Alloy

An Introduction using Traffic Network Modelling

Sagar Sen
2nd year PhD Student, INRIA, Rennes, France
Venue: MSDL, McGill Univ., Canada
Date: 23/06/2008

Outline

- What is Alloy?
- Who develops it and why?
- An Example: Traffic
- Modelling Traffic in Alloy
- Synthesizing Traffic Networks in Alloy
- Verifying Properties of the Traffic Networks Specification
- Behind the Scenes
- Conclusion

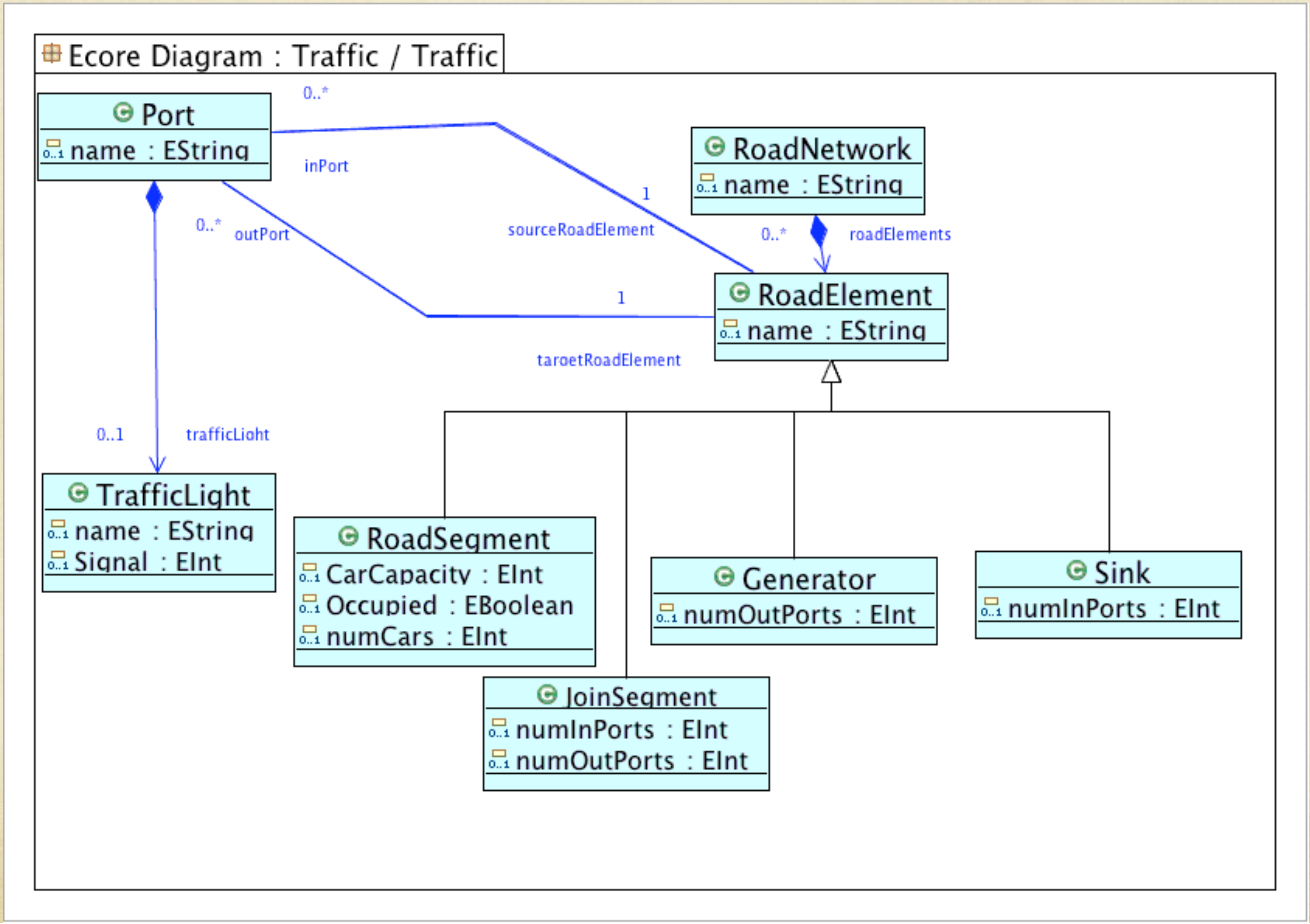
What is Alloy?

- Software Implementation of *first-order relational logic with quantifiers (FORLQ)*
- Declaratively specify a *set of instances (models in MDE)* as an Alloy Model (Meta-model in MDE)
- Transforms Alloy formulas (in FORLQ) of the Alloy Model to Boolean CNF
- Solves Boolean CNF using a satisfiability (SAT) solver to give one or more instances that conform to the initial Alloy Model
- Or, Solve Boolean CNF to give a counterexample instance that shows that an assertion does not hold true against an Alloy Model.

Who develops it and why?

- Software Design Group, MIT
- Founded by: Daniel Jackson
- Why develop Alloy despite the presence of NuSMV, Prolog, Z and numerous other software specification languages/tools ?
- Daniel Jackson envisioned a lightweight formal verification tool with well defined syntax and semantics to search for model instances with certain properties in a *finite scope*.
- This is the first tool that supports specification of *quantified constraints* on a set of objects and also a clean transformation to a Boolean satisfiability solver.
- Website: http://alloy.mit.edu/

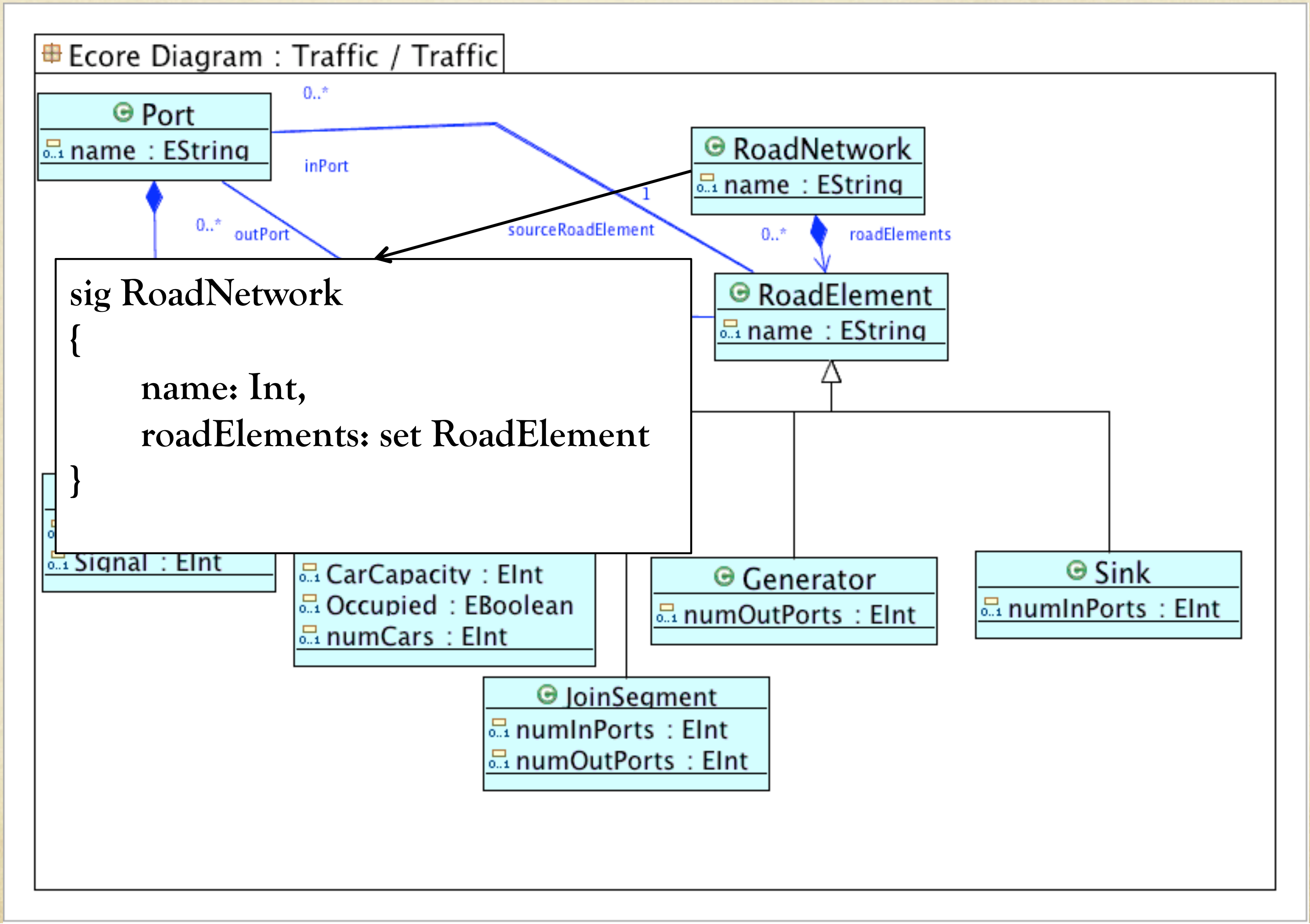
An Example: Traffic



Specifying Alloy Signatures for Classes

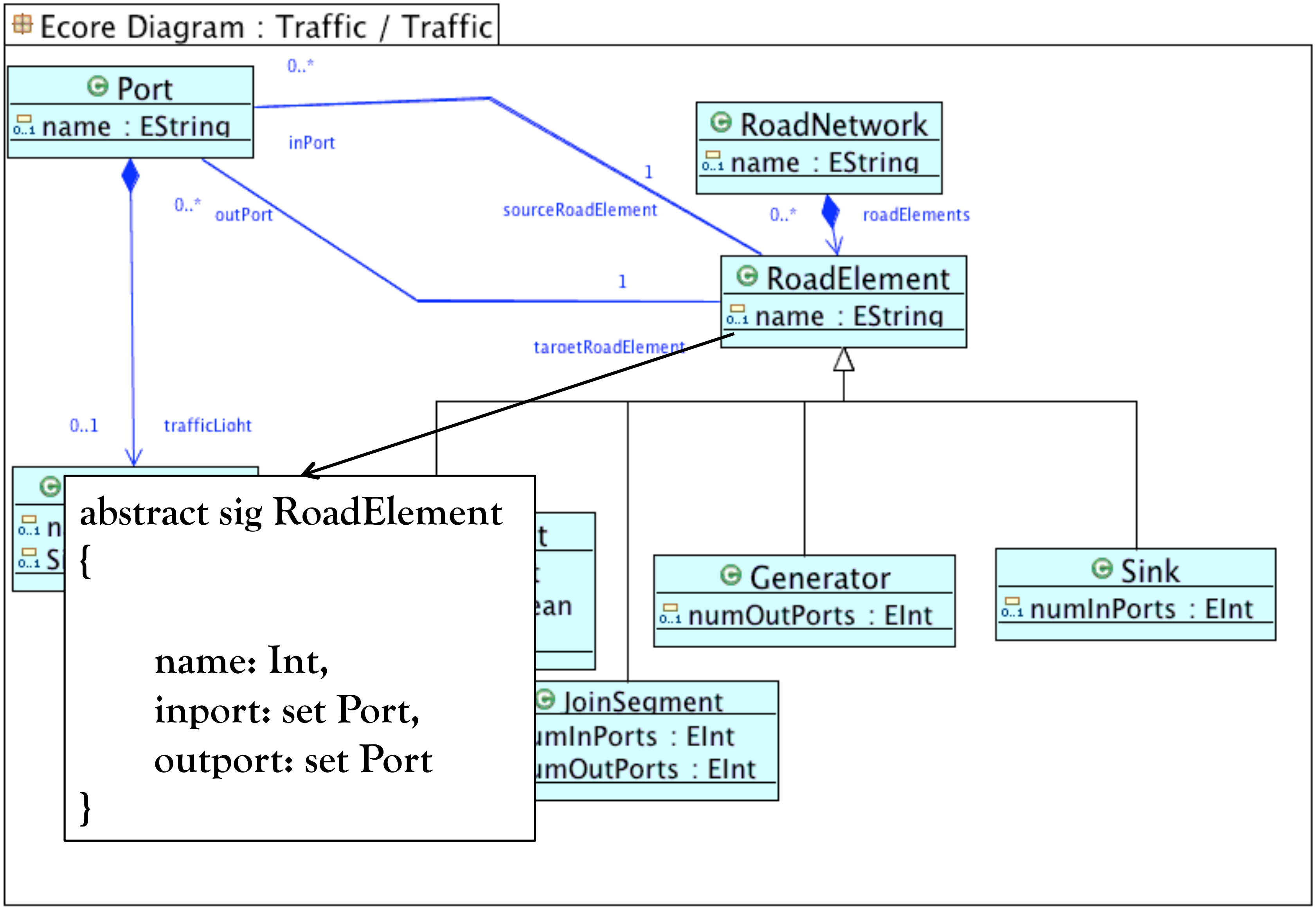
- An Alloy *signature* describes a class or set of *immutable atoms*.
- Signatures are used to build conceptual models of a world of objects.
- An instance of a *signature* is like an *object* that conforms to a Class.
- Lets transform the Traffic Network Classes to Signatures in Alloy…

An Example: Traffic



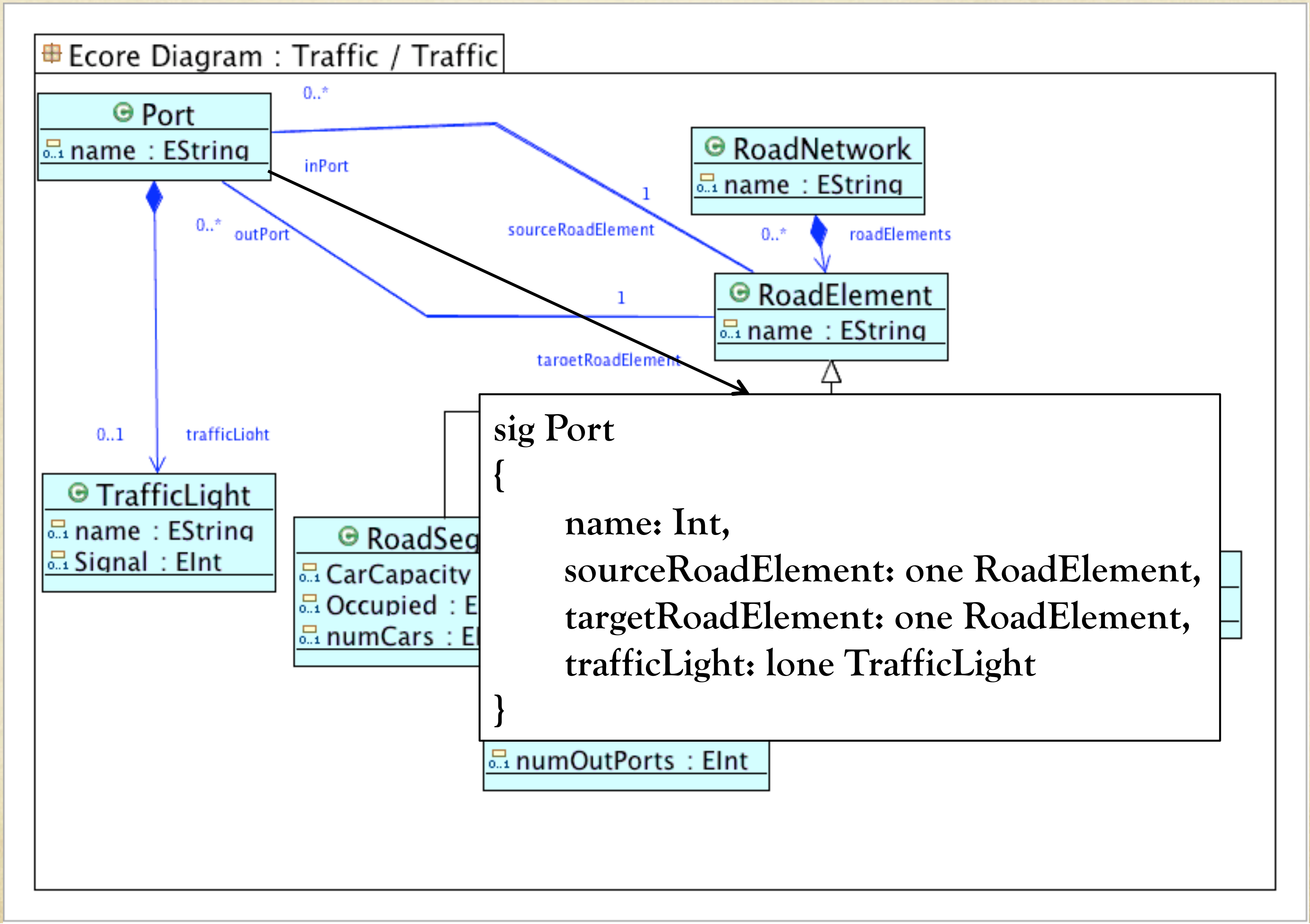

```
sig RoadNetwork
{
        name: Int,
        roadElements: set RoadElement
}
```

An Example: Traffic



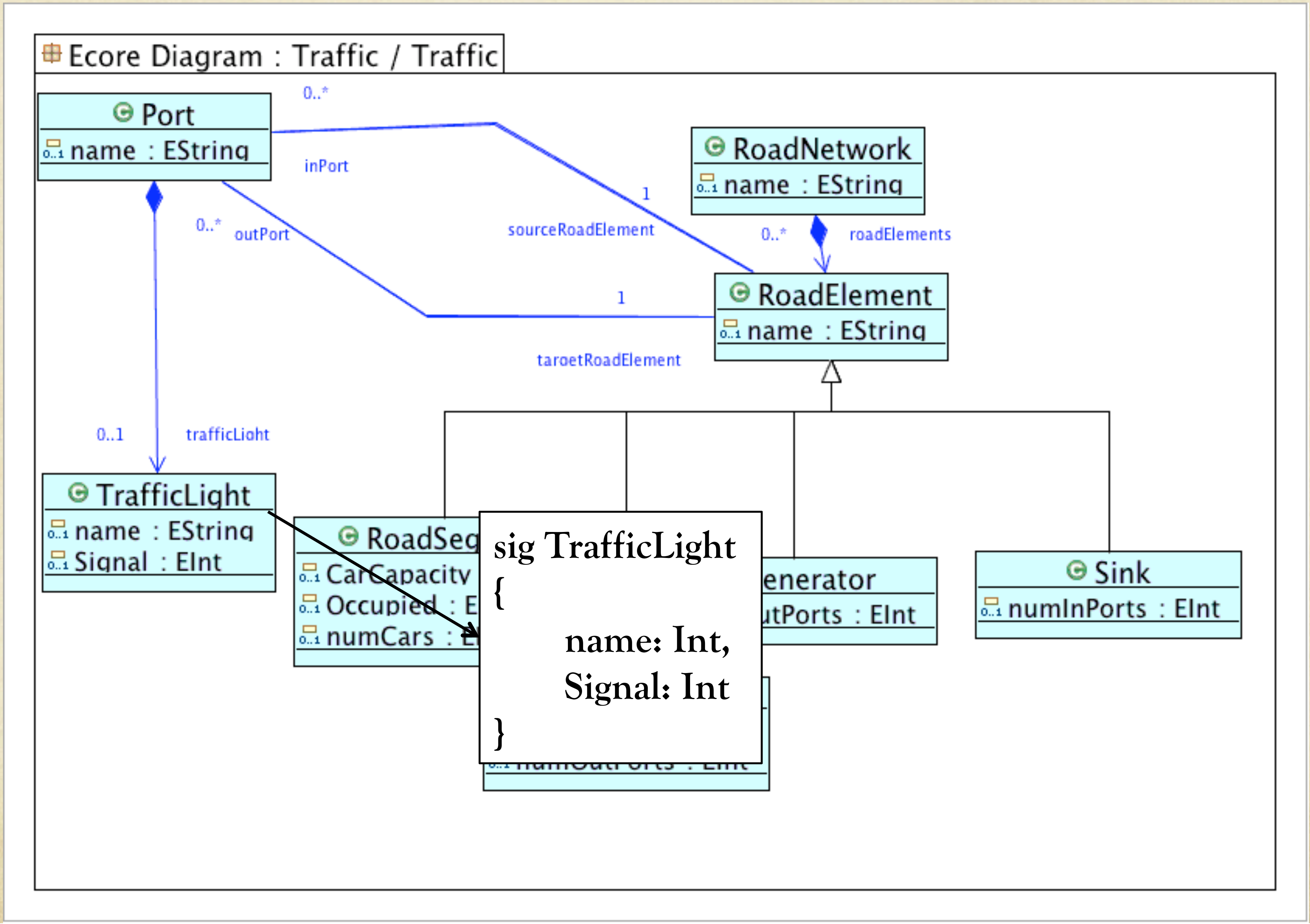

```
abstract sig RoadElement
{
        name: Int,
        inport: set Port,
        outport: set Port
}
```

An Example: Traffic

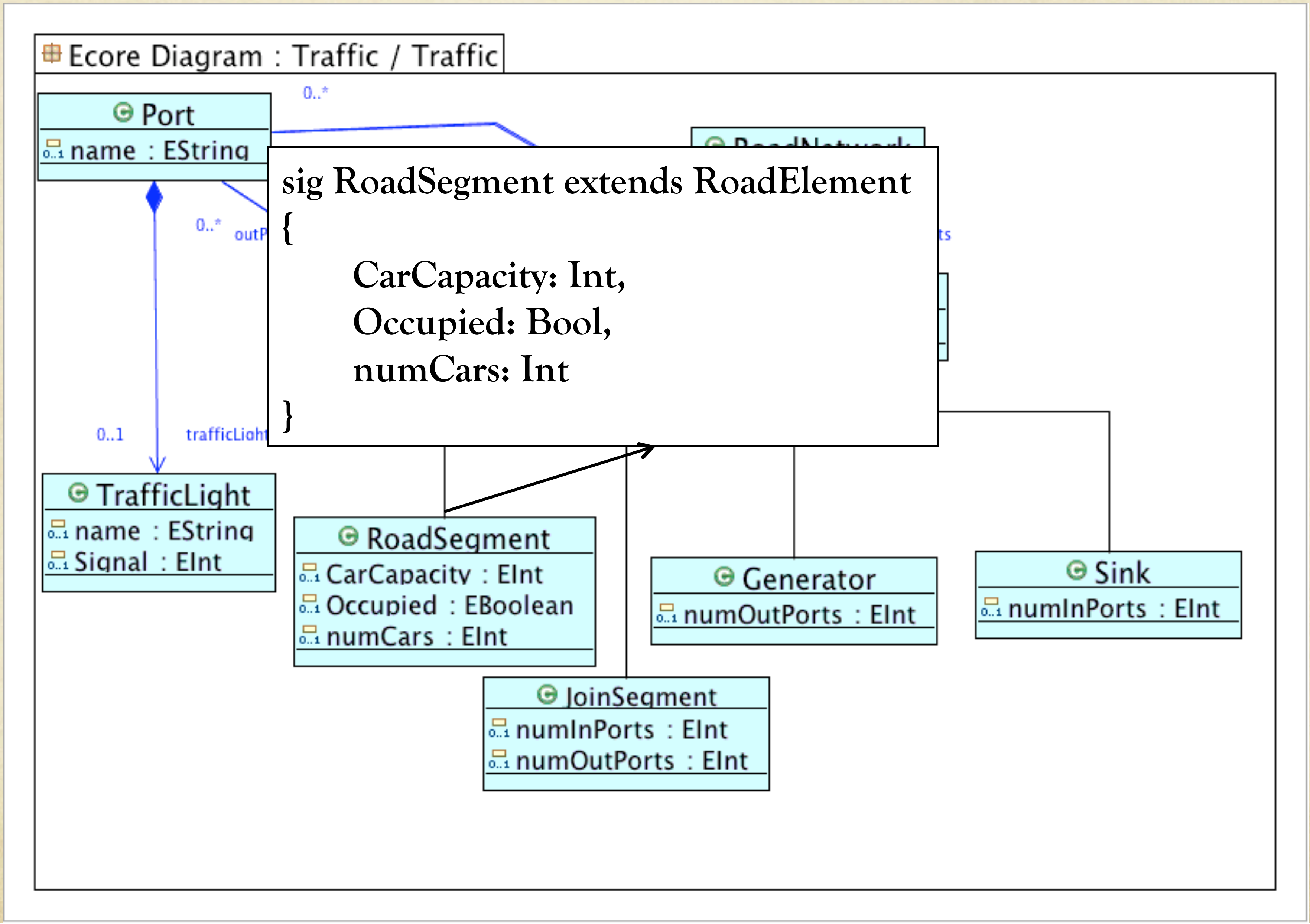



```
sig Port
{
    name: Int,
    sourceRoadElement: one RoadElement,
    targetRoadElement: one RoadElement,
    trafficLight: lone TrafficLight
}
```

An Example: Traffic

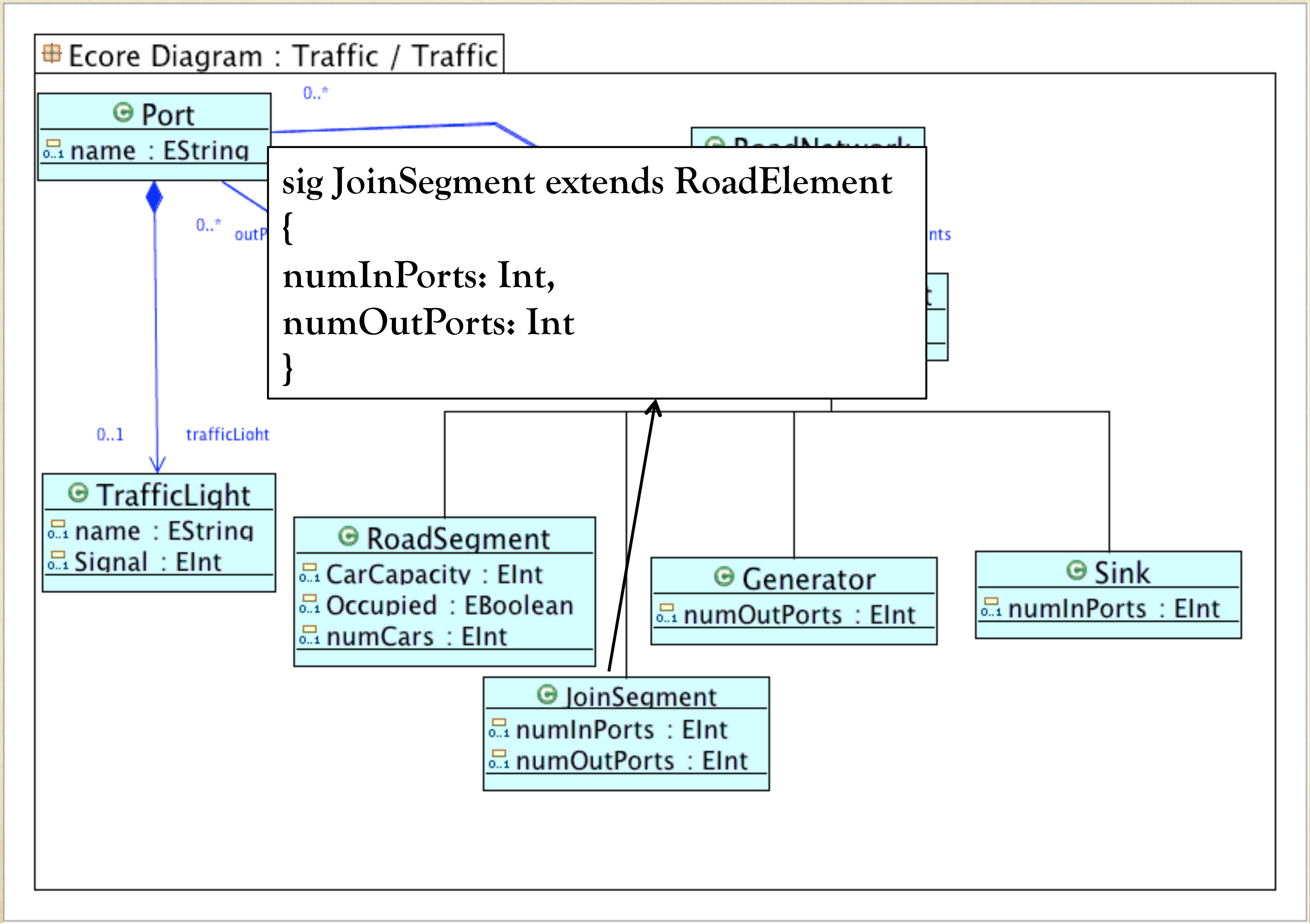


An Example: Traffic

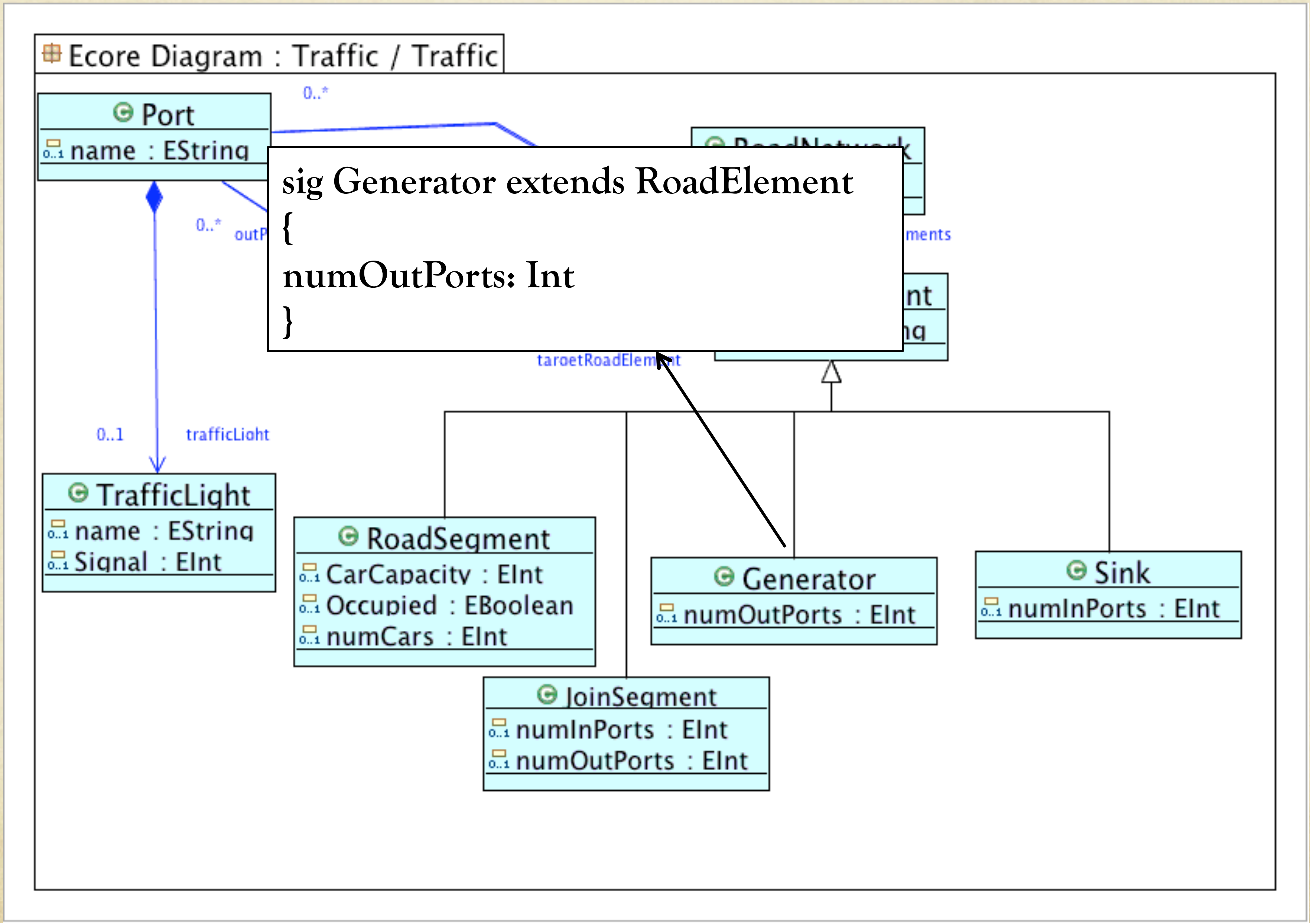



```
sig RoadSegment extends RoadElement
{
        CarCapacity: Int,
        Occupied: Bool,
        numCars: Int
}
```

An Example: Traffic

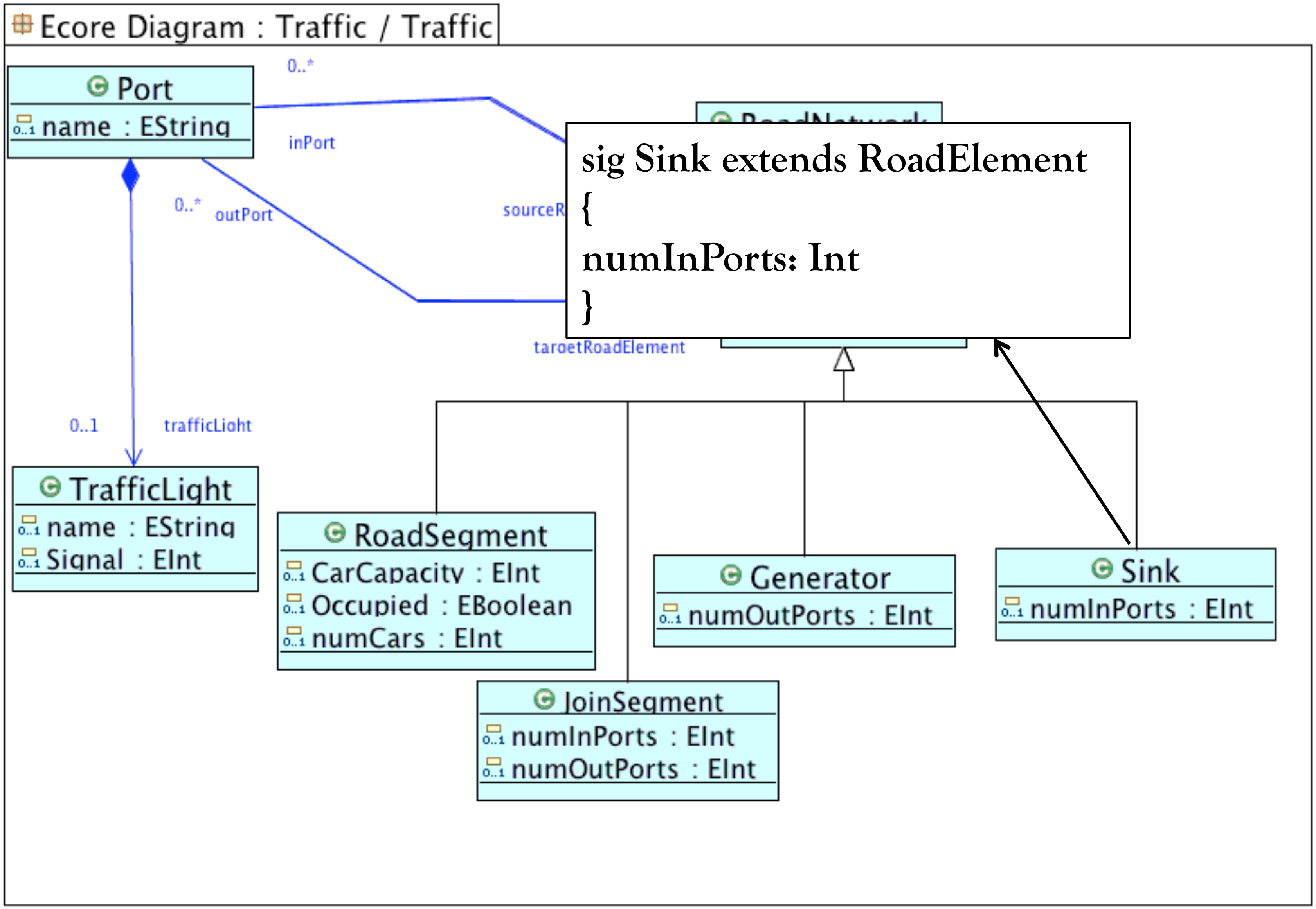


An Example: Traffic



sig Generator extends RoadElement
{
numOutPorts: Int
}

An Example: Traffic



Specifying Alloy Facts for Constraints

- A constraint that is *always* true in a domain-specific language is actually a *fact*.
- Hence, we transform all knowledge about the domain-specific language that are inexpressible as signatures to Alloy facts.
- An *instance* of the Alloy model containing signatures and facts is like an object diagram in MDE that satisfies the facts.
- Lets transform the Traffic Network Constraints to Facts in Alloy…

Containment Constraints

- All RoadElements are contained by one RoadNetwork:

```
fact containmentRoadNetwork
{
    all r:RoadElement| r in RoadNetwork.roadElements
}
```

- All TrafficLights are contained in a Port:

```
fact containmentTrafficLight
{
    all t:TrafficLight| t in Port.trafficLight
}
```

Facts on Road Networks

Exactly One Road Network

```
fact exactlyOneRoadNetwork
{
    one RoadNetwork
}
```

Facts on Road Elements

All Road Elements have unique names

```
fact uniqueNameRoadElements
{
    all r1:RoadElement, r2:RoadElement |
    r1!=r2 implies r1.name!=r2.name
}
```

Facts on Road Segments

A Road Segment has exactly one inport

```
fact roadSegmentInPort
{
    all r:RoadSegment | #r.inport = 1
}
```

A Road Segment must have exactly one outport

```
fact roadSegmentOutPort
{
    all r:RoadSegment | #r.outport =1
}
```

Facts on Join Segments

A Join Segment has numInPorts number of inports

```
fact joinInPort
{
    all j:JoinSegment | #j.inport = j.numInPorts
}
```

A Join Segment has numOutPorts number of outports

```
fact joinOutPort
{
    all j:JoinSegment | #j.outport = j.numOutPorts
}
```

Facts on Generators

A Generator Road Element has no inport

fact generatorInPort

{all g:Generator | #g.inport=0}

All Generators have at least one outport

fact generatorOutPortPositive

{all g:Generator | #g.outport>=1}

At Least One Generator in the Model

fact atleastOneGenerator

{#Generator >= 1}

A Generator Road Element has numOutPorts number of out ports

fact generatorOutPort

{all g:Generator | #g.outport=g.numOutPorts}

Facts on Sinks

All Sink elements have at least one inport

```
fact sinkInPortPositive
{all s:Sink | #s.inport>=1}
```

A Sink Road Element has no outport

```
fact sinkOutPort
{ all s:Sink | #s.outport=0 }
```

At Least One Sink in the Model

```
fact atleastOneSink
{ #Sink >= 1}
```

A Sink Road Element has numInPorts number of in ports.

```
fact sinkInPort
{ all s:Sink | #s.inport=s.numInPorts }
```

Facts on Sinks

All Sink elements have at least one inport

fact sinkInPortPositive

{all s:Sink | #s.inport>=1}

A Sink Road Element has no outport

fact sinkOutPort

{ all s:Sink | #s.outport=0 }

At Least One Sink in the Model

fact atleastOneSink

{ #Sink >= 1}

A Sink Road Element has numInPorts number of in ports.

fact sinkInPort

{ all s:Sink | #s.inport=s.numInPorts }

Facts on Ports

All Ports Unique Name

```
fact uniqueNamePorts
{
all p1:Port, p2:Port | p1!=p2 implies p1.name!=p2.name
}
```

Facts on Traffic Signals

A Traffic Signal can be Red, Yellow, or Green

```
fact trafficSignals
{
	all t:TrafficLight | t.Signal=1 or t.Signal=2 or t.Signal=3
}
```

Synthesizing Traffic Networks in Alloy

1. What we have ? : Alloy Model "Traffic.*als*" file

2. What does it contain ? : File contains the signatures and the facts that **declaratively specifies the Traffic Modelling Language**

3. We want to now see if we can actually **build traffic networks** that conform to this specification. Or, is the specification **correct and sufficient ?**

4. Lets look at the Alloy run command...

Synthesizing Traffic Networks in Alloy : Run Command(1)

1. We want to see if we can find an instance of Traffic in *finite scope.*

2. What is a scope ? : It is the *upper bound* on the number of atoms of each signature in the model (including integers).

3. Create an empty predicate and add it to Traffic.als

    pred testModel {}

4. Run command:

    run testModel for 20

5. Output is a Traffic instance that satisfies all facts up to a maximum of 20 atoms/signature.

Synthesizing Traffic Networks in Alloy: Run Command (2)

1. What is want to specify synthesis options?

2. Specifying an exact number of atoms:

   run testModel for exactly 20 RoadElement, exactly 20 Port, 5 TrafficLight, 5 int

3. Output is one or more instances containing exactly 20 road elements, 20 ports, 5 Traffic lights, and integers up to scope of 5.

Verifying Properties of the Traffic Networks Specification

- We want to see if an *assertion* about the Traffic Network Language is always True.
- Lets say: *All Ports have a Traffic Light*

```
assert AllPortsWithTrafficLights
{
all p:Port | #p.trafficLight=1
}
```

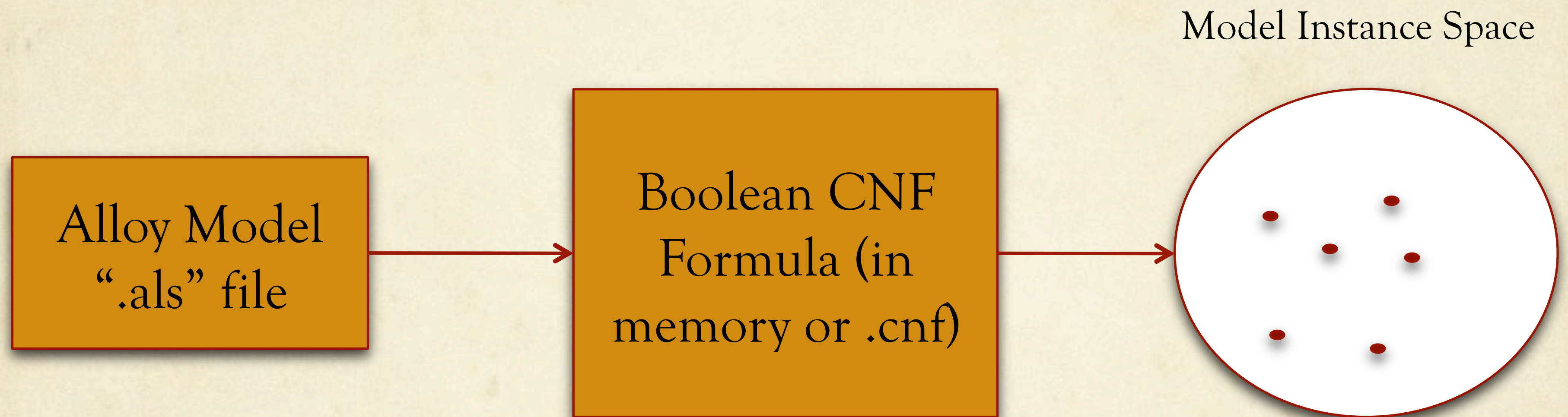
- We now run the *check* command for a scope of 20:

```
check AllPortsWithTrafficLights for 20
```

Verifying Properties of the Traffic Networks Specification

- However, a careful look at the Traffic MM reveals that a Port can have 0..1 TrafficLights
- The result of the check is now a *counterexample*
- *The counterexample* is a Traffic network with a Ports **without** TrafficLights.
- Such counter examples can be used to prove properties in specification for a finite scope.

Alloy: Behind the Scenes



KodKod Engine:
FORLQ to CNF

SAT Solvers such as…
SAT4J
ZChaff
BerkMin
transforms to Model Instance “.xml” file

Conclusion

- We looked at how *conceptual models* can be developed in Alloy
- *We have seen how we can use Alloy to synthesize instances that conform to the conceptual specification using the run command.*
- *We also use the check command to verifying assertions on an Alloy model.*
- *We finally show how Alloy works in the background.*