Building Effective Modelling Domains using Alloy

Sagar Sen
Equipe Triskell, INRIA, Rennes, France
19th November, 2008

Outline

1. Motivation

2. *Running Example : HFSM*

3. *Building Modelling Domains using Alloy*

3. *Signatures in Alloy*

4. *Facts in Alloy*

5. *Predicates and Functions in Alloy*

6. *Run Command in Alloy*

7. *Assert and Check Command in Alloy*

8. *Other features for Automation*

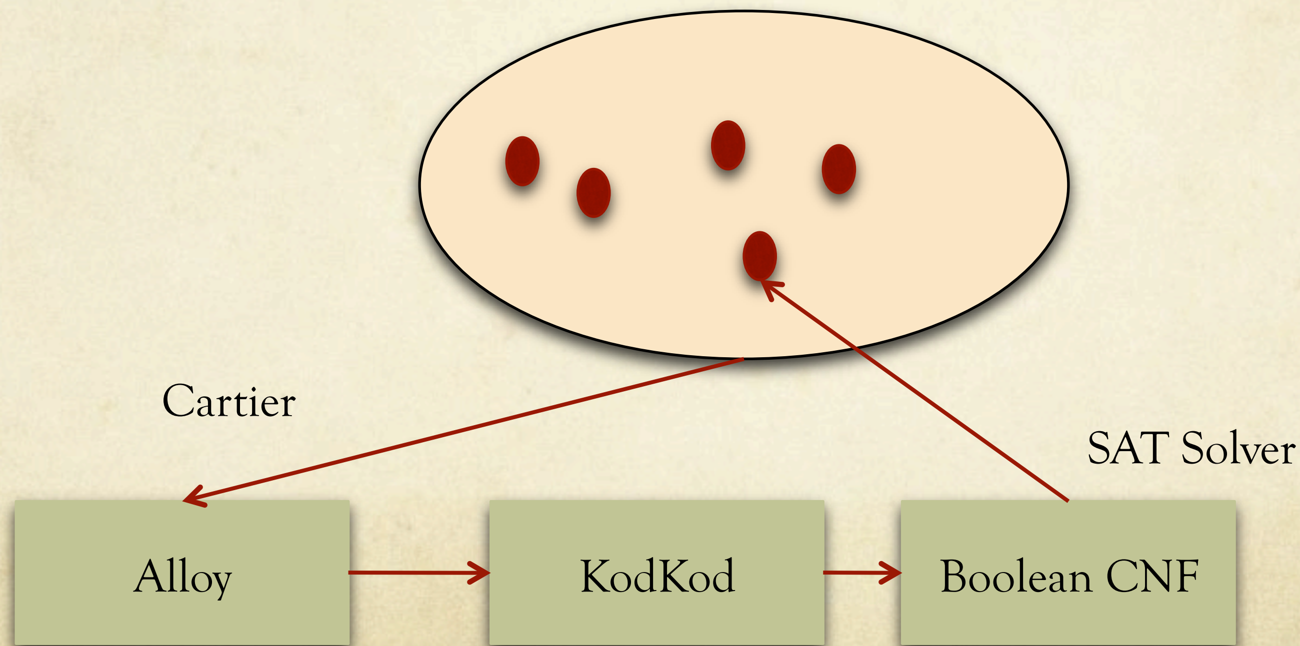
9. *Applications*

Motivation

- Today, software systems are engineered using complex graph structures called *models* based on principles of Model-driven Engineering (MDE)
- MDE is an emerging trend with need for tools to help design modelling languages and models.
- Problems in building such tools include: (a) Model Conformance (b) Model Synthesis (for testing, completion etc.) (c) Evolution of Specification Language (d) Improving Model Design (e) Maximum AUTOMATION

Building Modelling Domains using Alloy

- We want to solve the problems presented in motivation
- We propose the mapping of a modelling domain to declarative constraint specification language, Alloy.

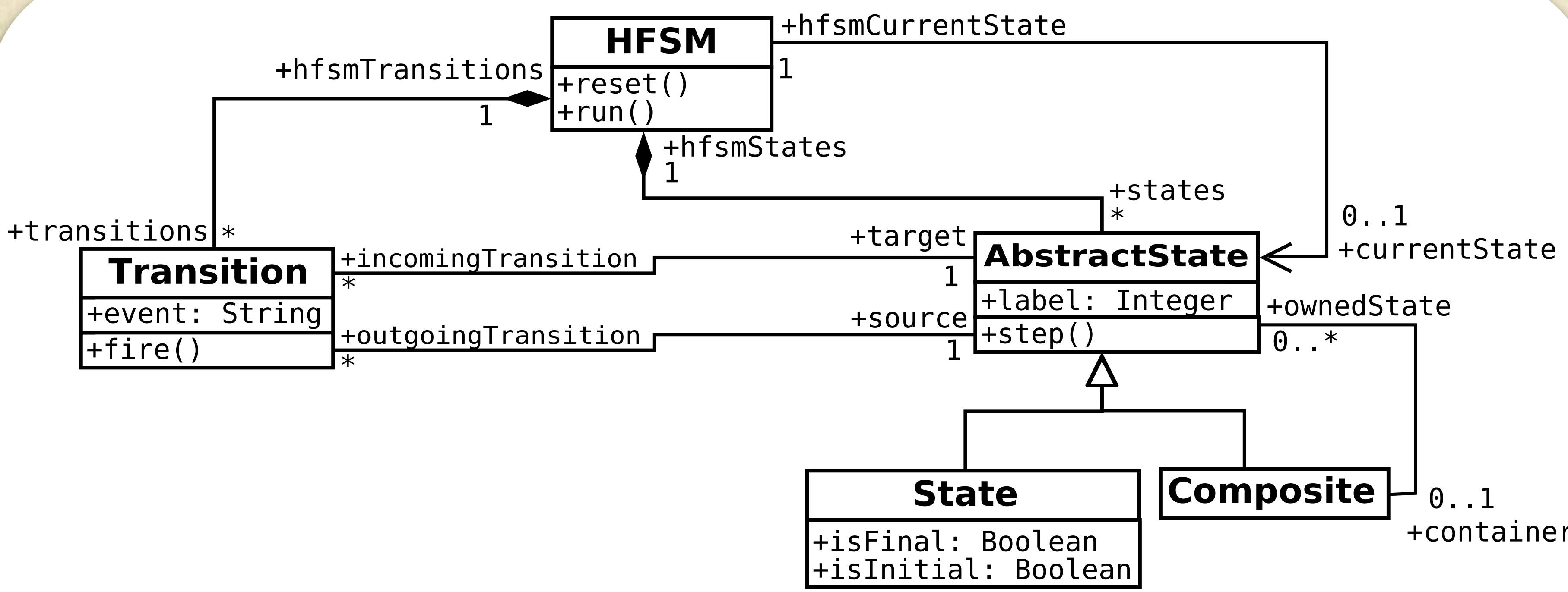


What is Alloy?

- 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.

Running Example

- Hierarchical Finite State Machine Modelling Domain

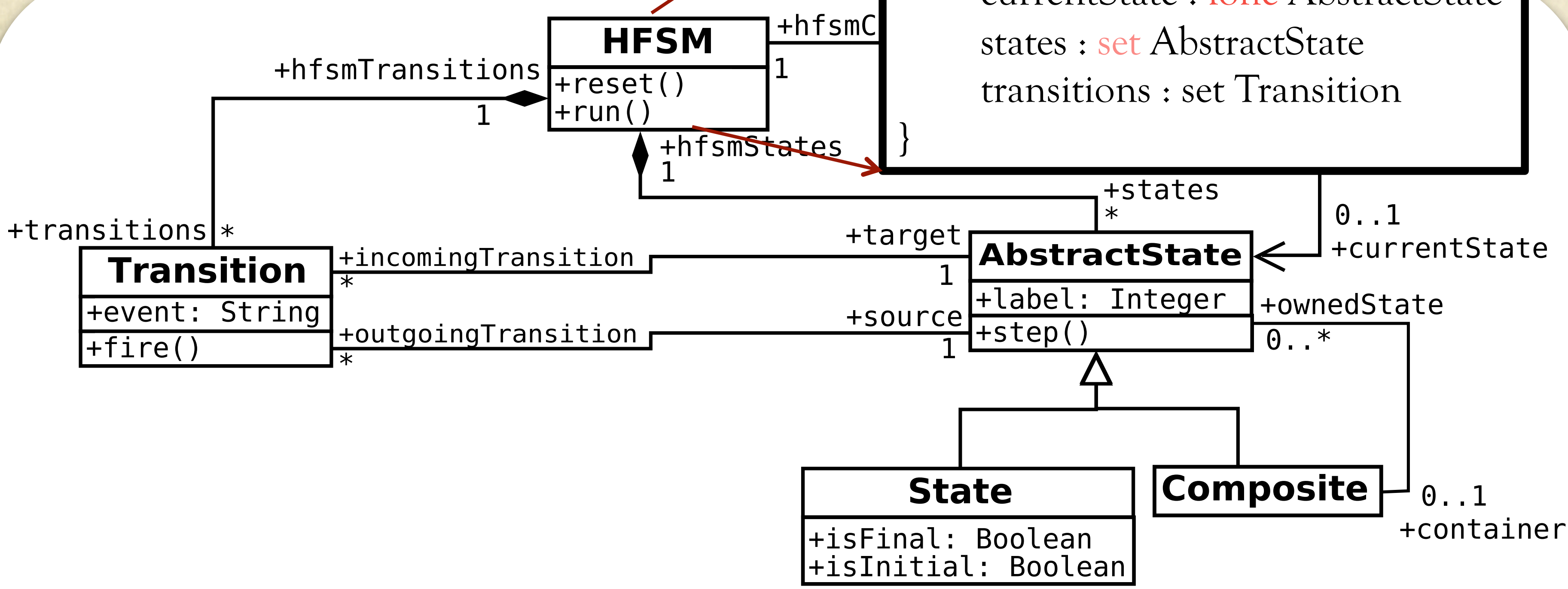


Alloy Signatures

To Specify Concepts

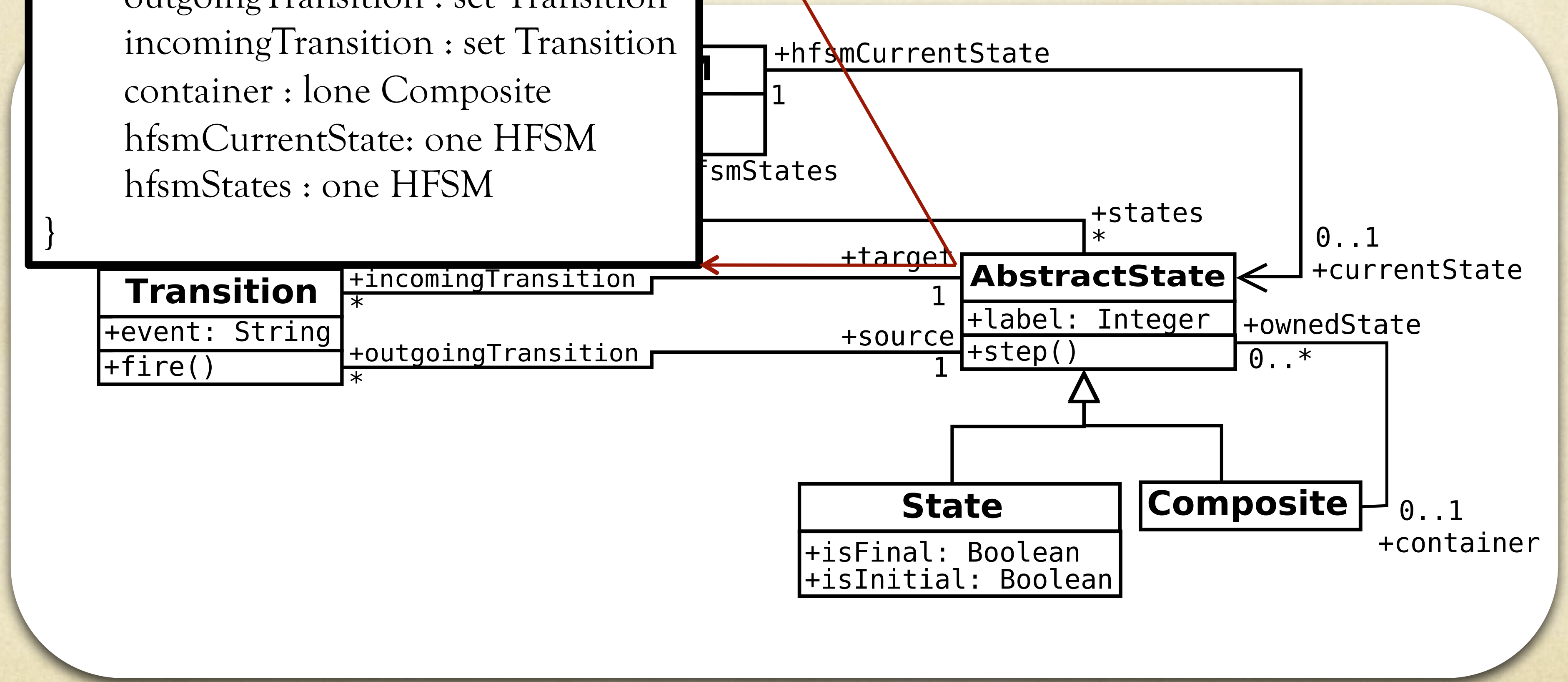
Signatures in Alloy

```
one sig HFSM
{
    currentState : lone AbstractState
    states : set AbstractState
    transitions : set Transition
}
```



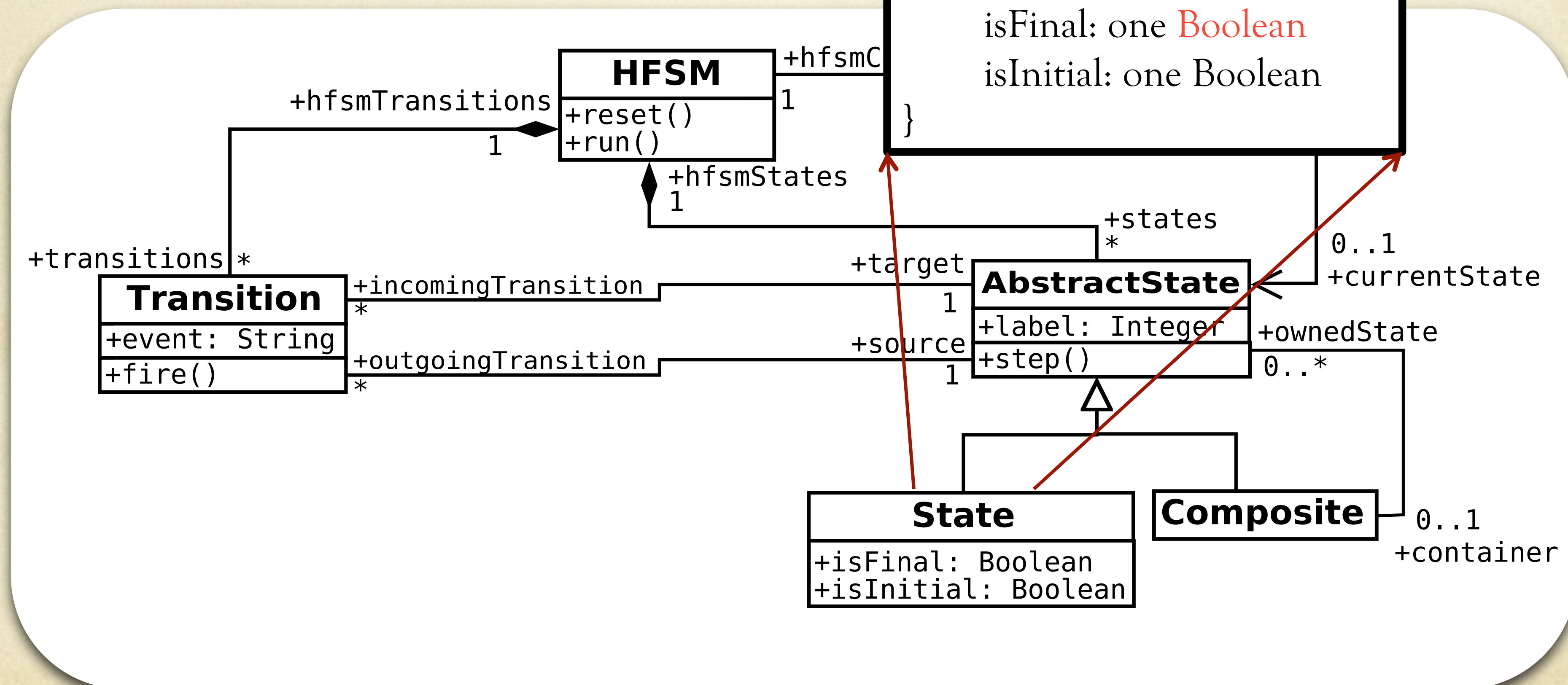
Signatures in Alloy

```
abstract sig AbstractState
{
	label : one Int
	outgoingTransition : set Transition
	incomingTransition : set Transition
	container : lone Composite
	hfsmCurrentState: one HFSM
	hfsmStates : one HFSM
}
```



Signatures in Alloy

```
sig State extends AbstractState
{
        isFinal: one Boolean
        isInitial: one Boolean
}
```



Alloy Facts

To Specify Constraints on Concepts

Facts in Alloy

Example : Containment Constraints

```
fact HFSM_states_composite
{
all p: State | p in HFSM.states and          <- Quantifier Expression
all owningClassObject1 : HFSM, owningClassObject2 : HFSM |
all property1 : HFSM.states, property2:HFSM.states |
property1 = property2 implies owningClassObject1 = owningClassObject2
}

fact HFSM_transitions_composite
{
all p: Transition | p in HFSM.transitions and
all owningClassObject1 : HFSM, owningClassObject2 : HFSM |
all property1 : HFSM.transitions, property2:HFSM.transitions |
property1 = property2 implies owningClassObject1 = owningClassObject2
}
```

Facts in Alloy

Example : Exactly One Initial State and At least one Final State

```
fact exactlyOneInitialState {
	one s:State|s.isInitial == True
}

fact atleastOneFinalState {
	some s:State | s.isFinal == True
}
```

Facts about Alloy Facts

- A fact is ALWAYS true in a model
- A fact contains expressions such as quantifier expressions, binary expression, compare expressions etc.
- A quantifier expression is used specify properties on a set of objects
- Allowed quantifiers are : all, some, one, and lone

Alloy Predicates

To Specify On/off Constraints on Concepts

Predicates in Alloy

Example : At least 2 Composite States

```
pred atleast2Composite()

{
#Composite > 2
}
```

A predicate can be satisfied when desired.

Alloy Functions

Named Expressions

Functions in Alloy

Example : Number of states in a Composite State is a functions as named expression

```
//
fun numberOfStates(composite:Composite): one Int
{
  #composite.ownedStates
}
```

Using a function value in a predicate:

```
pred atleastTwoAbstractStatesInComposite
{
numberOfStates[Composite]>2
}
```

Alloy Run Command

To Generate Instances

Run Command in Alloy

- Generating instances conforming to an Alloy model

Scope (Up to N objects for a signature)

pred example { ...}
run example for 20

Exact numbers and scope

run example for exactly 10 State, exactly 10 Composite, 1 HFSM, exactly 20 Transition

Output:

Alloy instance if all facts and called predicates are satisfied.

Alloy Assertions and Check Command

To Generate Counterexamples

Verifying Properties using Assert and Check

- Lets see if an Alloy model contains a Composite State with itself.

```
assert compositeStatedoesnotContainItself
{
all c:Composite | c not in c.ownedStates
}

check compositeStatedoesnotContainItself
for 20
```

We include a fact to avoid this... (Improving Spec.)

```
fact compositeCannotContainItself
{
all c1:Composite, c2:Composite | c1  = c2 => c2
not in c1.ownedStates and c1 not in
c2.ownedStates

}
```

Alloy Other Automation

Features

Alloy API

1. API based parsing of Alloy model, execution of multiple run commands for generation of models

2. Setting of different types of SAT solvers: Zchaff, MiniSAT, Berkmin etc. for solving resulting Boolean CNF

Applications

- Test Model Generation for Model Transformation Testing, Service testing etc.
- Completion of Partially Specified Models
- Improving Model Design ?