to outputs as the semantic domain:

$$IOfunc : I \rightarrow O$$

For our example language

$$IOfunc : Nat \rightarrow Nat$$

defined by

$$IOfunc(n) = n(n+1)/2.$$

## Semantic Domain: Dataflow Diagrams

Another semantic domain could be the set of traces, which includes observations of inputs and outputs in an interleaved manner:

$$IOtrace = \{x | x \in (I \cup O)*\}$$

where * denotes Kleene iteration

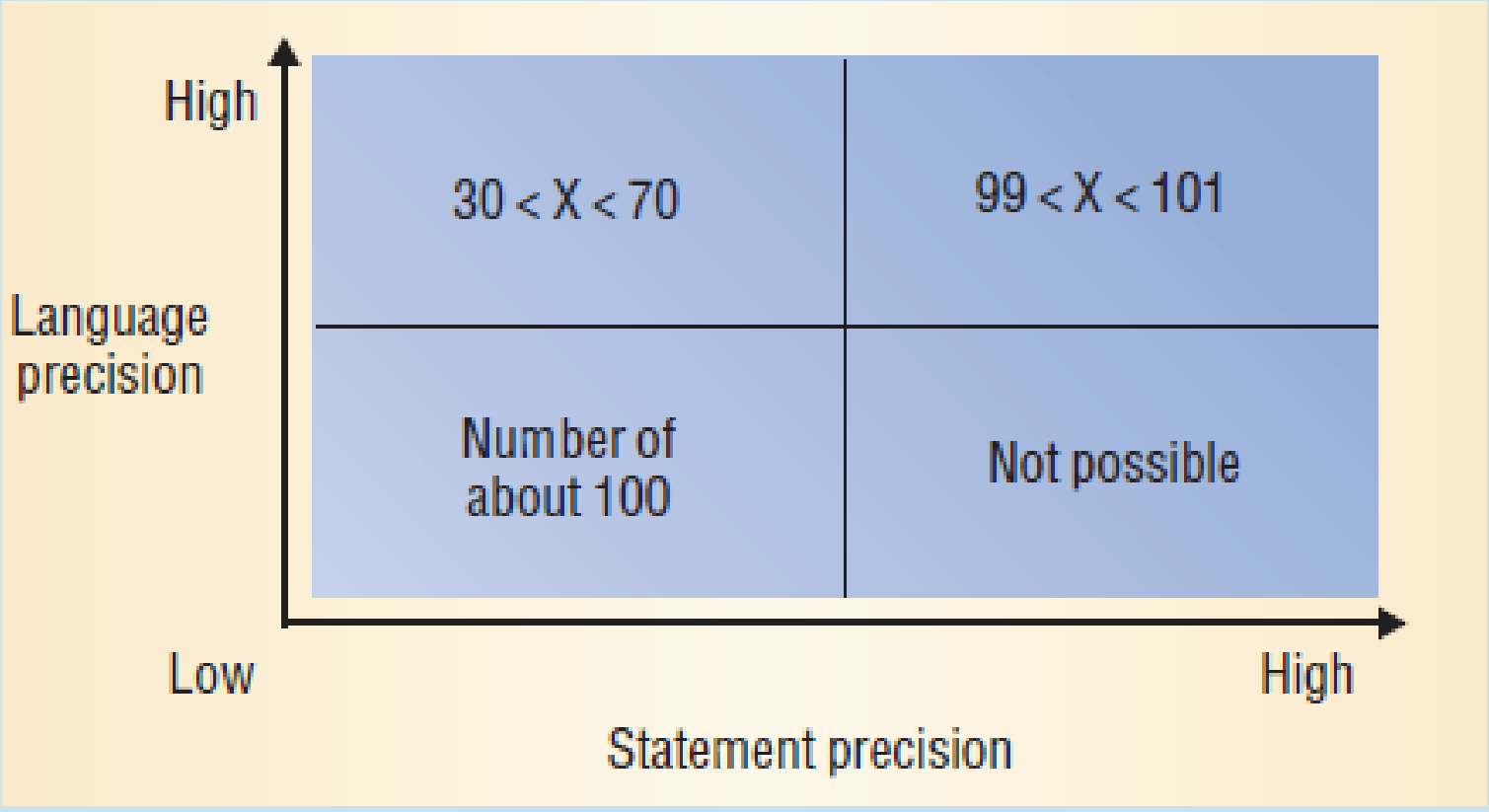
## Abstract syntax

"essence", "in memory"

## Requirements for semantic mapping function

- total (defined for all elements of language)
- unique (single meaning) . . .
  how about non-deterministic semantics?

# Degree Of Formality

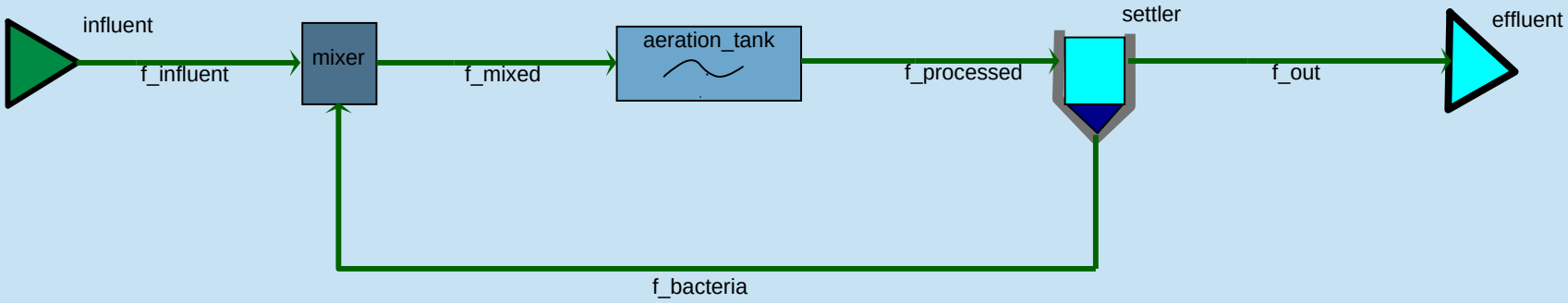


# Operational vs. Denotational (Translational) semantics

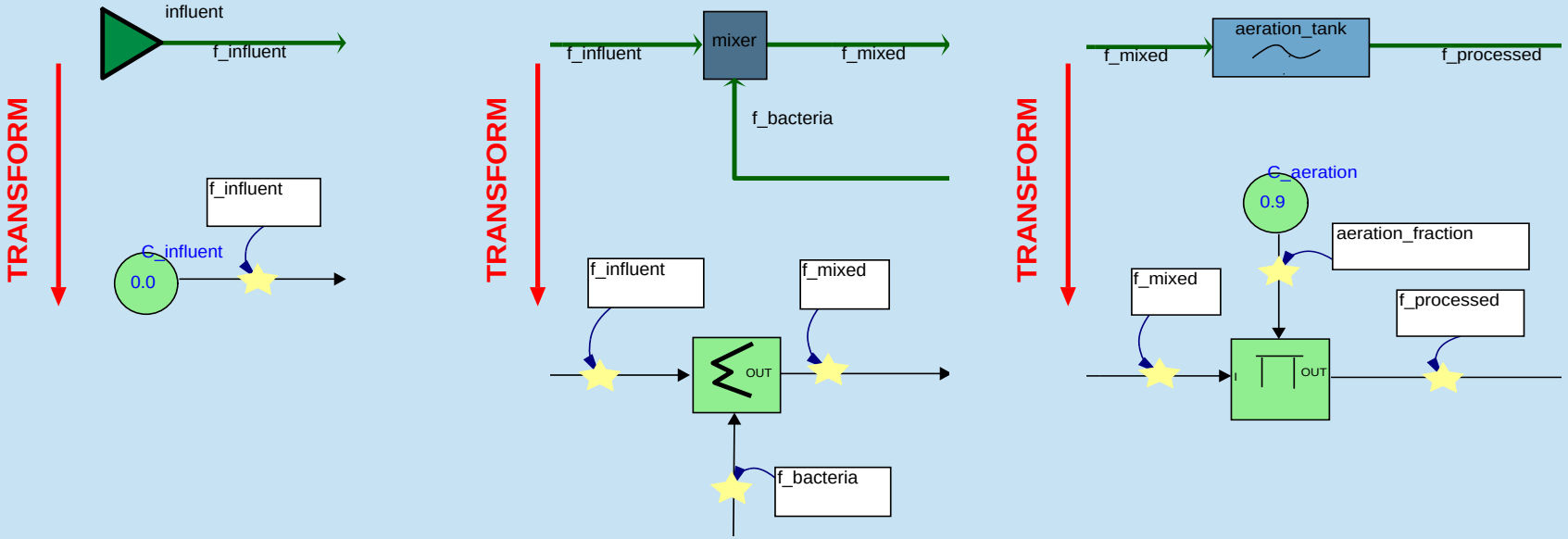
NATO's Sarajevo Waste Water Treatment Plant

`www.nato.int/sfor/cimic/env-pro/waterpla.htm`

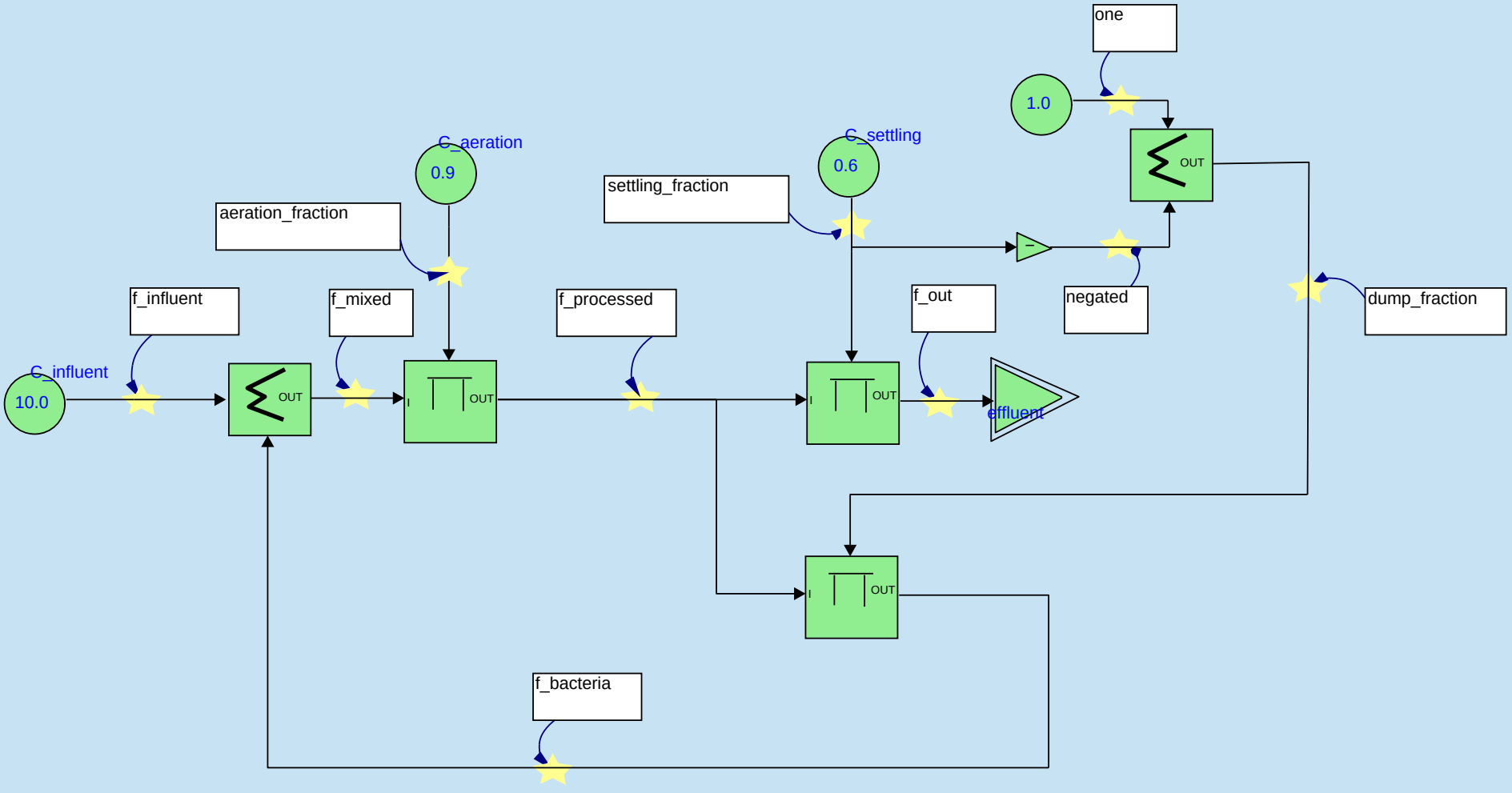
# What does this WWTP model mean?



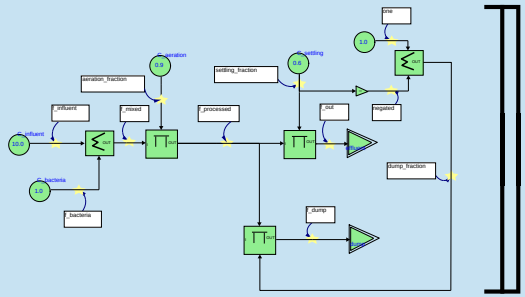
# semantic mapping of WWTP onto . . .



# . . . its meaning (steady-state abstraction): Causal Block Diagram (CBD)



# Meaning of the CBD . . . semantic mapping onto algEqns

$$[\![\ \text{  }\ ]\!] = \left\{ \begin{array}{rcl} f\_influent & = & C\_influent \\ f\_bacteria & = & C\_bacteria \\ f\_mixed & = & f\_influent + f\_bacteria \\ aeration\_fraction & = & C\_aeration \\ f\_processed & = & aeration\_fraction * f\_mixed \\ settling\_fraction & = & C\_settling \\ negated & = & -settling\_fraction \\ one & = & 1 \\ dump\_fraction & = & one + negated \\ f\_dump & = & f\_processed * dump\_fraction \\ f\_out & = & settling\_fraction * f\_processed \end{array} \right.$$

## Misconception 1: Semantics is the metamodel

A metamodel is a model of a language's (abstract) syntax.
A semantic domain as well as a semantic mapping are still required.
Note that in practice

1. the semantic domain's syntax is also given by means of a metamodel;
2. the semantic mapping is described at the meta-level as a transformation between syntactic elements of the language and its semantic domain.

## Misconception 2: Semantics is the semantic domain

Using semantics and semantic domain interchangeably is erroneous, since it avoids the most crucial part of the semantics–the semantic mapping.

## Misconception 3: Semantics is the context conditions

This use of the term has its roots in compiler theory, where everything beyond the basic context-free grammar is viewed as semantics.
It seems to have had a great influence on the way the Object Constraint Language constraints are used on top of the UML's metamodel.
In the UML standardization documents, **static semantics** is used instead of **context conditions**.
This does not entail a semantic domain nor a semantic mapping. It simply **further constrains the syntax**.

## Misconception 4: Semantics is dealing with behaviour

For some languages, semantics explains behaviour. However, structure description languages, for example, don't talk about behaviour, but they still need semantics. Semantics and behaviour are not to be confused.

## Misconception 5: Semantics is being executable

Taking the previous point one step further, some people equate having semantics with being executable.
Clearly, if a language is executable, it probably has an adequate semantics, although that semantics might not have been given an adequately clear representation. However, not all languages specify behaviour, and not all those that do so are (or need to be) executable. Also, even if the language is meant to be executable, it can have a nonexecutable, denotational semantics.

## Misconception 6: Semantics is the behaviour of a system

Sometimes people talk about the semantics of a particular **system** – the way it behaves, its reaction time, and so on. This is quite different from the semantics of the **languages** used to describe that system.

## Misconception 7: Semantics is the meaning of individual constructs

People often refer to the semantics of some part of the language, even just one construct. Clearly, there is much more to semantics than that.

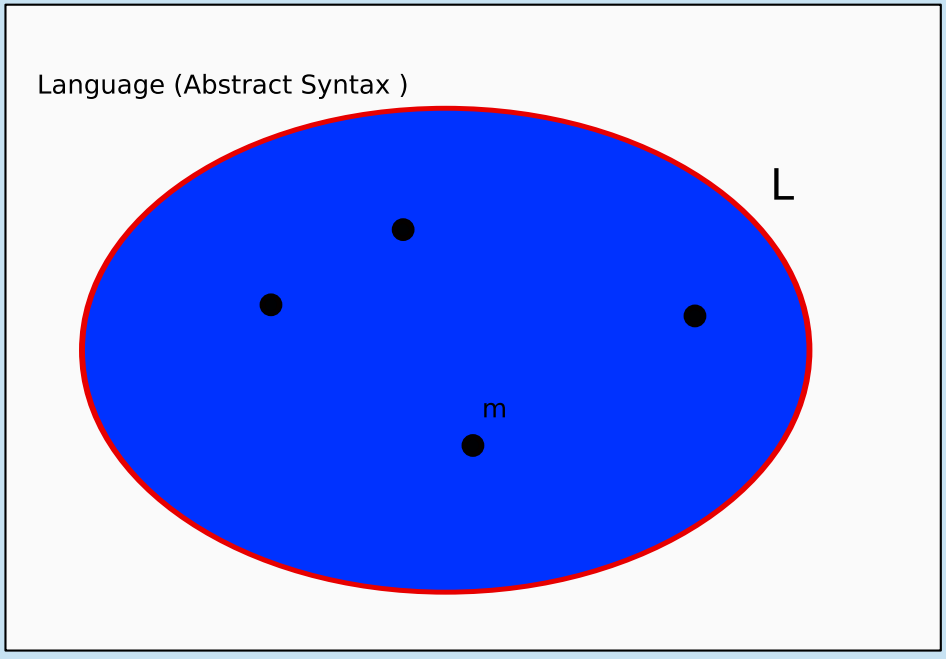
## Misconception 8: Semantics means looking mathematical

When some people see that parts of a language definition have mathematical symbols, they are convinced that it is probably also **precisely defined**. This is simply not true.

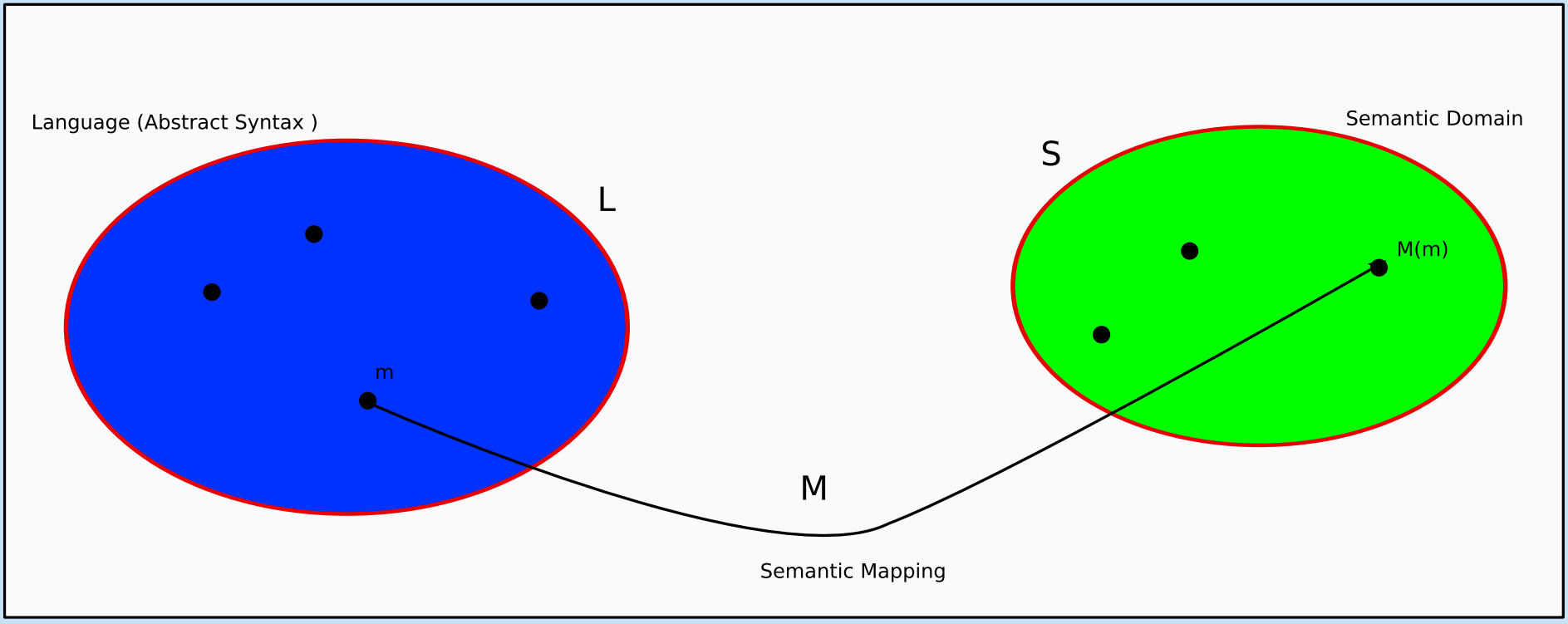
# Deciding on terminology


$$\sum$$

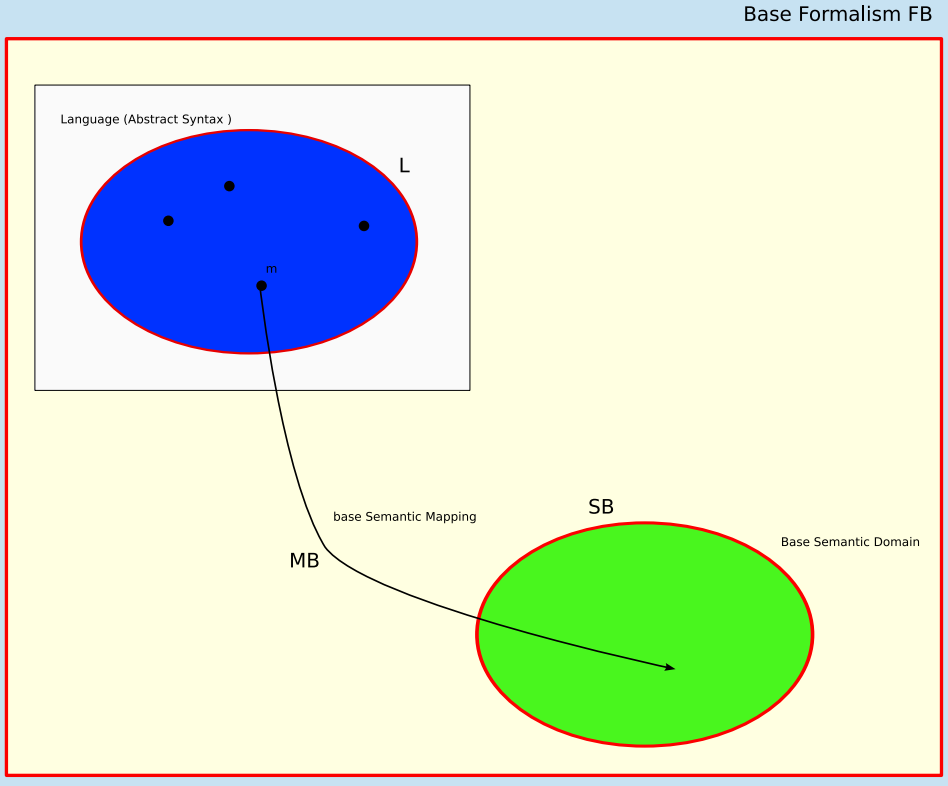
# What's in a name ? **Language**



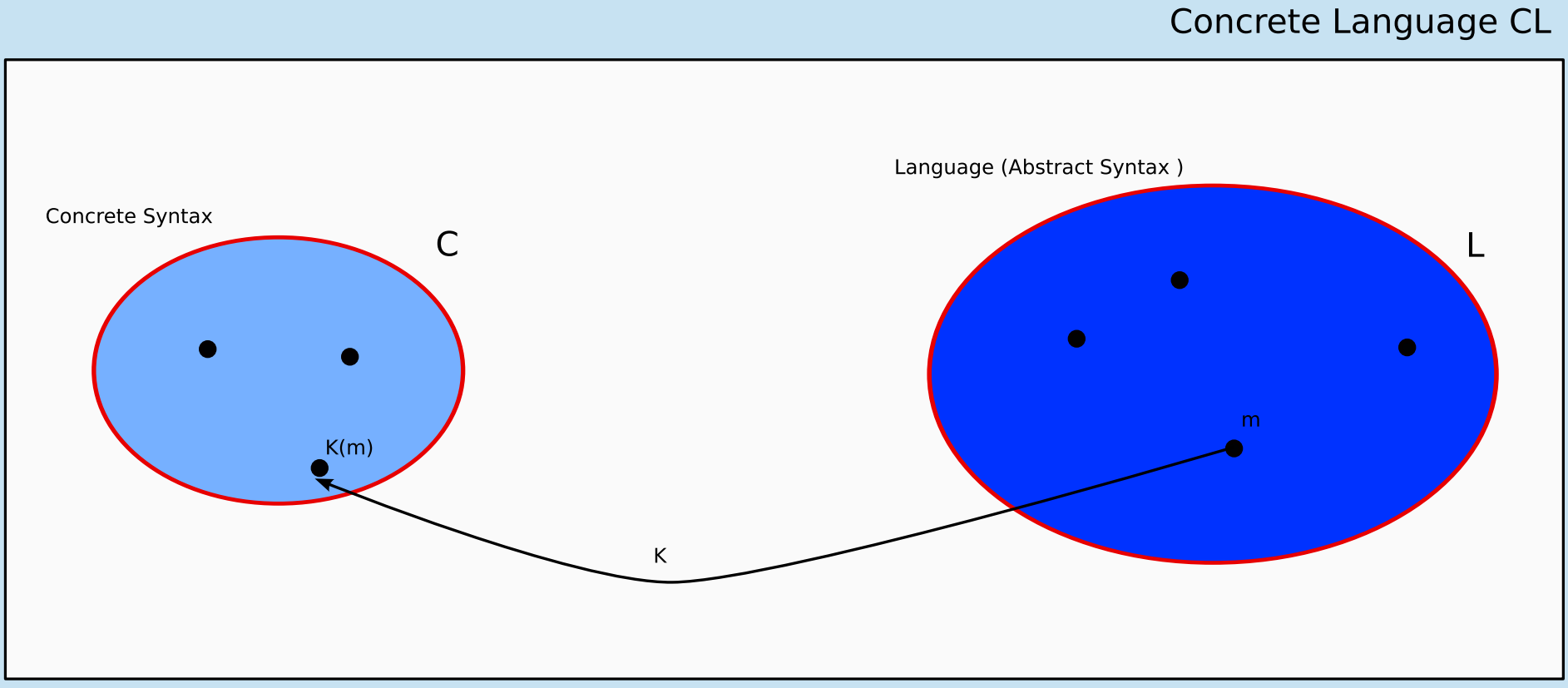
# What's in a name ? **Formalism**



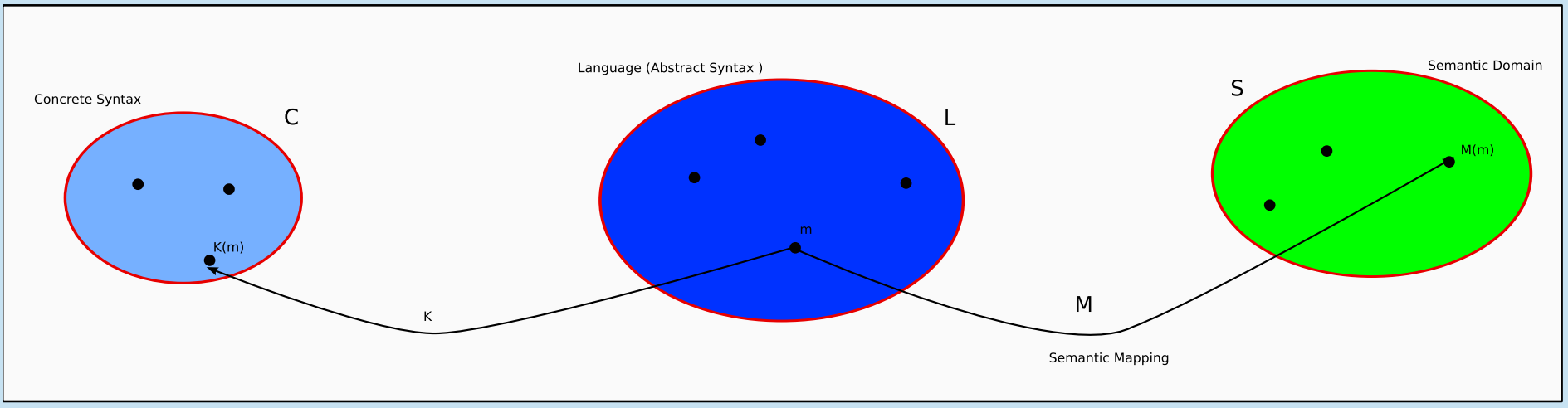
# What's in a name ? **Base Formalism**



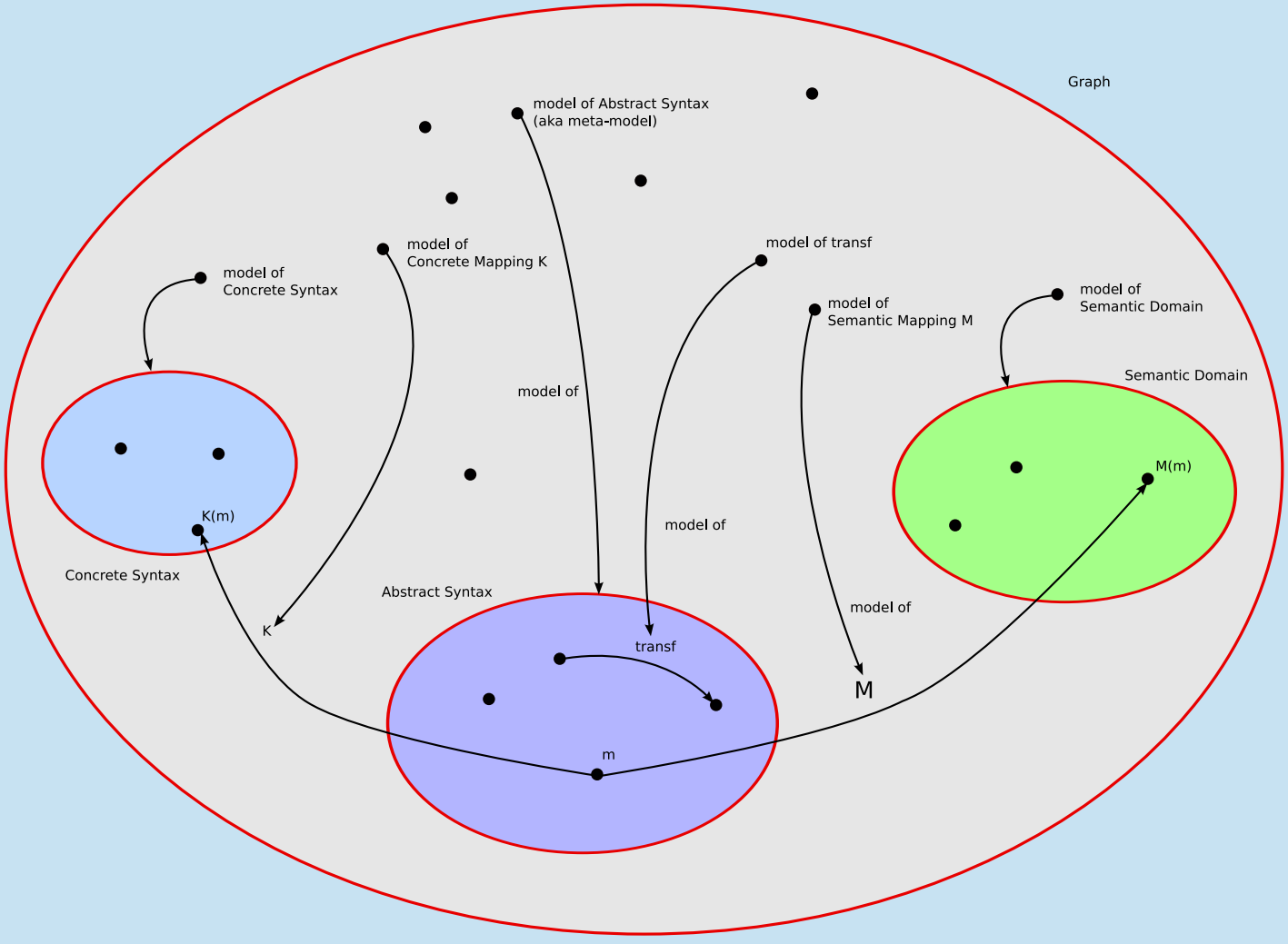
# What's in a name ? **Concrete Language**



# What's in a name ? **Concrete Formalism**



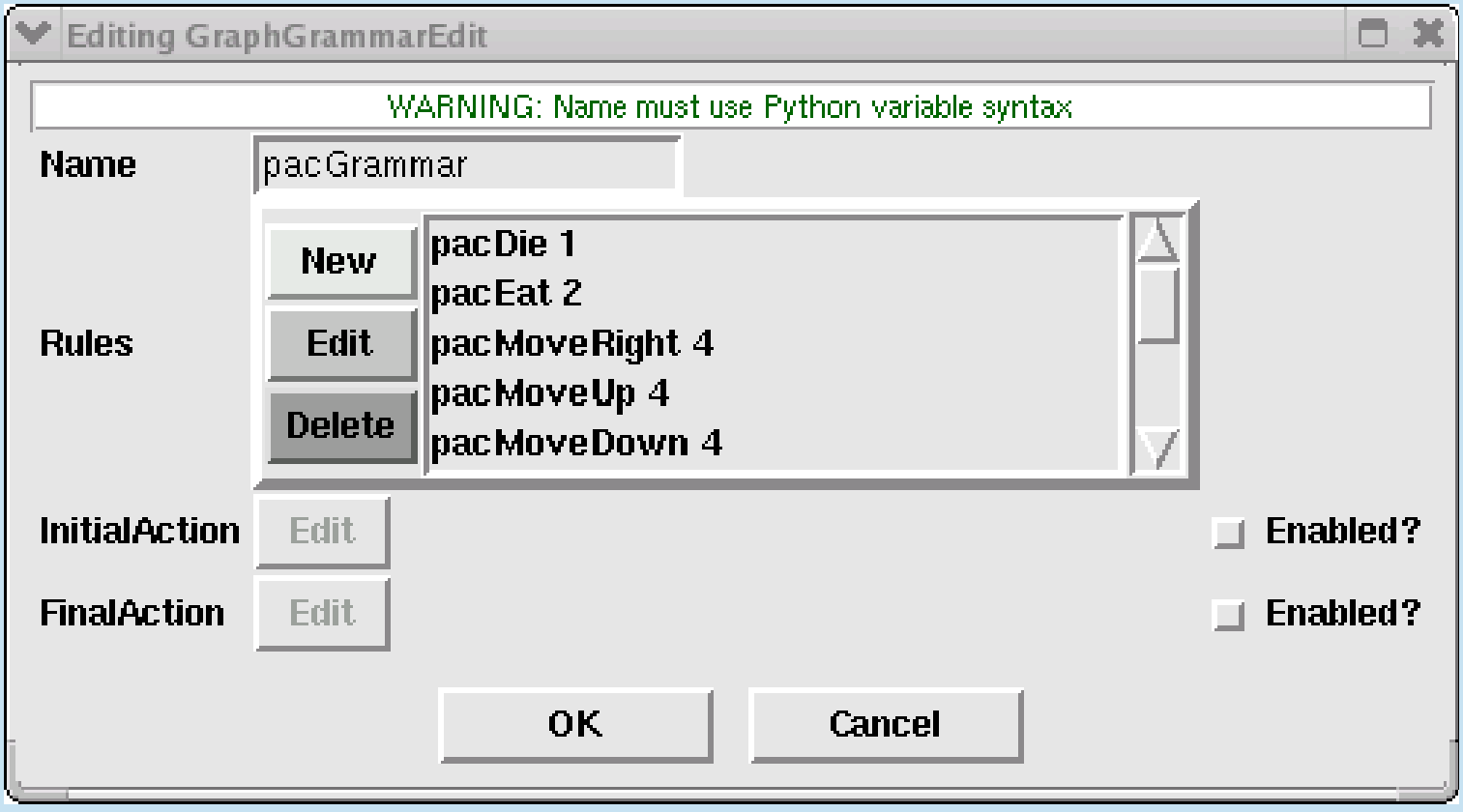
# Modelling a Modelling Language/Formalism



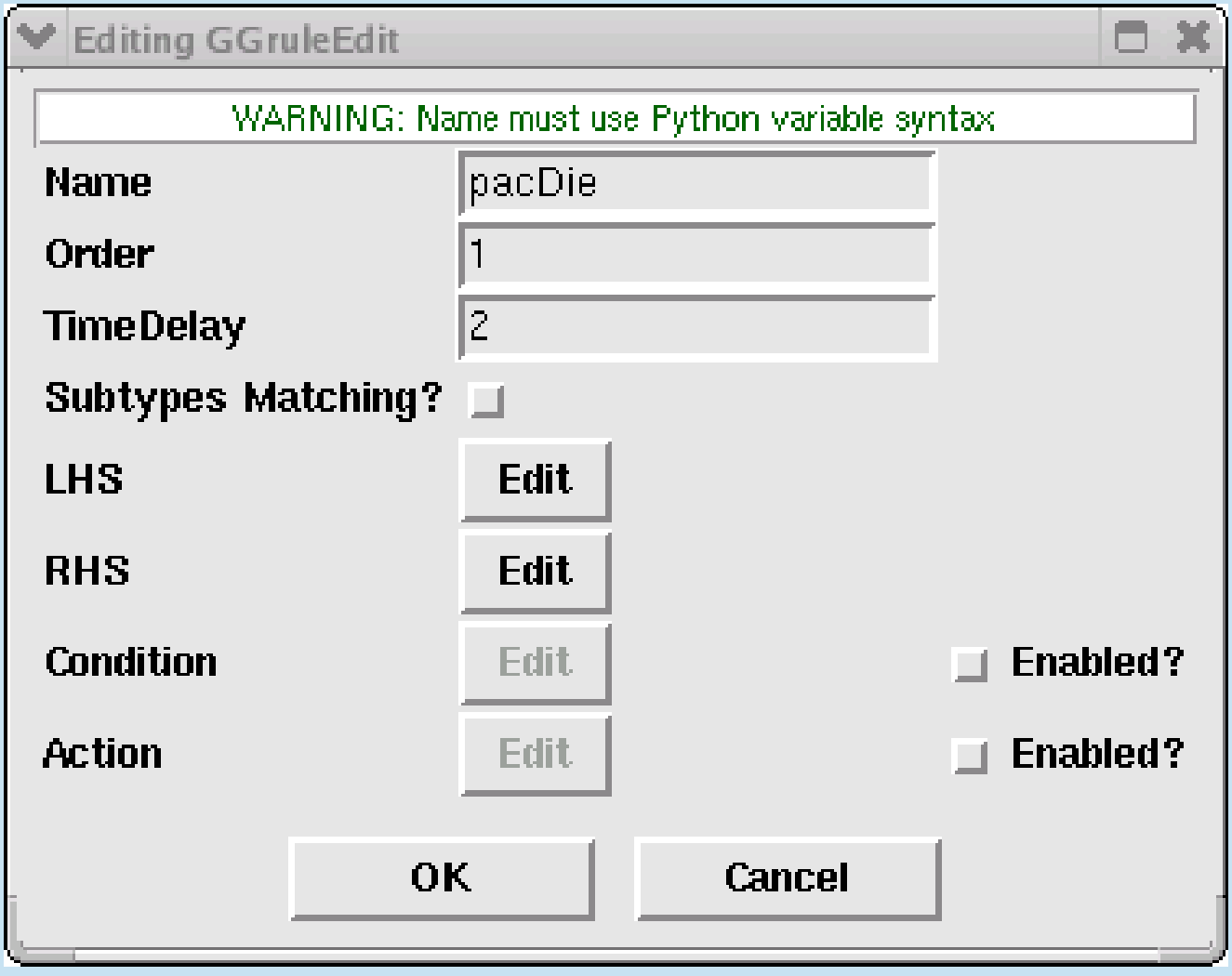
## Graph Grammars
## to Specify Model Transformations

Rationale:
Models are often graph-like $\Rightarrow$ natural to express model transformation by means of graph transformation models.
Ehrig, H., G. Engels, H.-J. Kreowski, and G. Rozenberg.
**Handbook of graph grammars and computing by graph transformation.**
1999. World Scientific.
Tools:
GME/GReAT, PROGRES, AGG, AToM$^3$, Fujaba, GROOVE, ...
First two used (and Fujaba) in large industrial applications.

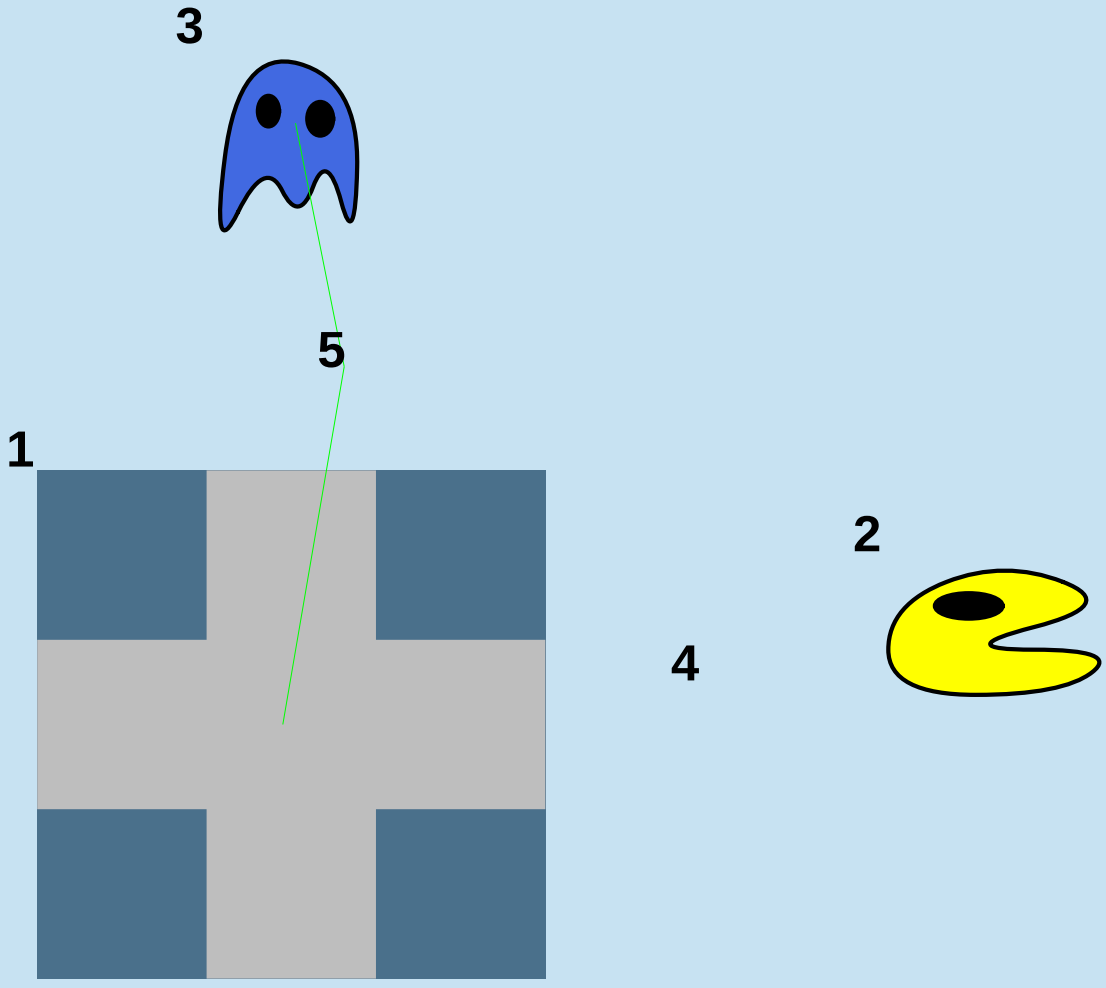
# Model Operational Semantics using GG



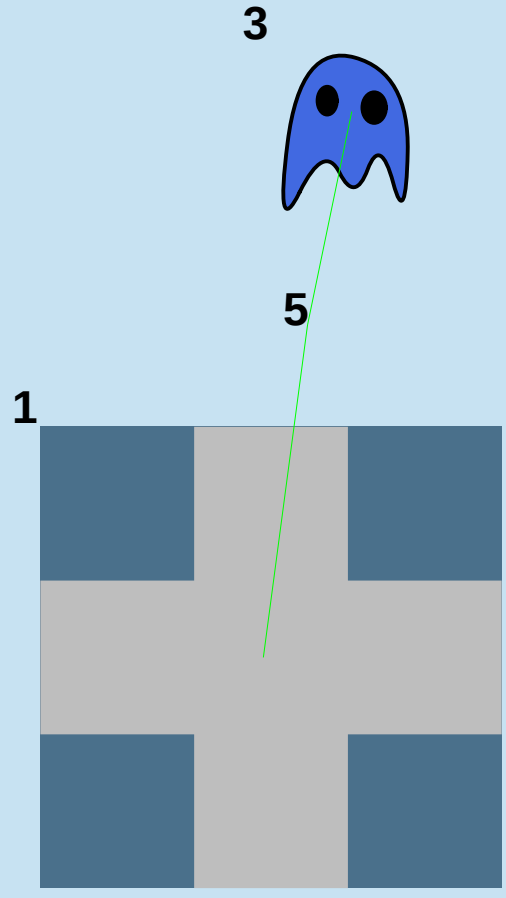
# PacMan Die rule



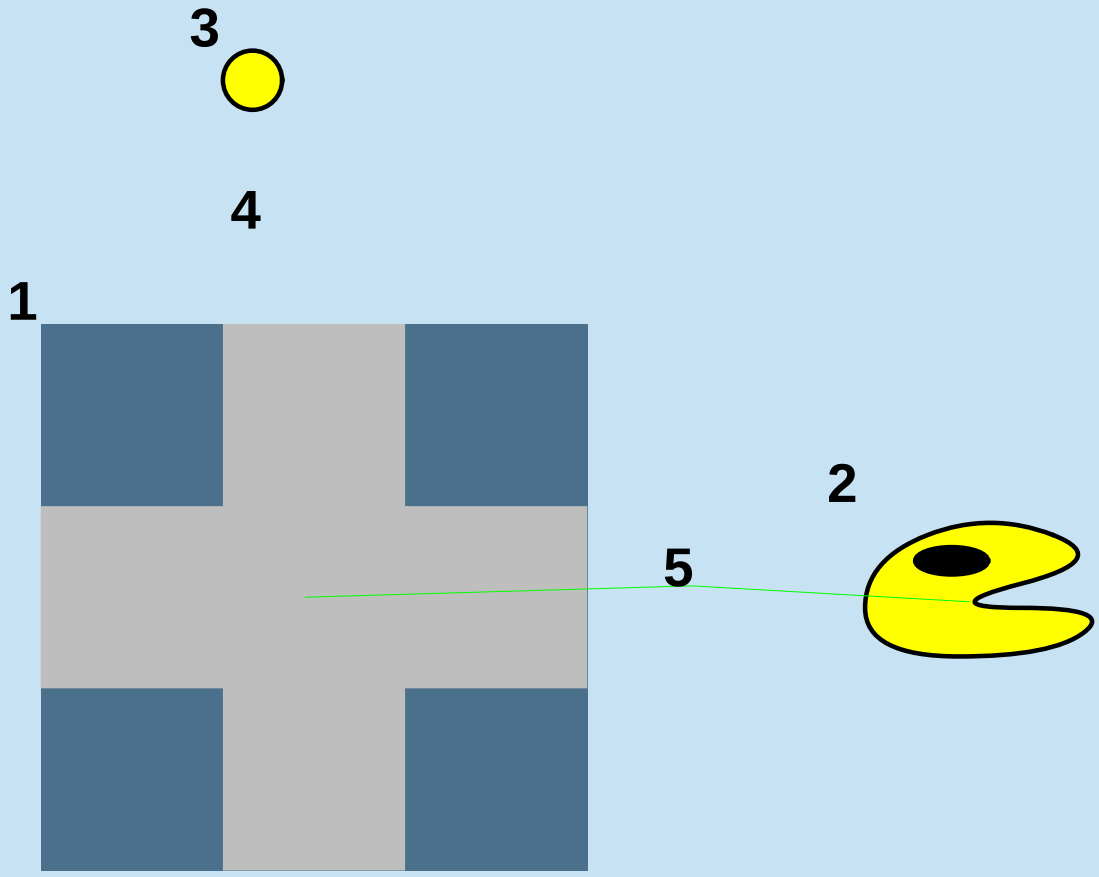
# PacMan Die rule LHS



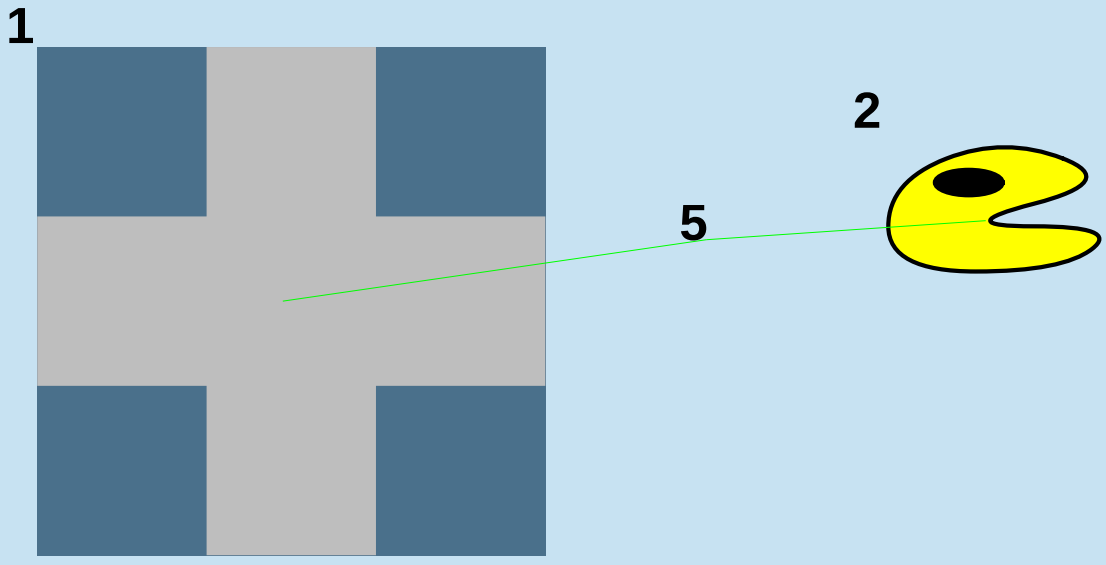
# PacMan Die rule RHS



# PacMan Eat rule LHS

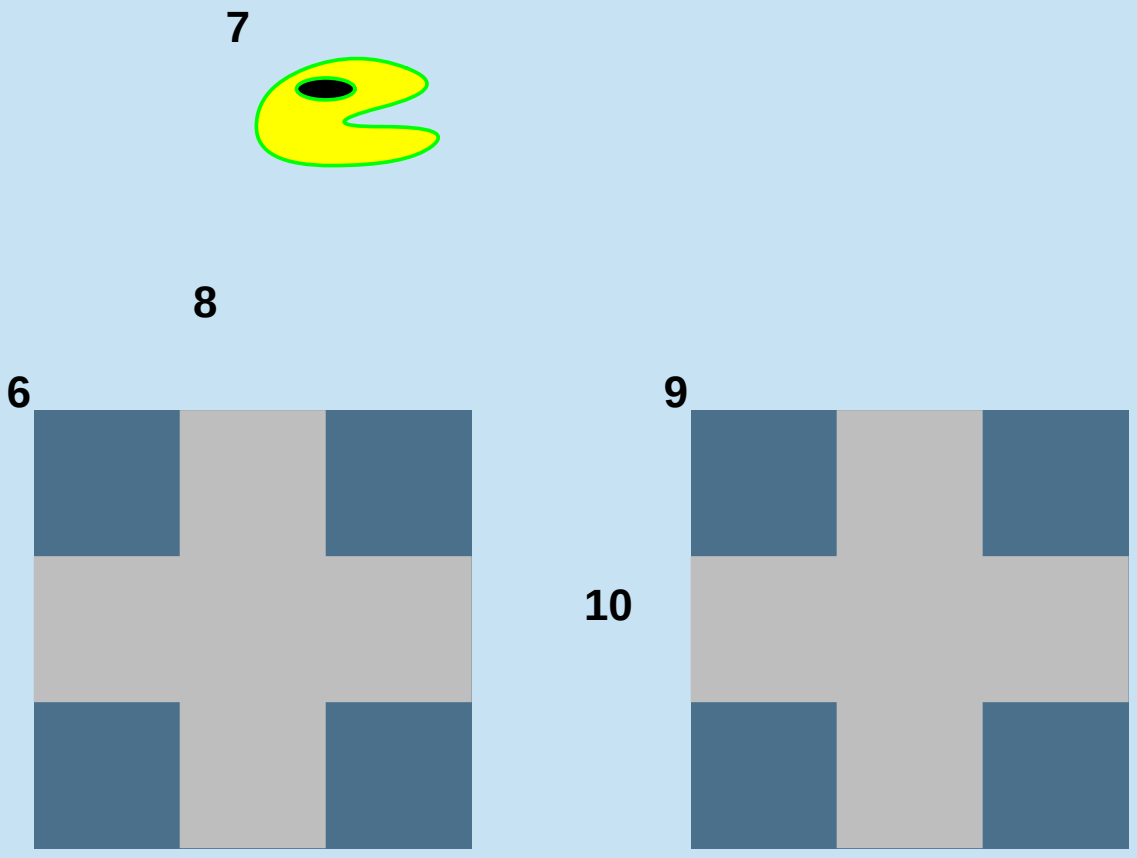


# PacMan Eat rule RHS

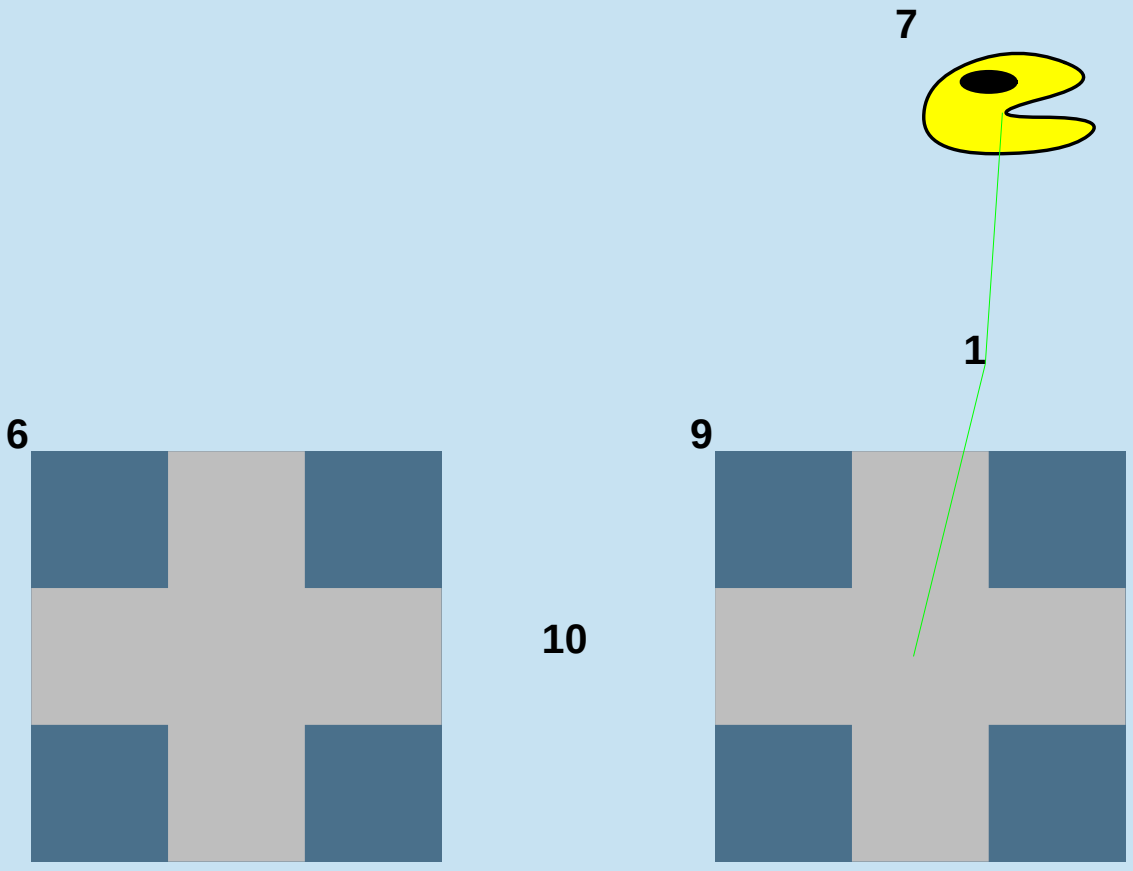



```
scoreBoard = None
scoreBoards = atom3i.ASGroot.listNodes['ScoreBoard']
if (not scoreBoards):
  return
else:
  scoreBoard = scoreBoards[0]
  scoreVal = scoreBoard.score.getValue()
  scoreBoard.score.setValue(scoreVal+1)
  scoreBoard.graphObject_.ModifyAttribute('score',scoreVal+1)
```

# PacMan Move rule LHS

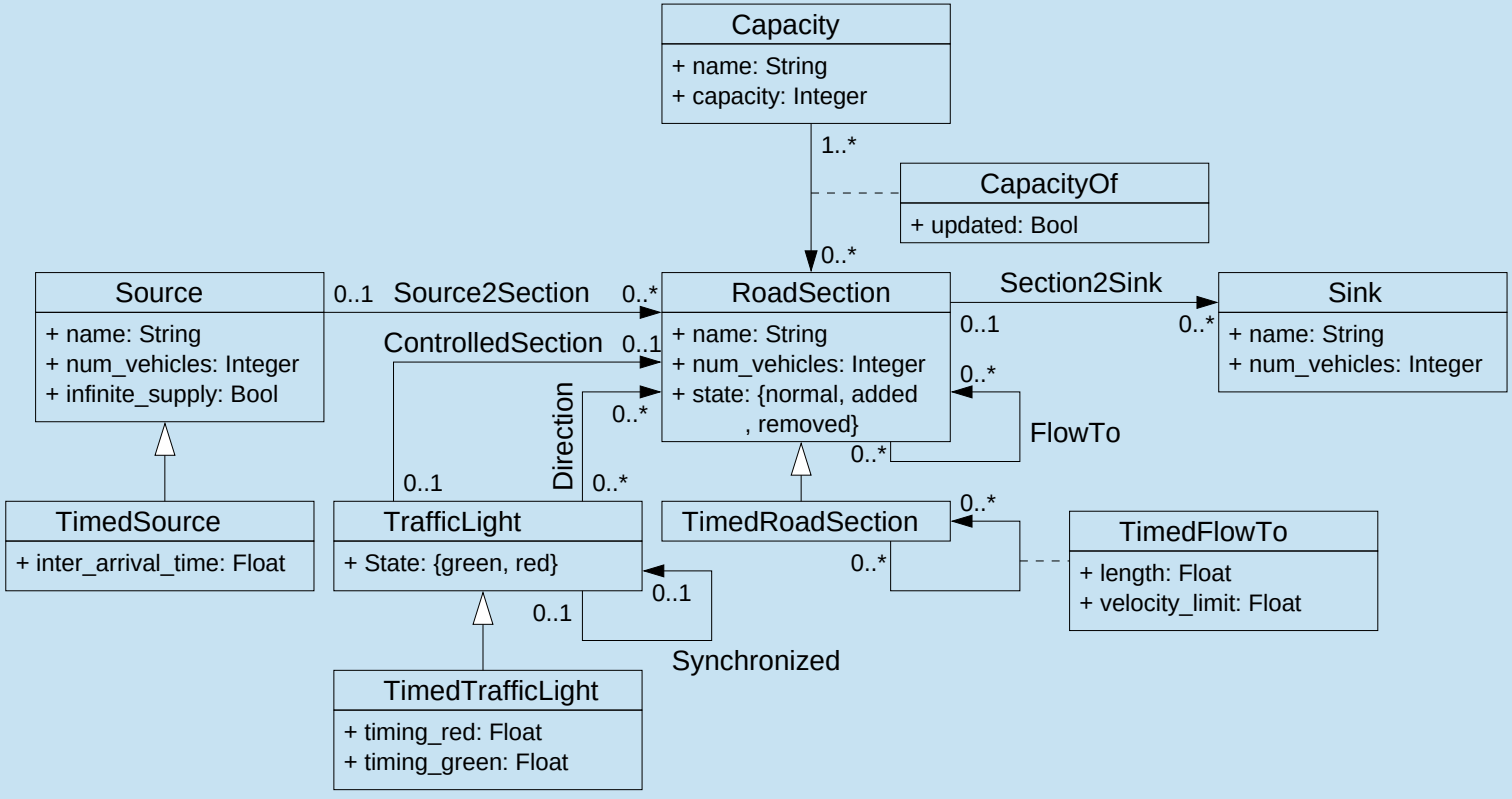


# PacMan Move rule RHS



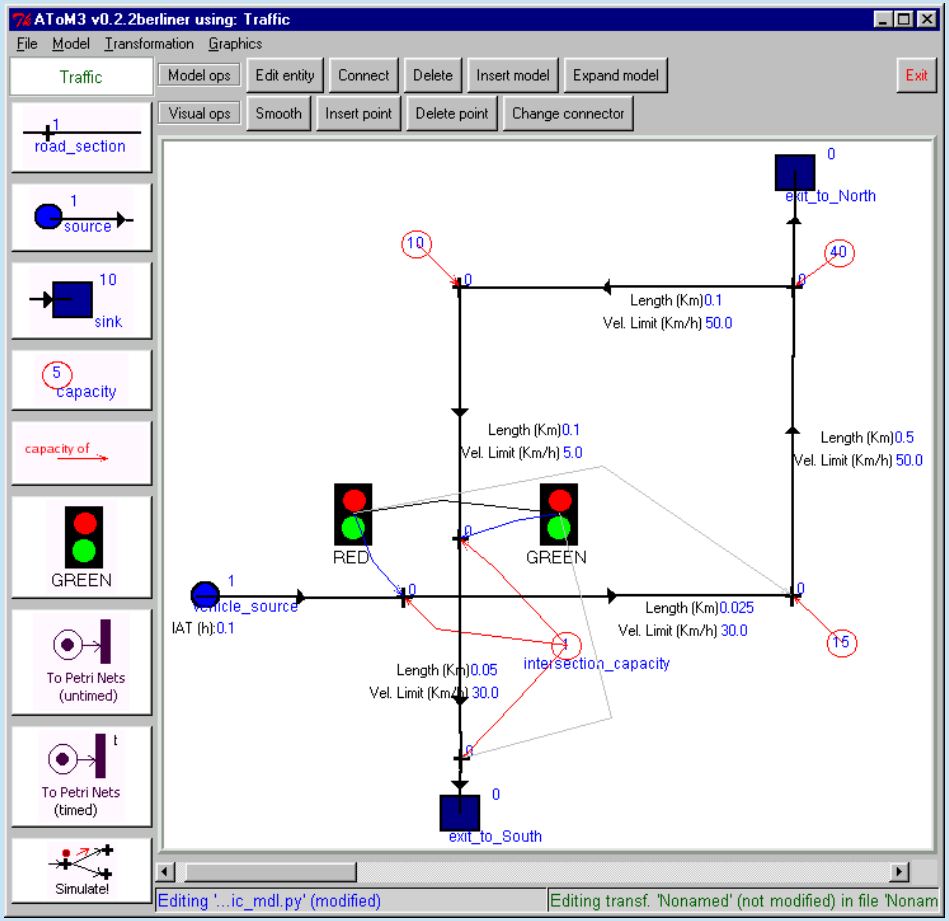
# Formalism Transformation Example: Model/Analyze/Simulate Traffic Networks

# Un-timed and timed **Traffic** meta-model (a UML Class Diagram)



# **Traffic** Concrete Syntax (the Capacity Entity)

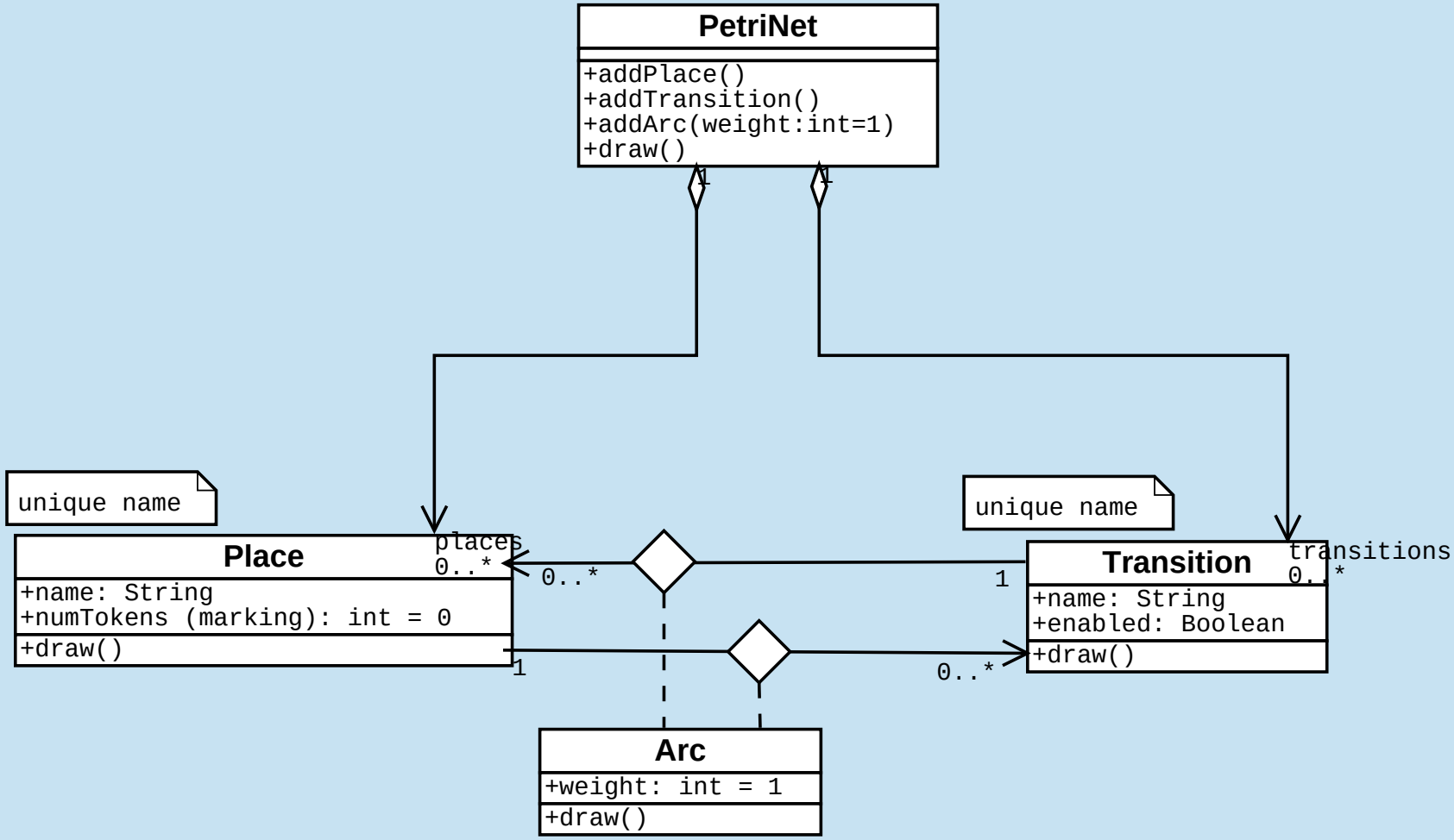
# Synthesized **Traffic** Visual Modelling Environment



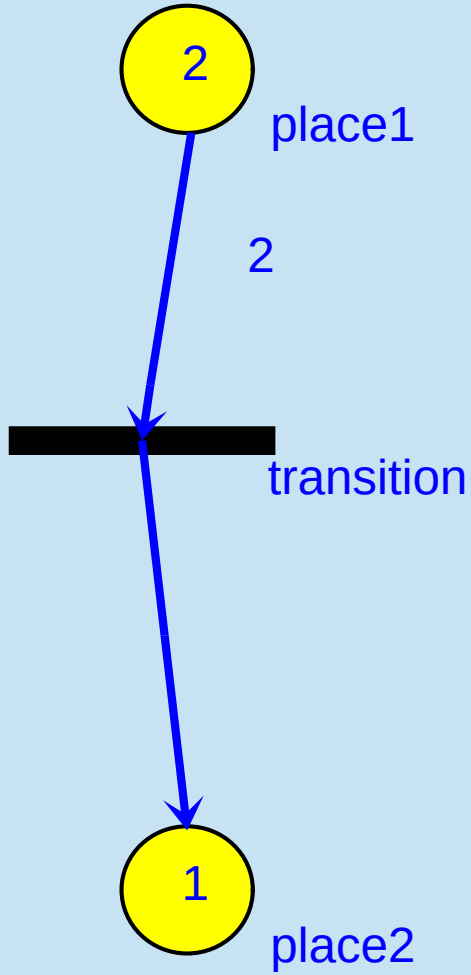
## Modelling Traffic's Semantics

- choices: timed, un-timed, . . . (level of abstraction)
- **denotational**: map onto known formalism (TTPN, PN)
  . . . good for analysis purposes
- **operational**: procedure to execute/simulate model
  . . . may act as a reference implementation
- note: need to **prove** consistency between denotational and operational semantics if both are given !

# Place-Transition **Petri Net** Abstract Syntax (**UML Class Diagram** formalism)



# Place-Transition **Petri Net** Concrete Syntax



## **Petri Net** Behaviour

State Transition Function $f$ of marked Petri net $(P, T, A, w, x_0)$

$$f : \mathbb{N}^n \times T \rightarrow \mathbb{N}^n$$

is defined for transition $t_j \in T$ if and only if

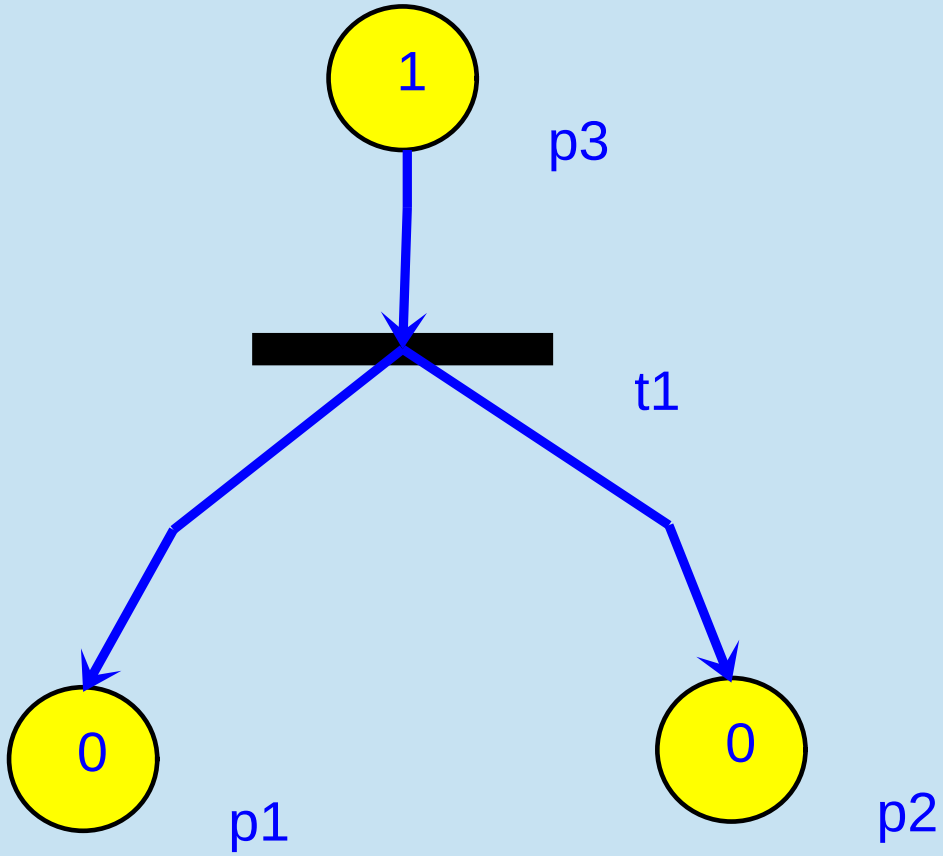
$$x(p_i) \geq w(p_i, t_j), \forall p_i \in I(t_j)$$

If $f(\mathbf{x}, t_j)$ is defined, set $\mathbf{x}' = f(\mathbf{x}, t_j)$ where

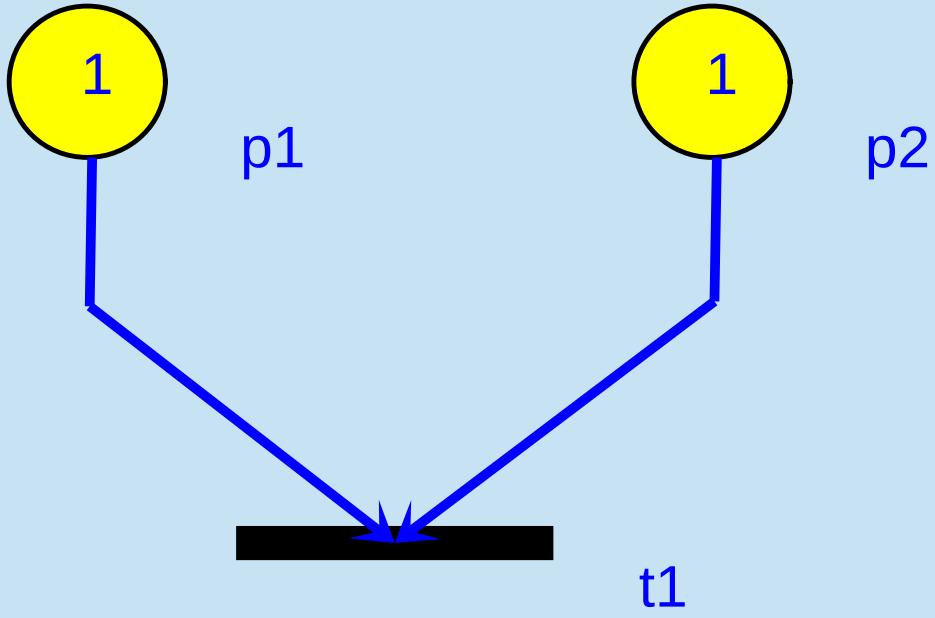
$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i)$$

- State transition function $f$ based on *structure* of Petri net
- Number of tokens *need not be conserved* (but can)

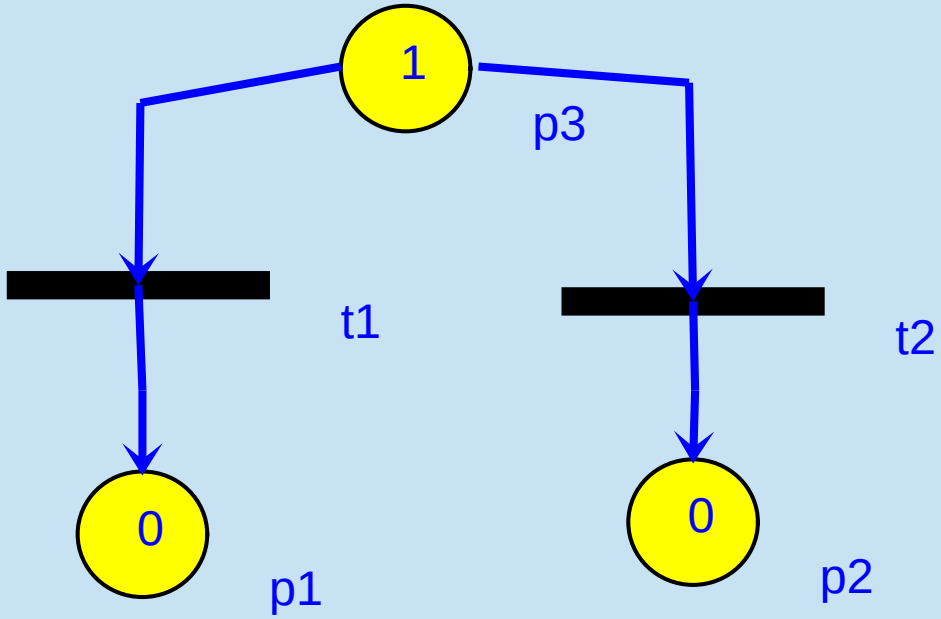
## Behaviour: Fork



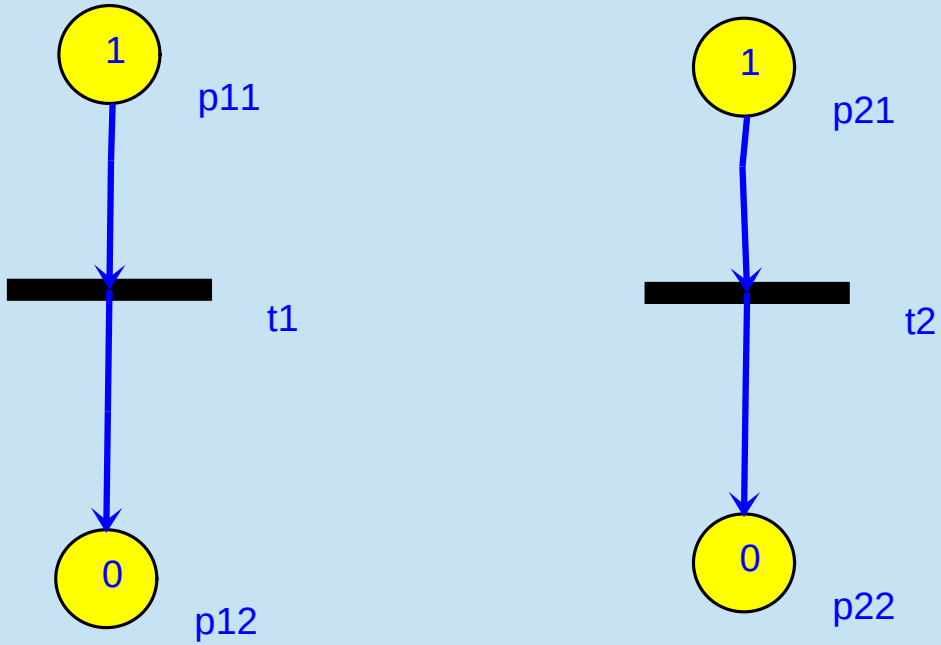
## Behaviour: Join



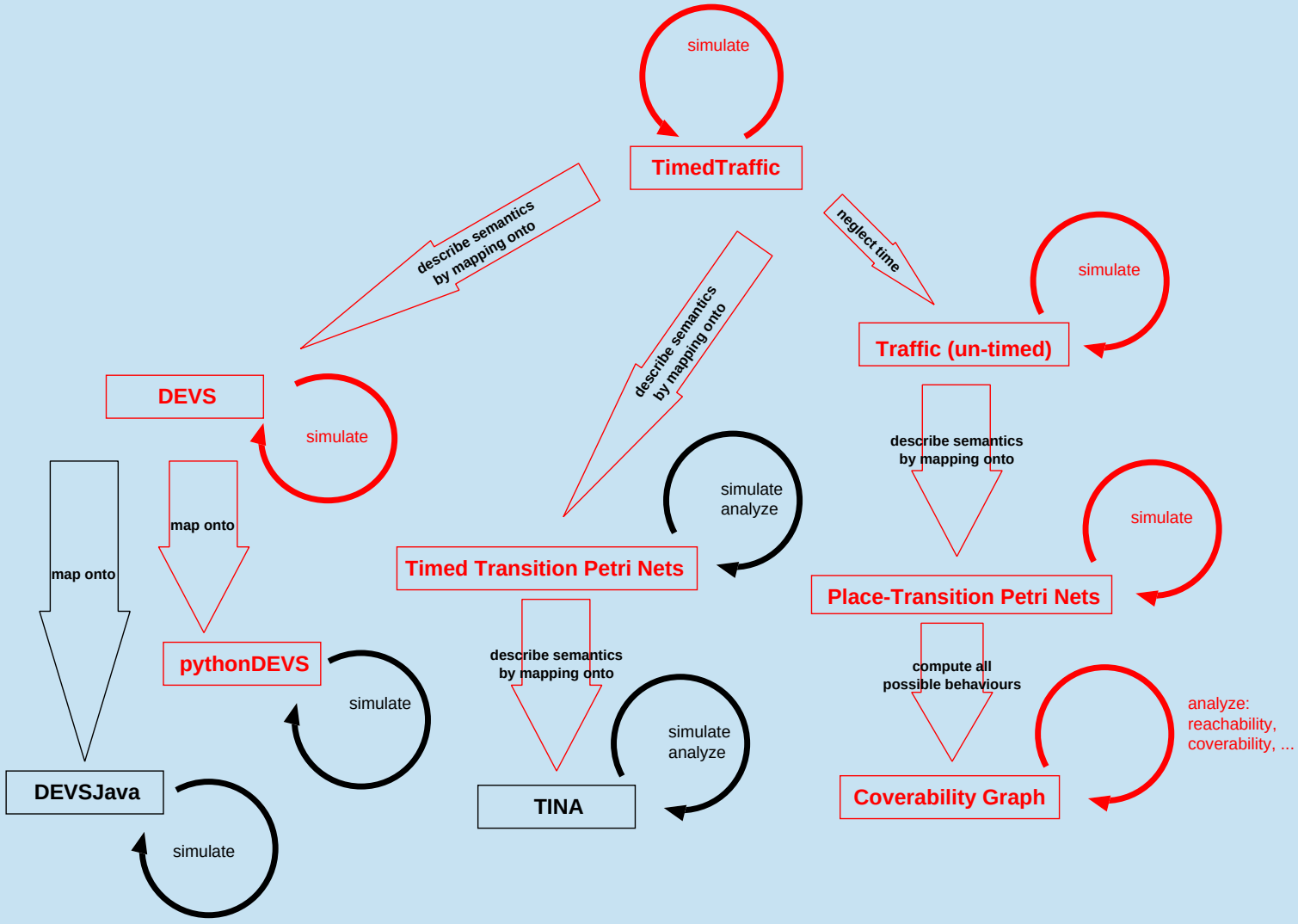
## Behaviour: Conflict, choice, decision



# Behaviour: Concurrency



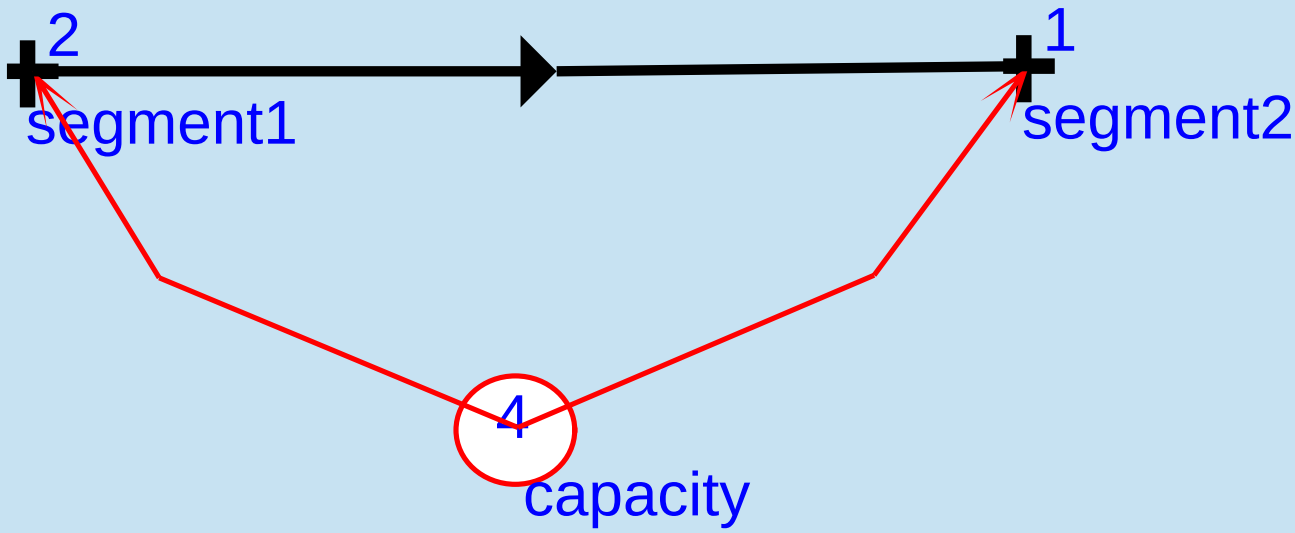
# The Big Picture: **Transformations**



## **Traffic**'s (un-timed) semantics in terms of **Petri Net**s

- need a meta-model of **Traffic** (shown before)
- need a meta-model of **Petri Net**s (shown before)
- need a meta-model of **Generic Graph** (glue)
- need a model of the mapping: **Traffic** $\Rightarrow$ **Petri Net**

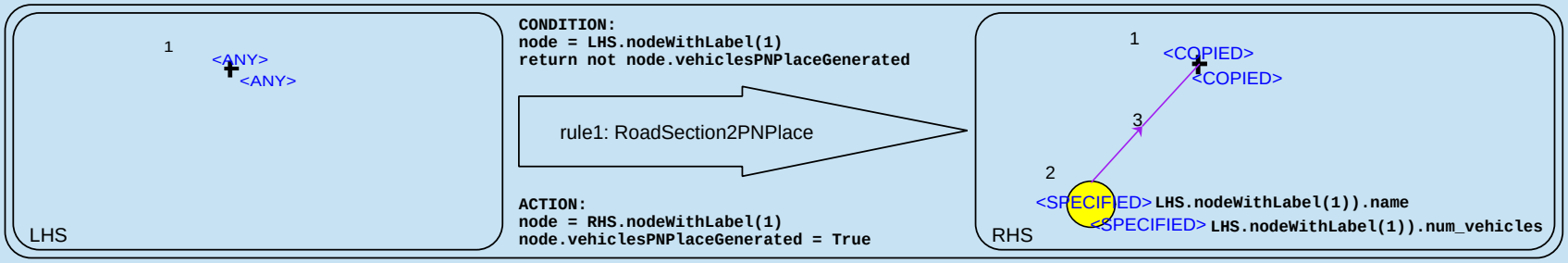
## A very simple **Traffic** model



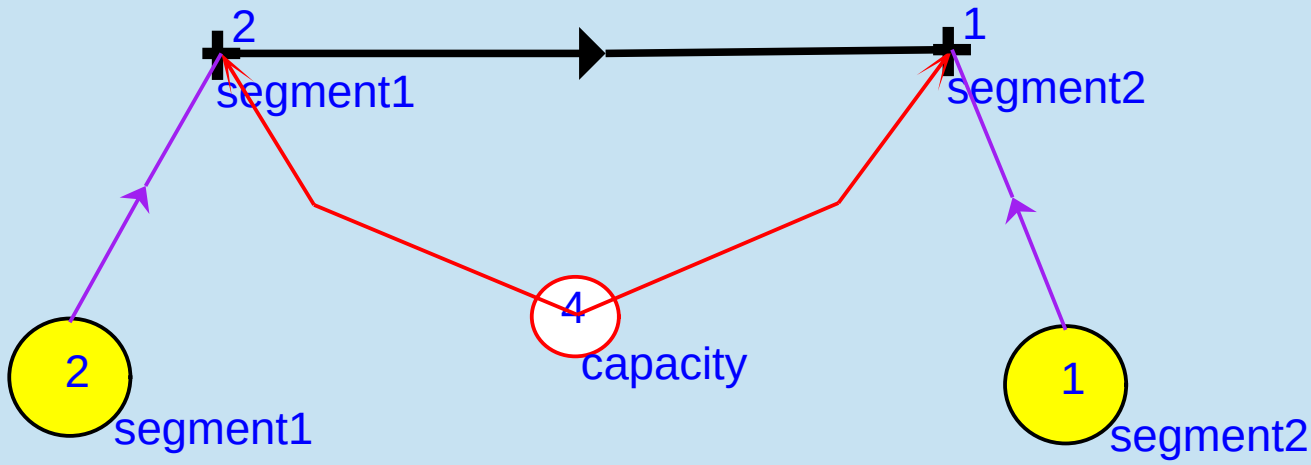
## **Traffic** to **Petri Net** Graph Grammar rules

```
INITIAL ACTION:
for node in graph.listNodes["RoadSection"]:
 node.vehiclesPNPlaceGenerated=False
```

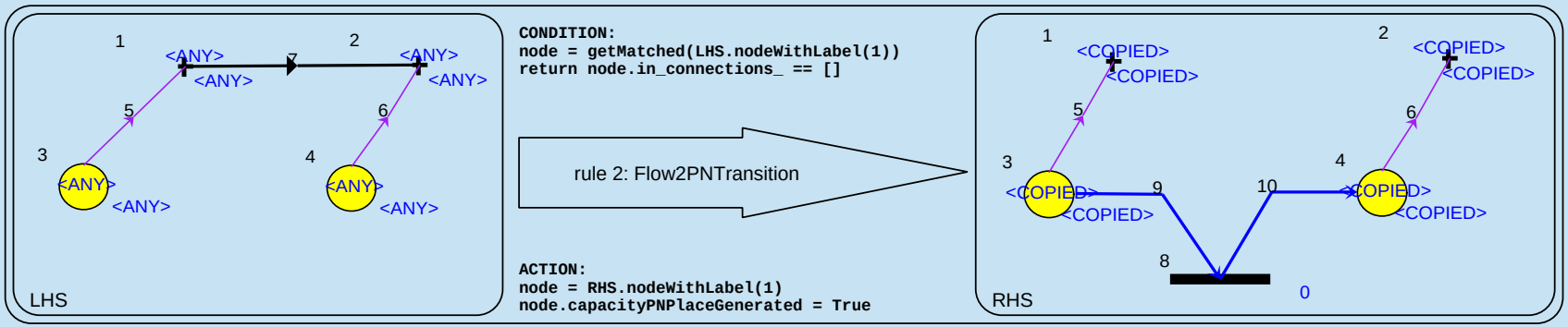
# **Traffic** to **Petri Net** Graph Grammar rules



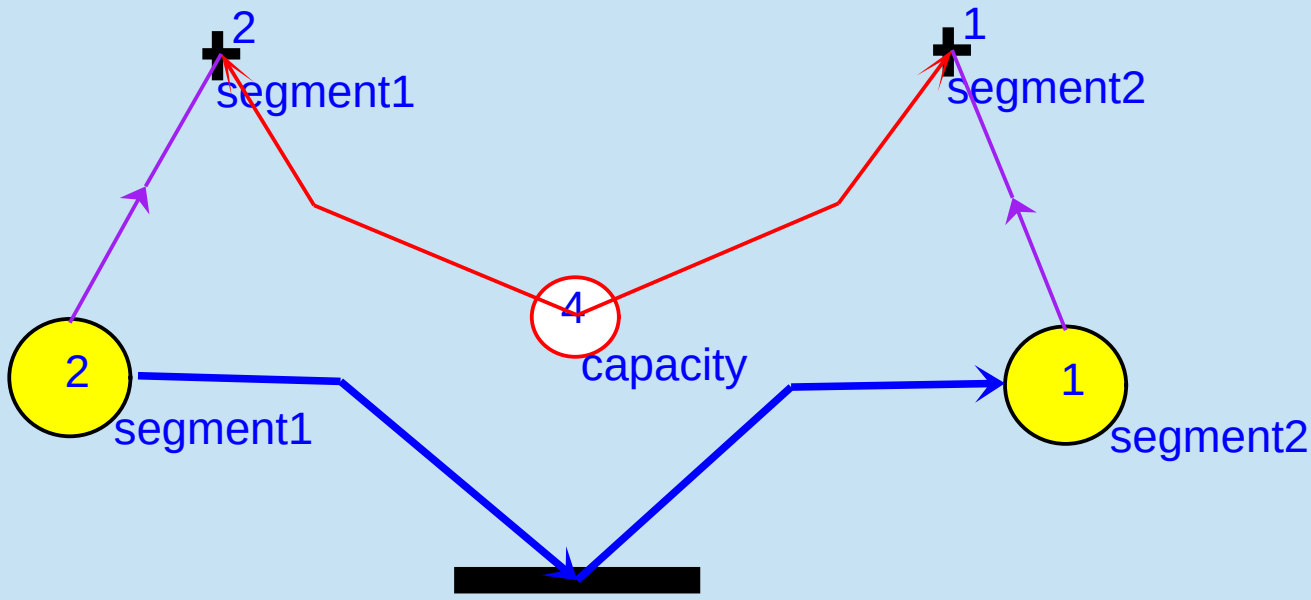
## Road Sections converted to **Petri Net** Places



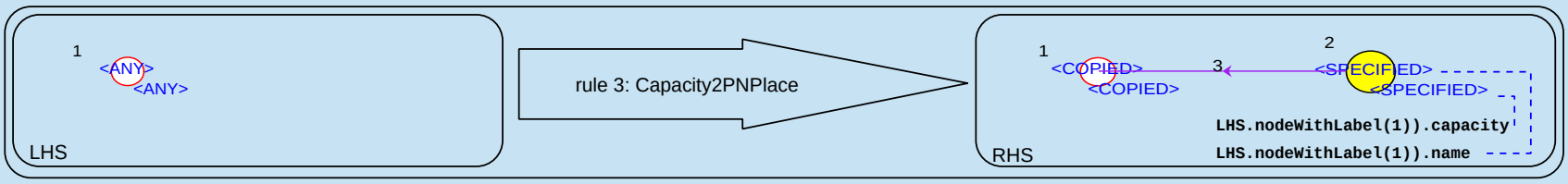
# **Traffic** to **Petri Net** Graph Grammar rules



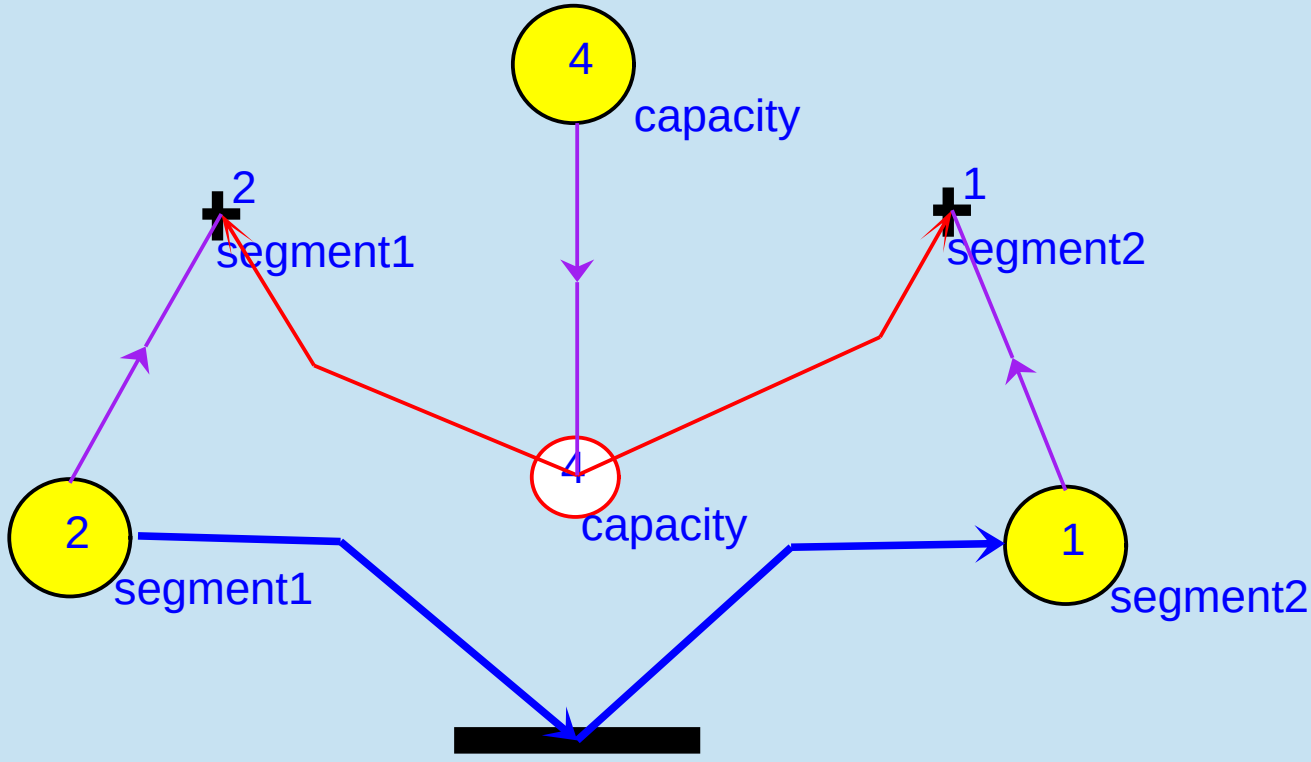
## **Traffic** Flow to **Petri Net** Transitions



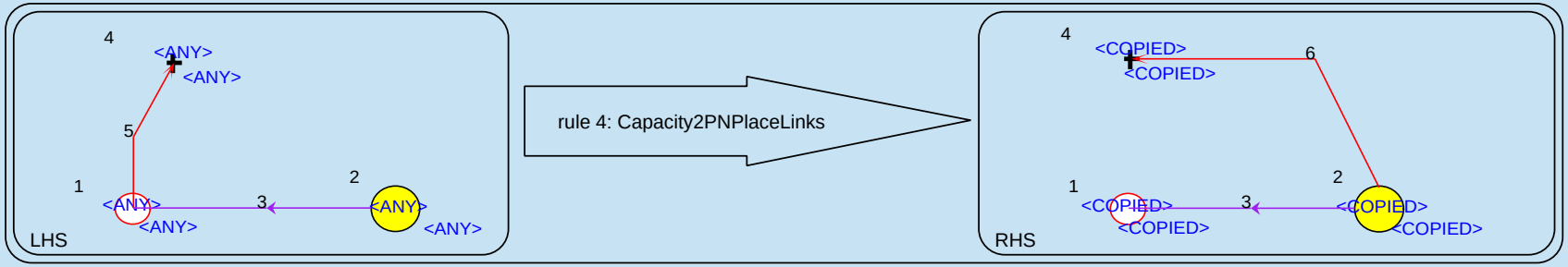
# **Traffic** to **Petri Net** Graph Grammar rules



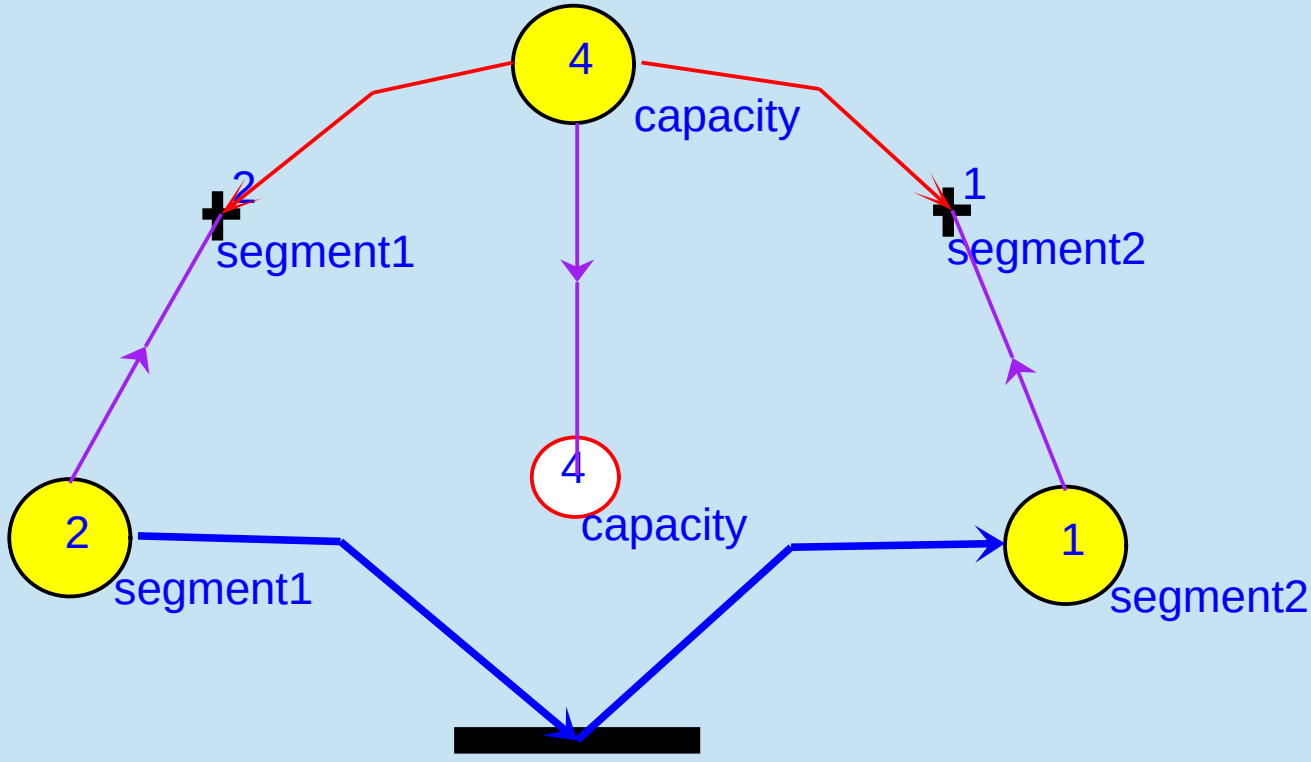
## **Traffic** Capacity to **Petri Net** Place



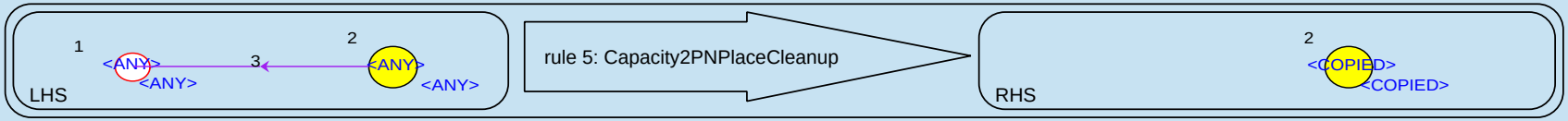
# **Traffic** to **Petri Net** Graph Grammar rules



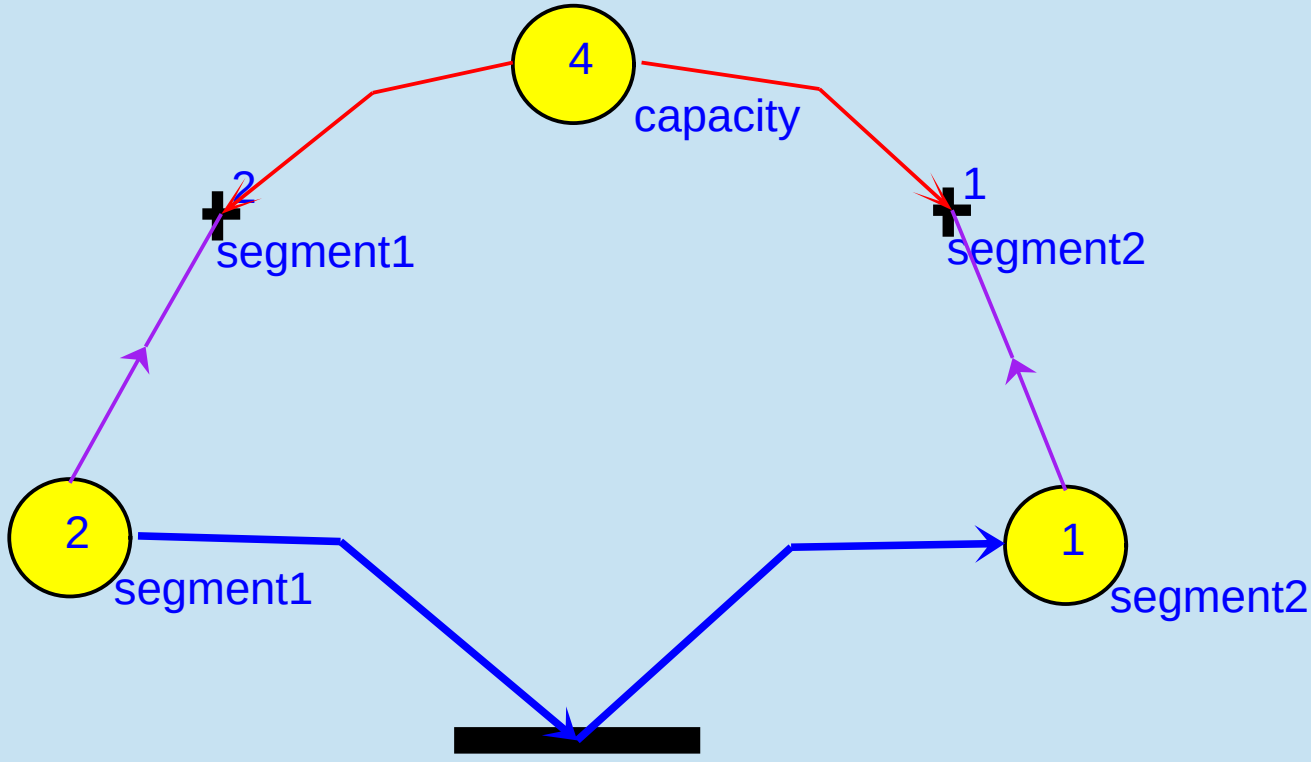
## **Traffic** Capacity to **Petri Net** Place (links)



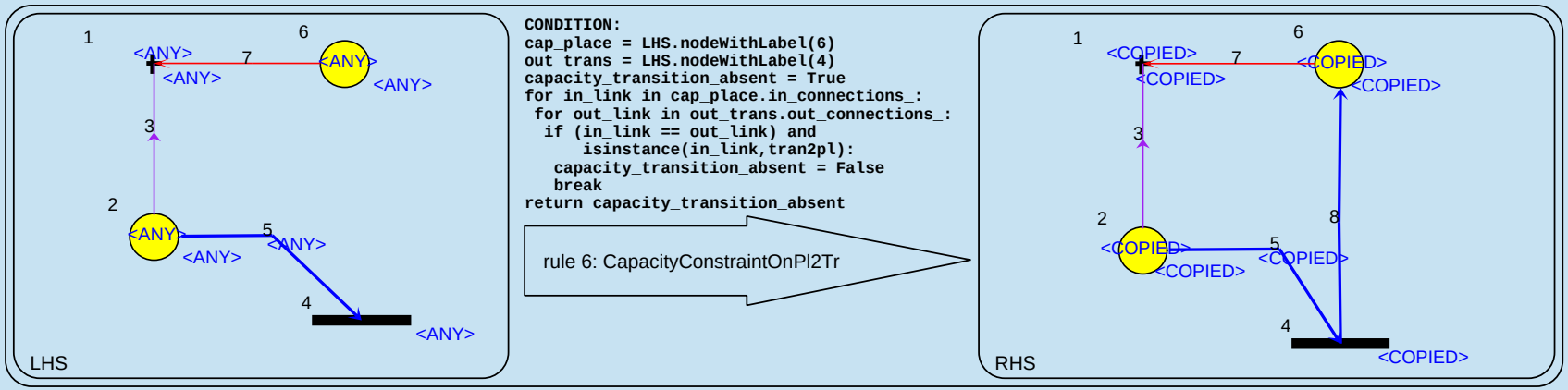
# **Traffic** to **Petri Net** Graph Grammar rules



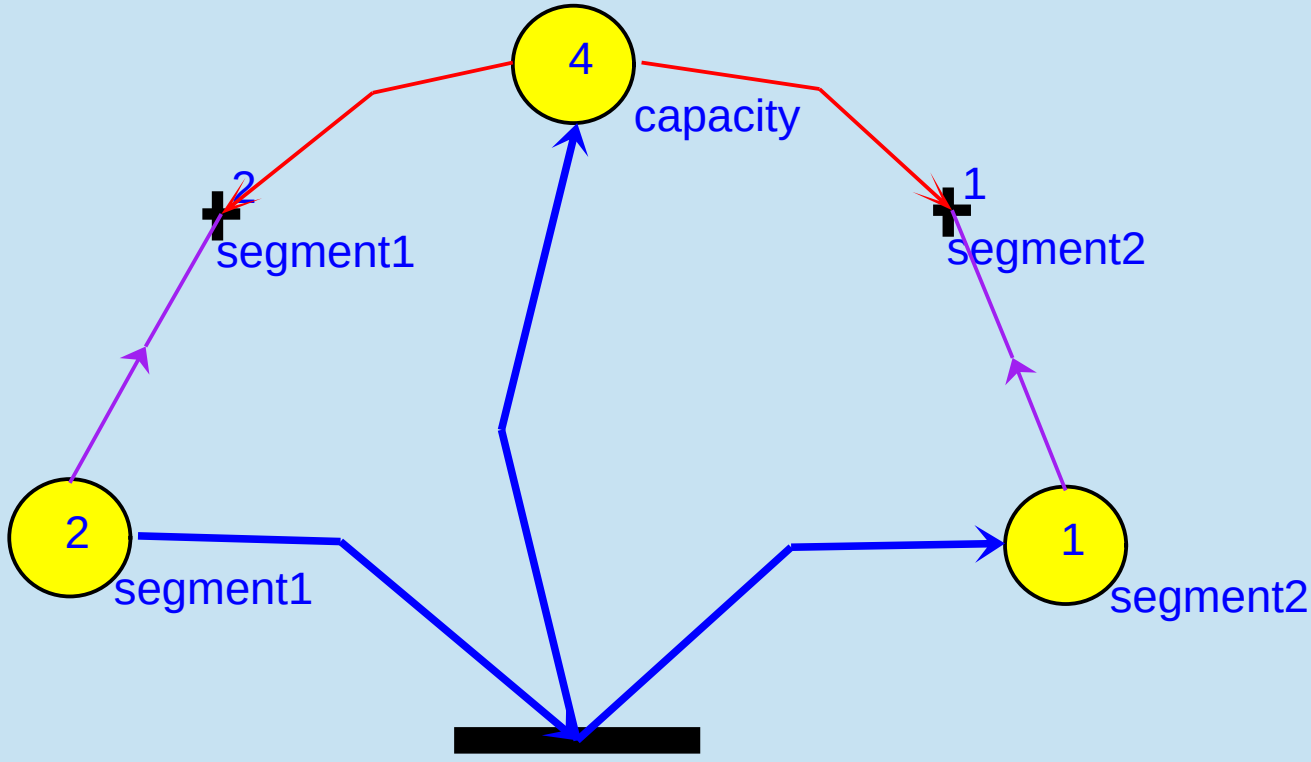
## **Traffic** Capacity to **Petri Net** Place cleanup



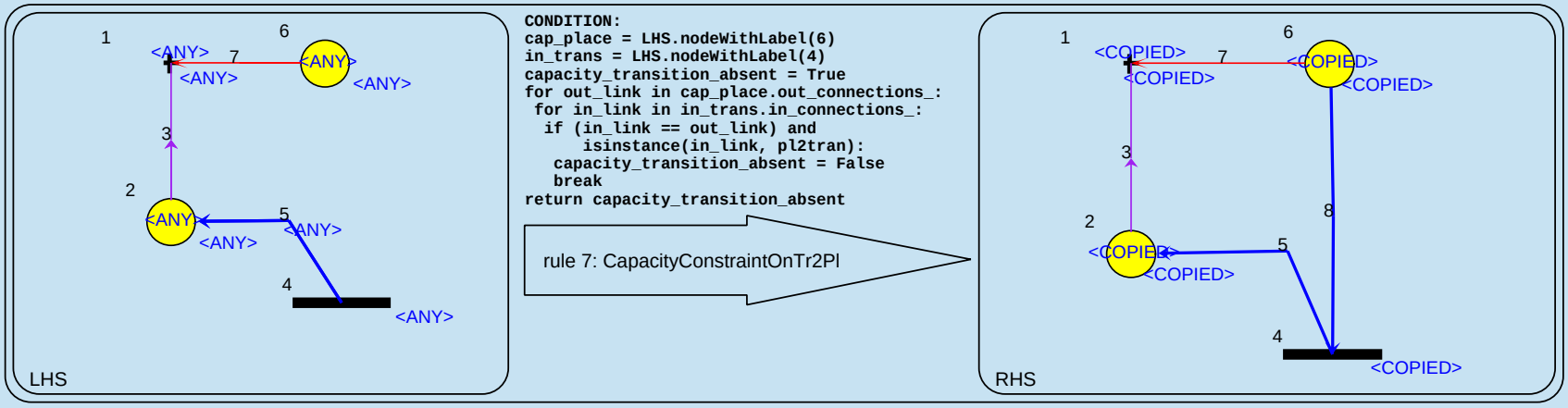
# **Traffic** to **Petri Net** Graph Grammar rules



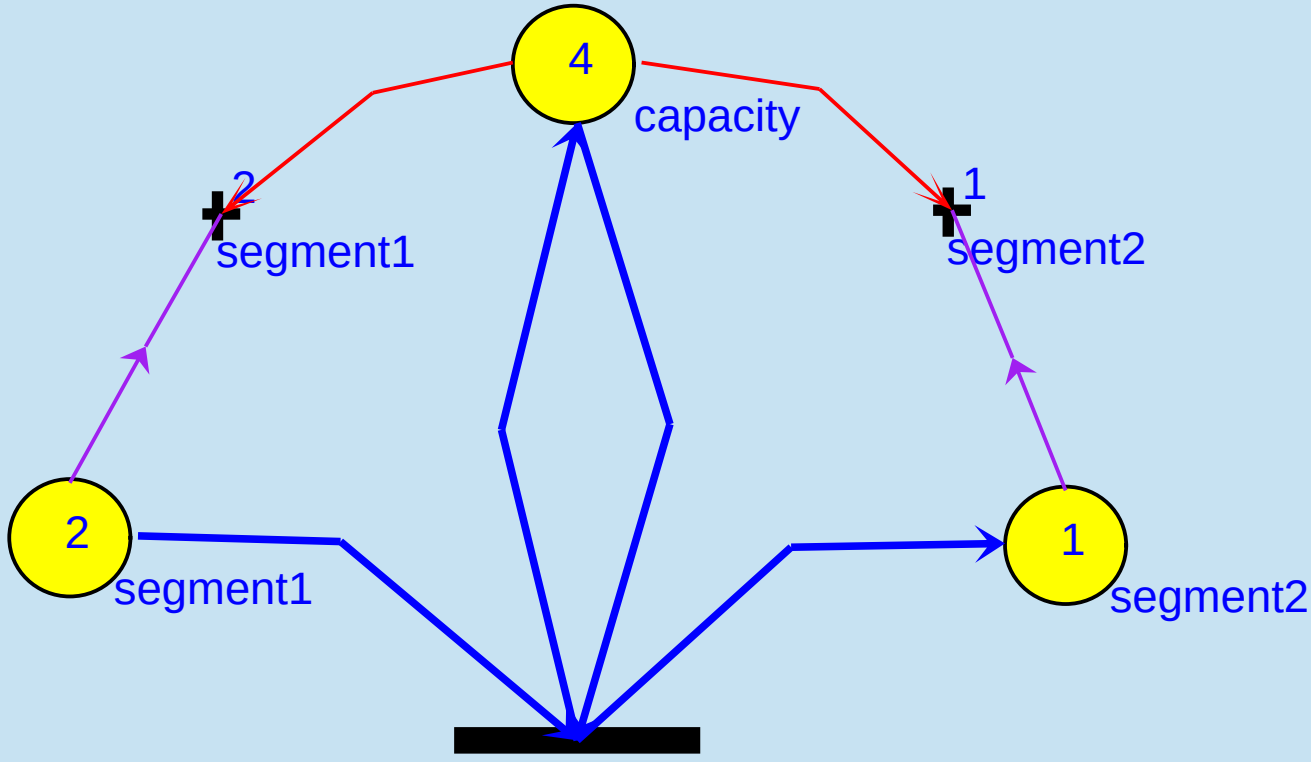
# Capacity Constraint on Place to Transition



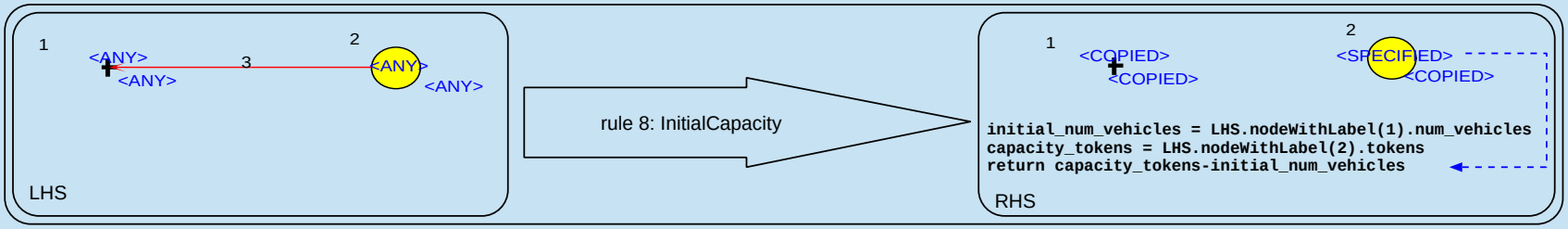
# Traffic to Petri Net Graph Grammar rules



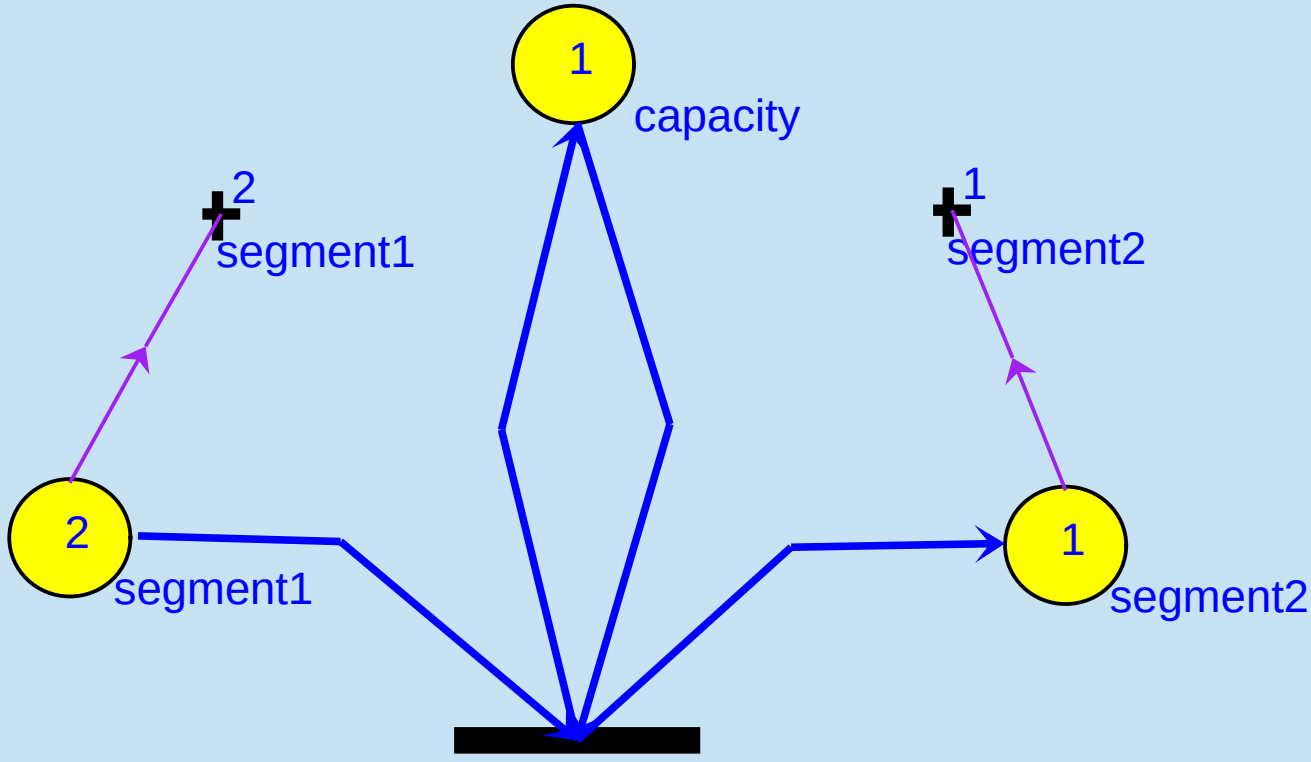
# Capacity Constraint on Transition to Place



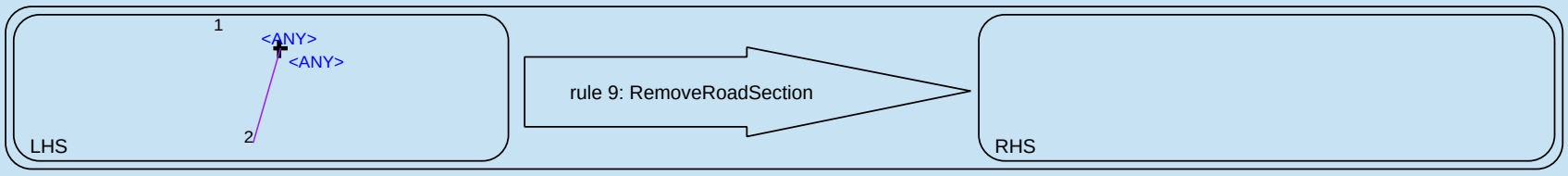
# **Traffic** to **Petri Net** Graph Grammar rules



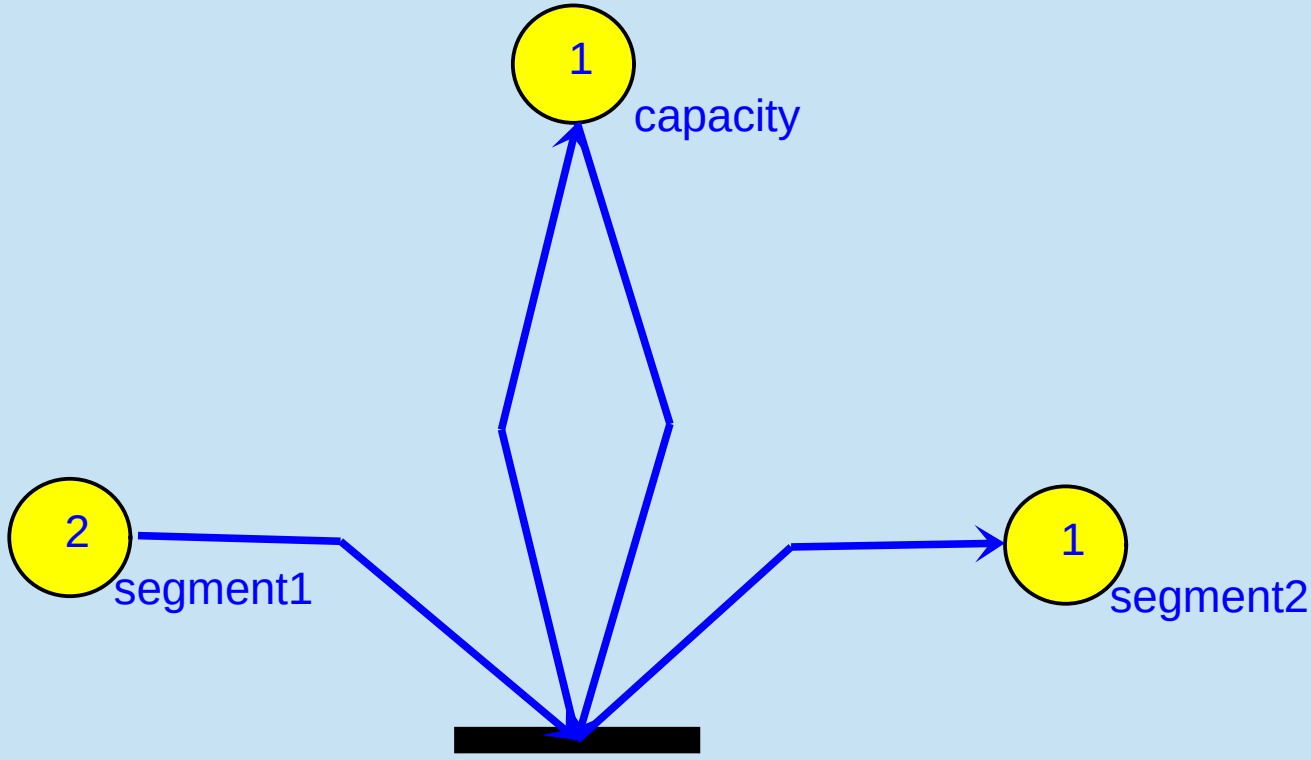
# Model Initial Capacity (applied rule twice)



# **Traffic** to **Petri Net** Graph Grammar rules



## Removed **Traffic** Road Section, now only **Petri Net**



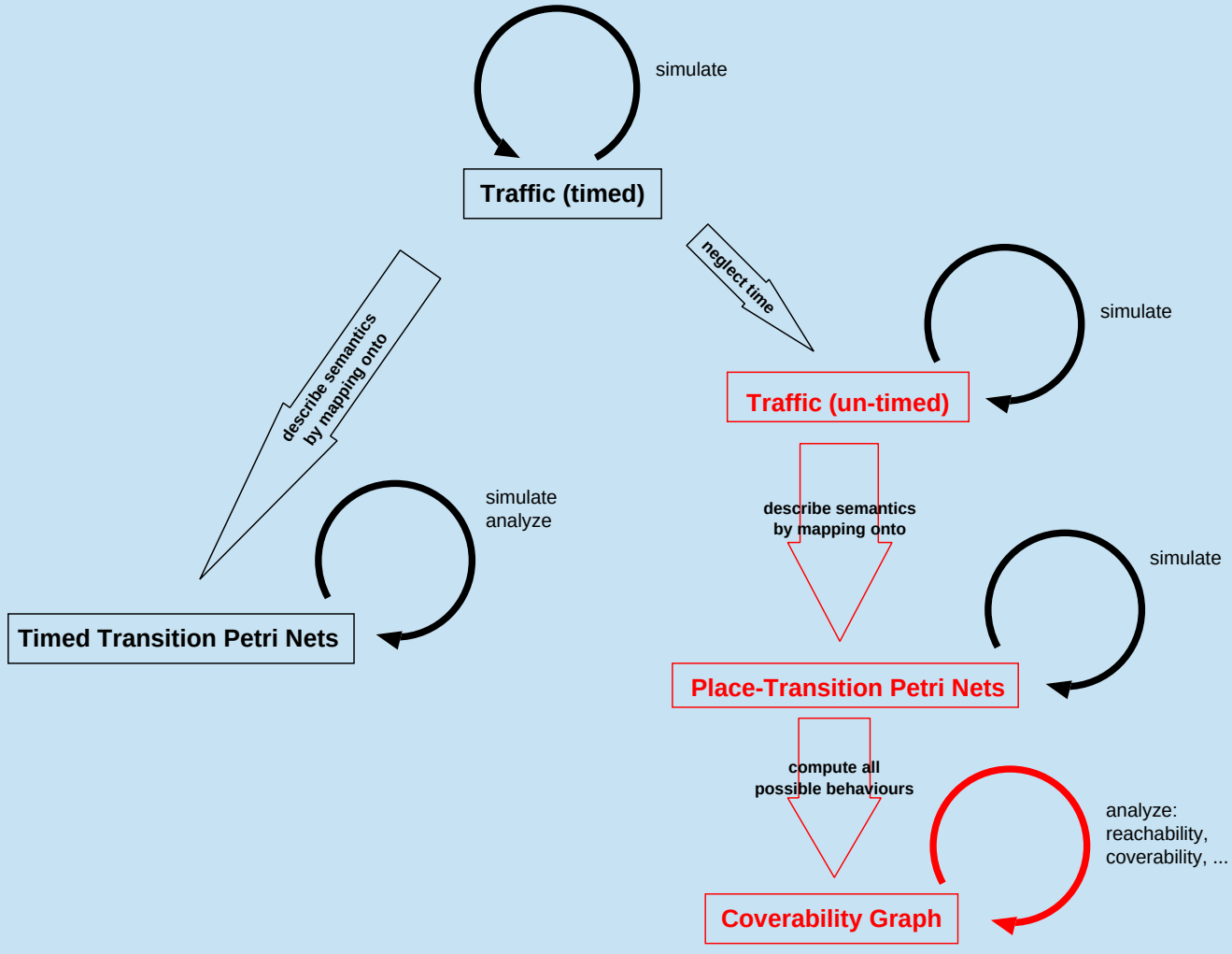
## Static **Analysis** of the Transformation Model

The transformation specified by the Graph Grammar model must satisfy the following requirements:

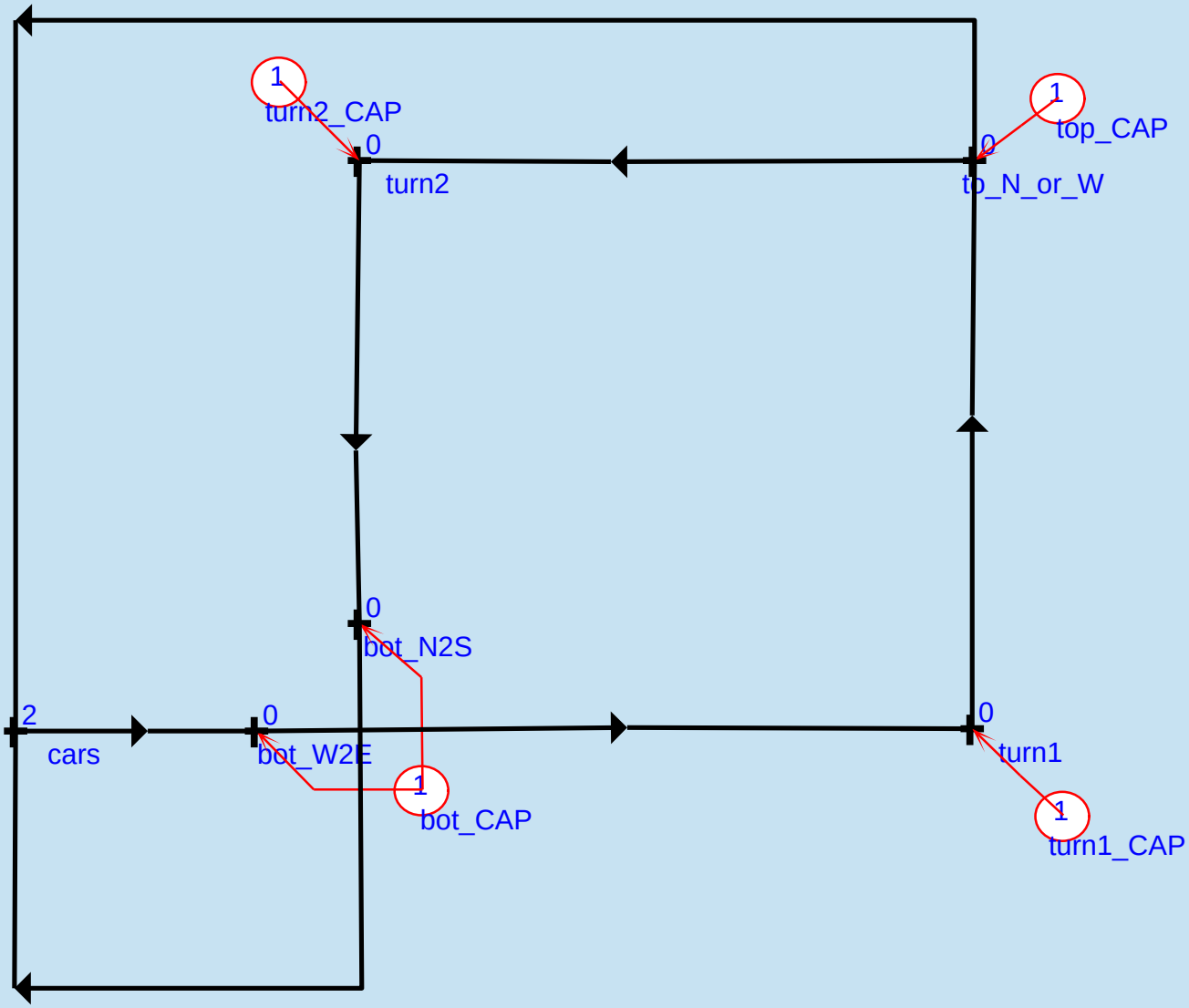
- **Termination:**
  the transformation process is *finite*
- **Convergence/Uniqueness:**
  the transformation results in a *single* target model
- **Syntactic Consistency:**
  the target model must be *exclusively* in the target formalism

These properties can often (but not always)
be **statically** checked/proved.

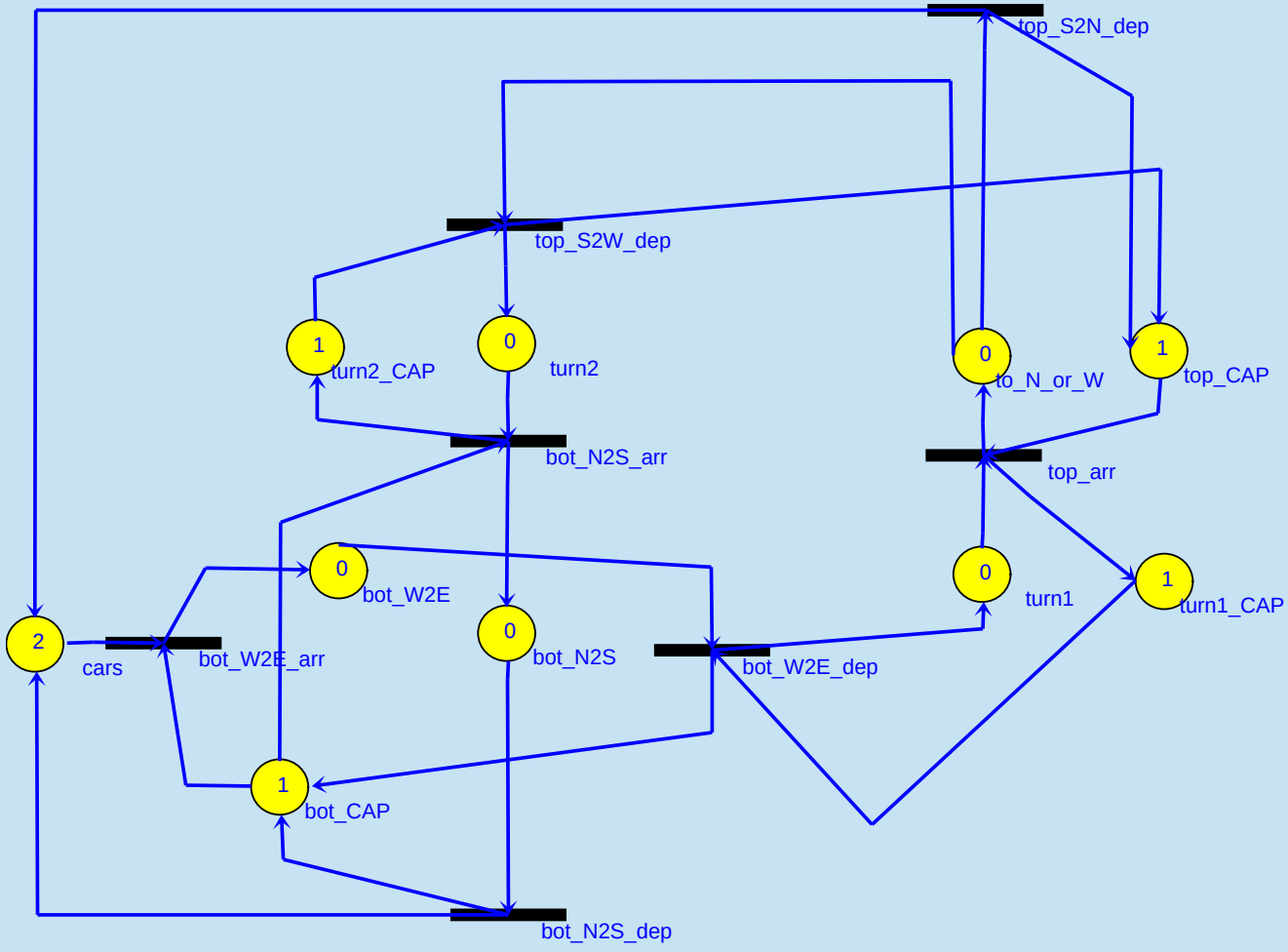
# Un-timed Analysis



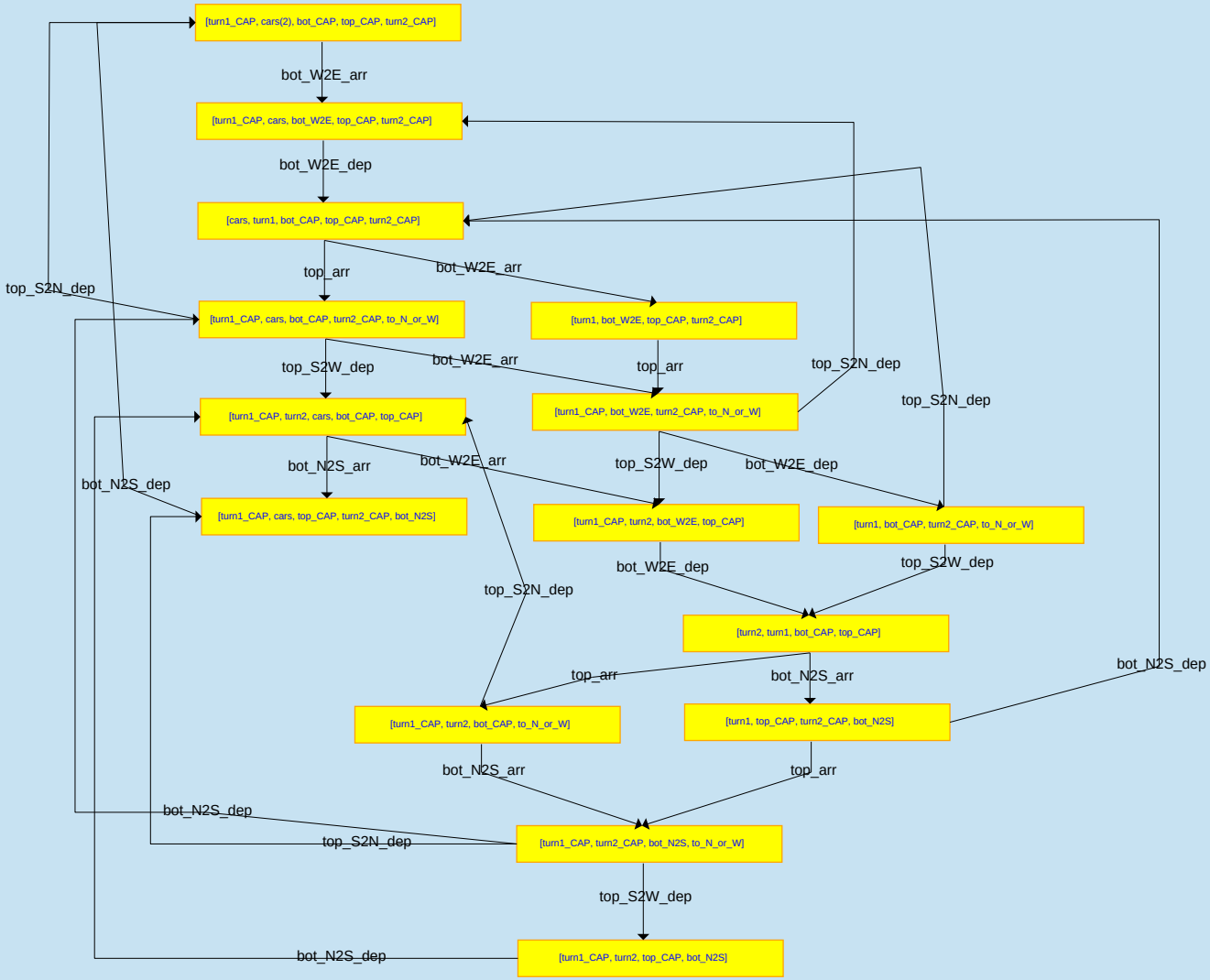
# An un-timed **Traffic** model



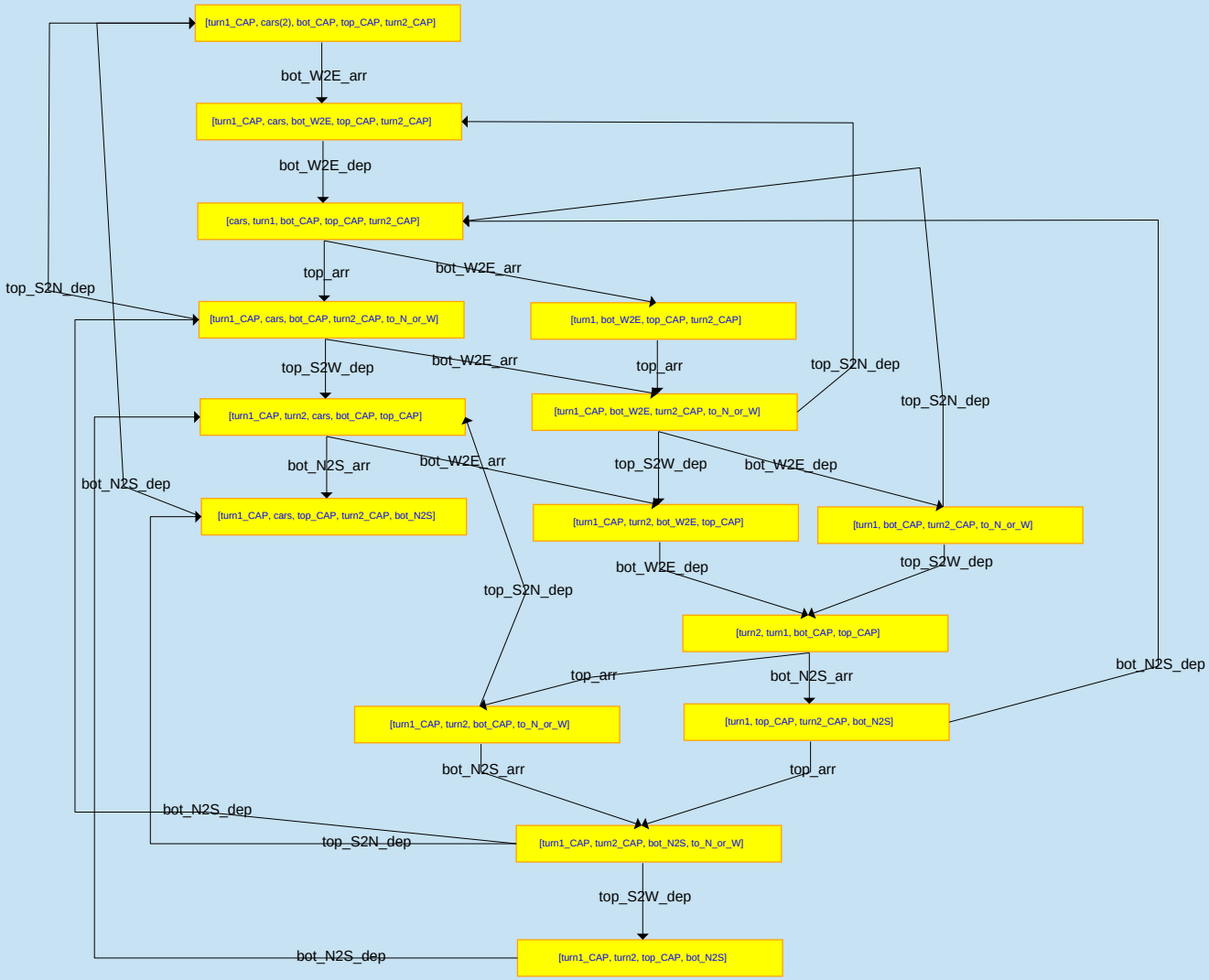
# the **Petri Net** describing its behaviour obtained by Graph Rewriting



# Analysis: Coverability Graph of the **Petri Net**



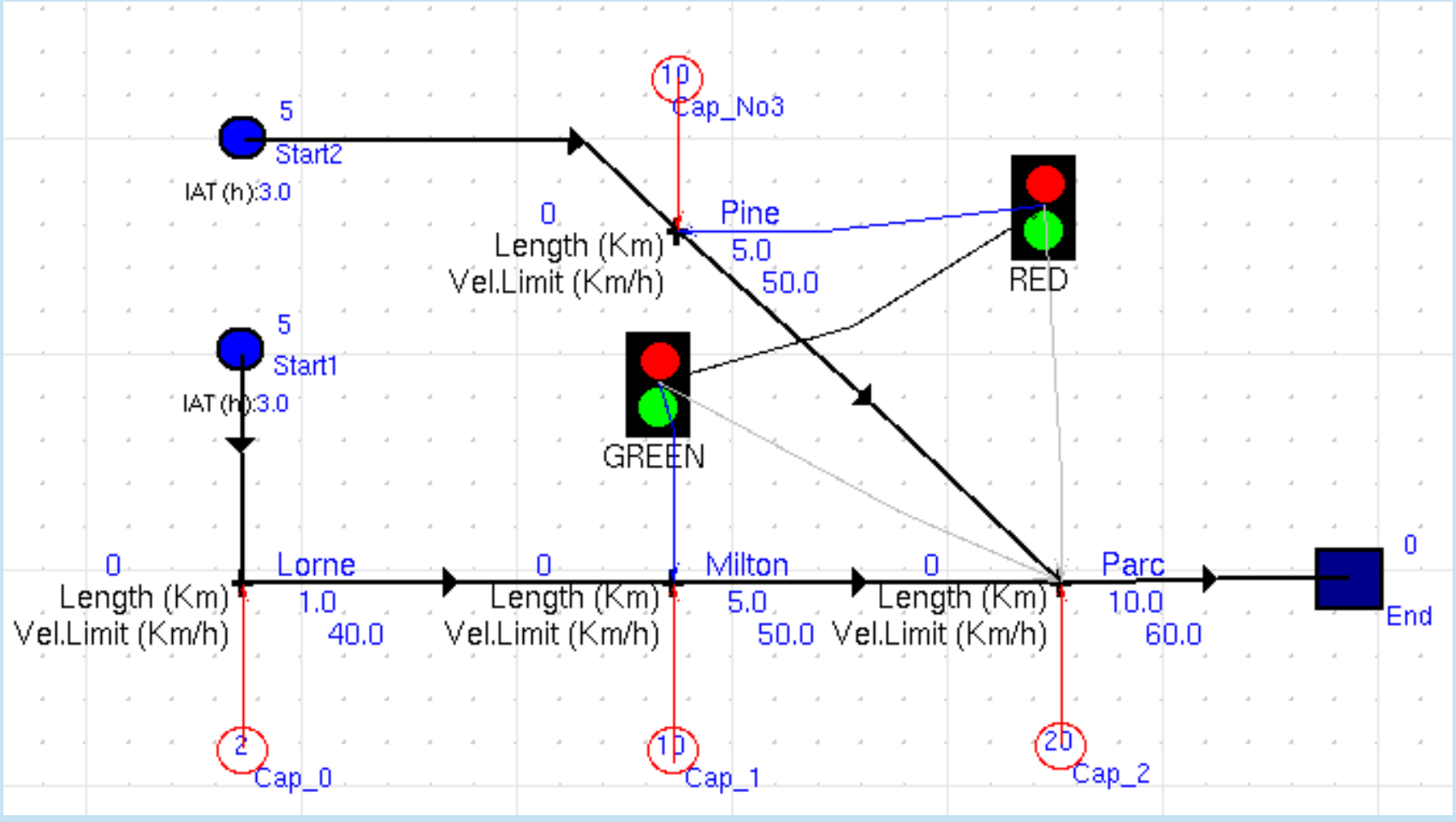
# Liveness Analysis



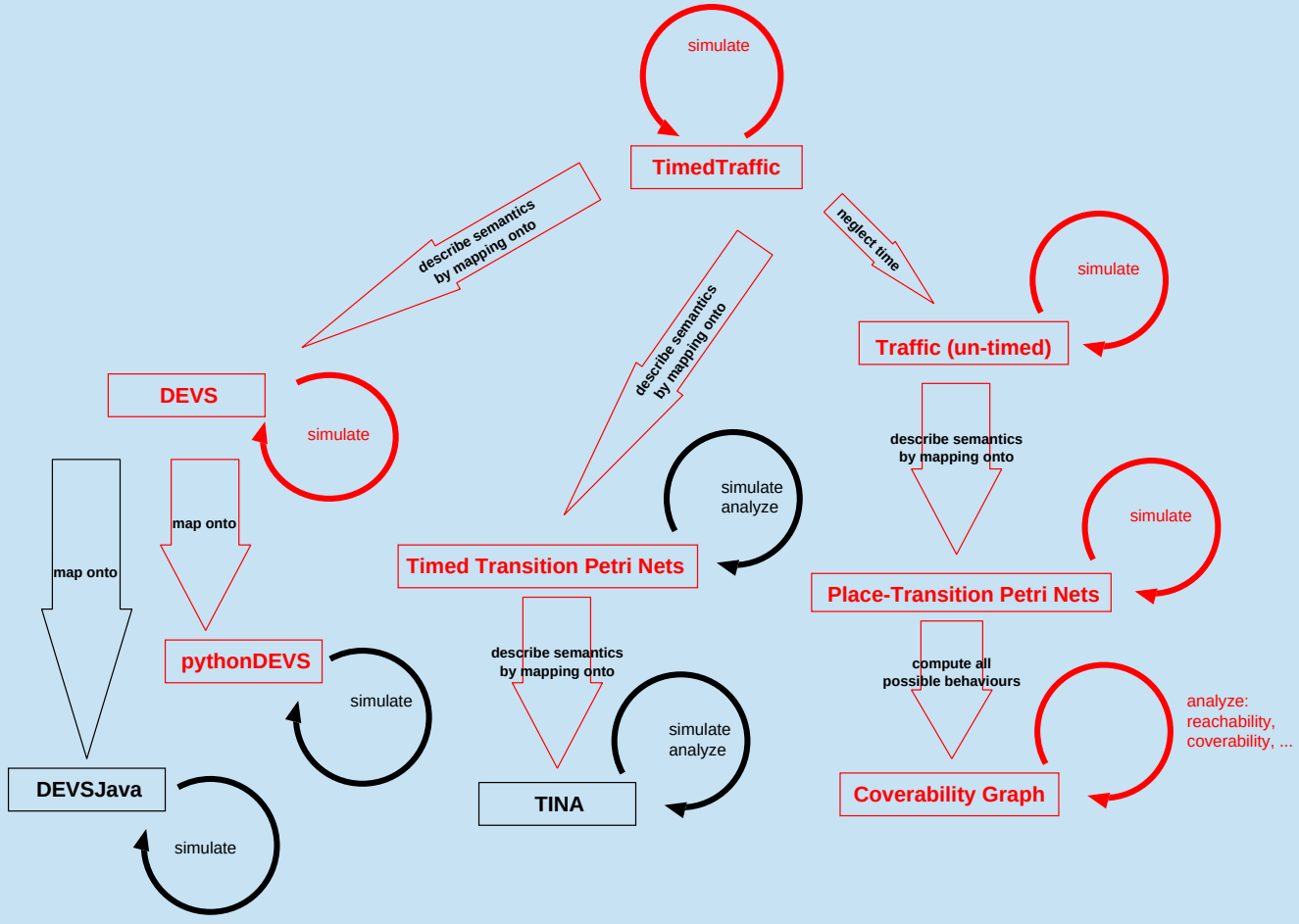
## Conservation Analysis

```
1.0 x[turn1_CAP] + 1.0 x[turn1] = 1.0

1.0 x[cars] + 1.0 x[bot_W2E] +
1.0 x[turn1] + 1.0 x[to_N_or_W] +
1.0 x[turn2] + 1.0 x[bot_N2S] = 2.0

1.0 x[top_CAP] + 1.0 x[to_N_or_W] = 1.0

1.0 x[turn2_CAP] + 1.0 x[turn2] = 1.0

1.0 x[bot_CAP] +
1.0 x[bot_W2E] + 1.0 x[bot_N2S] = 1.0
```

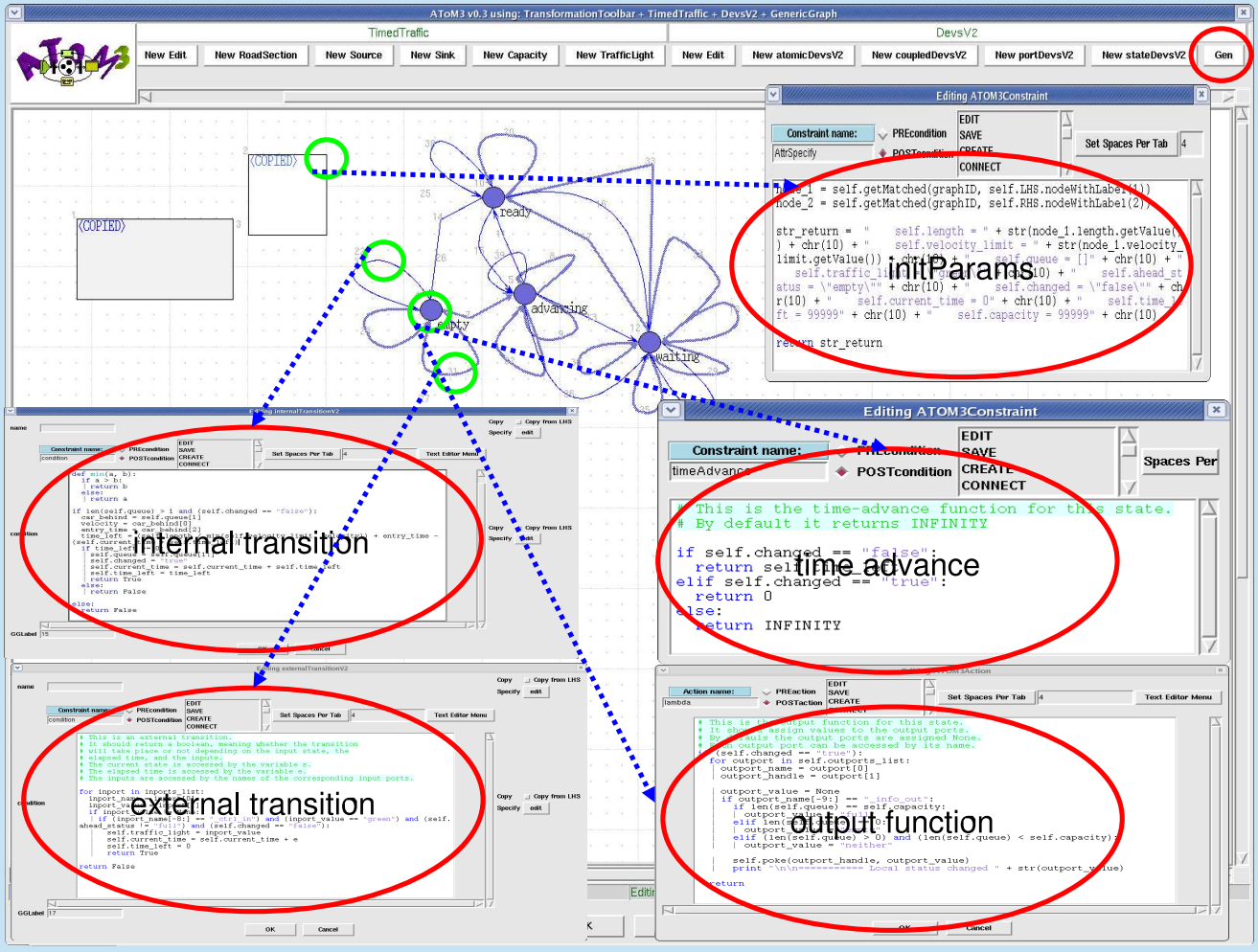
# Timed Traffic Network



# Mapping onto **DEVS** for Simulation (performance Analysis)



# **Traffic** mapped onto a **DEVS** model



## Conclusions

1. Through anecdotal evidence, demonstrated the usefulness of **Domain-Specific Visual Modelling** in the broad context of CAMPaM.
2. Demonstrated **feasibility** of rapidly and re-usably building Domain-Specific Visual Modelling, Analysis, Simulation tools using **meta-modelling** and **graph rewriting**.

**model everything !**

Jean Bézivin

Everything is a model !

Jean-Marie Favre

Nothing is a model !

Hans Vangheluwe

Model everything !