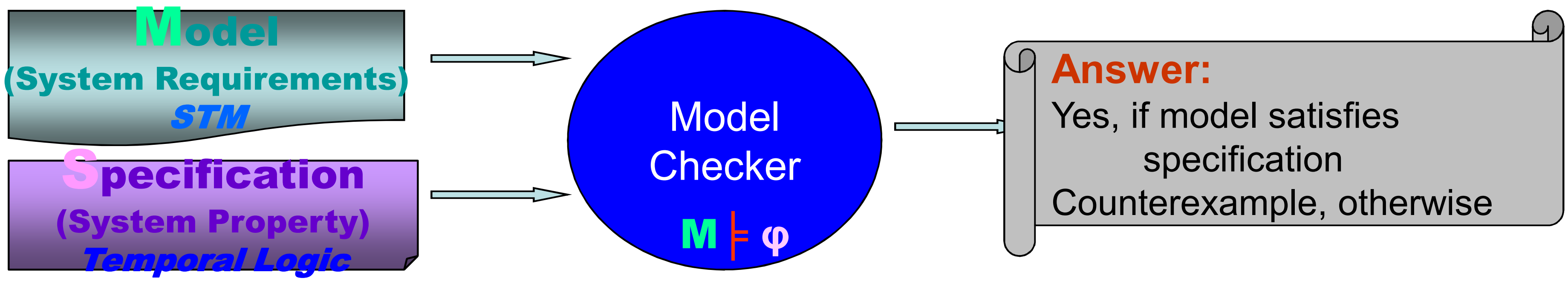
Model Finder

Comparison with Model Checking

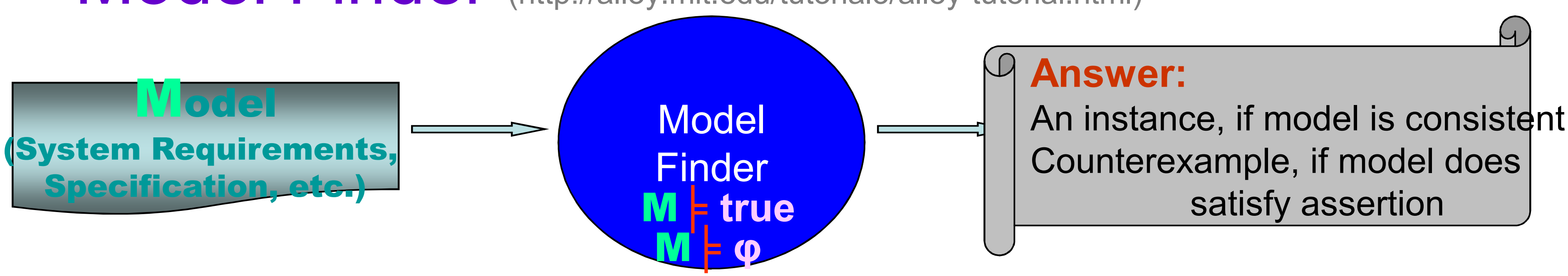
- Model Checking



For increasing our confidence in the correctness of the model:

- ❑ Verification: The model satisfies important system properties
- ❑ Debugging: Study counter-examples, pinpoint the source of the error, correct the model, and try again

- Model Finder (http://alloy.mit.edu/tutorial3/alloy-tutorial.html)



Characteristics of Alloy

- **finite scope check** – analysis of the model needs a scope (size).
  The analysis is *sound* (it never returns false positives) but *incomplete* (since it only checks things up to a certain scope). However, it is *complete up to scope*; it never misses a counterexample which is smaller than the specified scope. Small scope checks are still extremely valuable for finding errors.
- **infinite model** - The models in Alloy do not reflect the fact that the analysis is finite. That is, you describe the components of a system and how they interact, but do not specify how many components there can be (as is done in traditional "model checking").
- **declarative** - a declarative modeler answers the question "how would I recognize that X has happened", as opposed to an "operational" or "imperative" modeler who asks "how can I accomplish X".
- **automatic analysis** - unlike some other declarative specification languages (such as *Z* and *OCL*, the object language of *UML*), Alloy can be automatically analyzed. You can automatically generate examples of your system and counterexamples to claims made about that system.
- **structured data** - Alloy supports complex data structures, such as trees, and thus is a rich way to describe state

A Walkthrough

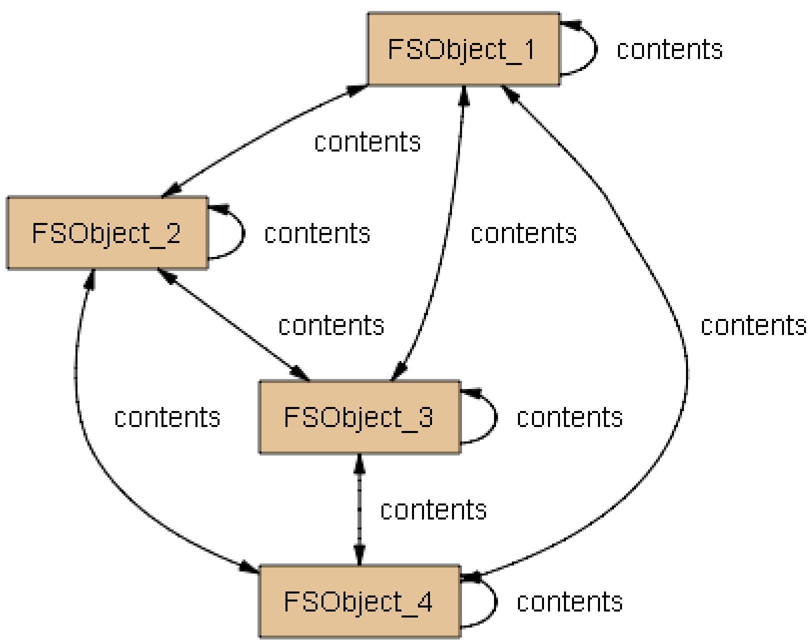
- a simple model of a file system. It has a notion of a "file system object" which can be either a file or a directory. Every file system object knows its parent, and directories also know their contents. We will also create the notion of a "root directory", which resides at the top of the file system.
- Define files, directories, and file system objects, and make sure that they have the appropriate fields (parent, contents, etc.). Write simple facts to constrain the file system and to impose simple sanity constraints. For instance: every file system object must be either a file or a directory (not both, not neither); the file system is connected, the root has no parent; and so on.
- Make assertions about properties you think (or hope) are true as a result of the sanity constraints you wrote. Check these assertions, and handle when they fail to hold. For instance, verify that a file system is acyclic, even though we do not explicitly force it to be so.
- The file system will have some shortcomings; most prominently, it will be static. Later, dynamic operations, such as move and delete, will be introduced later.

A Walkthrough: A file System I

module models/examples/tutorial/filesystem

// A set representing the set of all file system objects in the file system; A parent relation with the set as its domain.
sig FSObject { parent: lone Dir } // lone = 0 or 1

// A directory in the file system; "extends" = a disjoint partition; cf. "in" = a subset but not a disjoint partition
sig Dir extends FSObject { contents: set FSObject }



Any issues?

// A directory is the parent of its contents
fact defineContents { all d: Dir, o: d.contents | o.parent = d }

Quantifiers:
- all x:X | formula
- no x:X | formula
- one x:X | formula
- lone x:X | formula

A Walkthrough: A File System I

module models/examples/tutorial/filesystem

// A set representing the set of all file system objects in the file system; A parent relation with the set as its domain.
sig FSObject { parent: lone Dir } // lone = 0 or 1

sig Dir extends FSObject { contents: set FSObject } *// A directory in the file system*

fact defineContents { all d: Dir, o: d.contents | o.parent = d } *// A directory is the parent of its contents. "." = relation composition*

sig File extends FSObject {} *// A file in the file system: extends = 1) subset; 2) disjoint from other subsets*

// All file system objects are either files or directories; "+"= set union; Without this, a FSObject can be neither a Dir nor a File.

fact fileDirPartition { File + Dir = FSObject } // == abstract sig FSObject {...}

one sig Root extends Dir { }{ no parent } *// There exists a root; one = There will always be exactly* one instance.

// File system is connected; "in" = subset of (among other things); "." = reflexive transitive closure*
fact fileSystemConnected { FSObject in Root.*contents }

// A fact forces something to be true of the model. An assert claims that something must be true due to the behaviour of the model.
assert acyclic { no d: Dir | d in d.^contents } // *The contents path is acyclic;* ".^" = transitive closure

check acyclic for 5 // "for 5" = examine all examples whose top level signatures (those that don't extend other signatures) have up to 5 instances.

assert oneRoot { one d: Dir | no d.parent } *// File system has one root*

check oneRoot for 5

assert oneLocation { all o: FSObject | lone d: Dir | o in d.contents } *// Every fsobject is in at most one directory*

check oneLocation for 5

// "check": 1) "no solution found": no counterexamples to the assertion; 2) "solution found": a counterexample
// Ordering among signature declarations is irrelev

- **union (+):** t is in p+q if and only if t is in p *or* t is in q.
- **intersection (&):** t is in p&q if and only if t is in p *and* t is in q.
- **set subtraction (-):** t is in p-q if and only if t is in p *but* t is *not* in q.
- **set membership/subset (in):** Set membership and subsets are both denoted **in**.

The same symbol is used, since ***Alloy does not distinguish between atoms and singleton sets.***

A Walkthrough: A File System I – Checking Assertions

assert acyclic { no d: Dir | d in d.^contents } *// The contents path is acyclic;* ".^" = transitive closure

check acyclic for 5 // "for 5" = examine all examples whose top level signatures (those that don't extend other signatures) have up to 5 instances.

→ "*no counterexamples found: acyclic may be valid.(00:05)*"

assert oneRoot { one d: Dir | no d.parent } *// File system has one root*

check oneRoot for 5

→ "*no counterexamples found: oneRoot may be valid.(00:05)*"

assert oneLocation { all o: FSObject | lone d: Dir | o in d.contents } *// Every fsobject is in at most one directory*

check oneLocation for 5

→ "*no counterexamples found: oneLocation may be valid.(00:05)*"

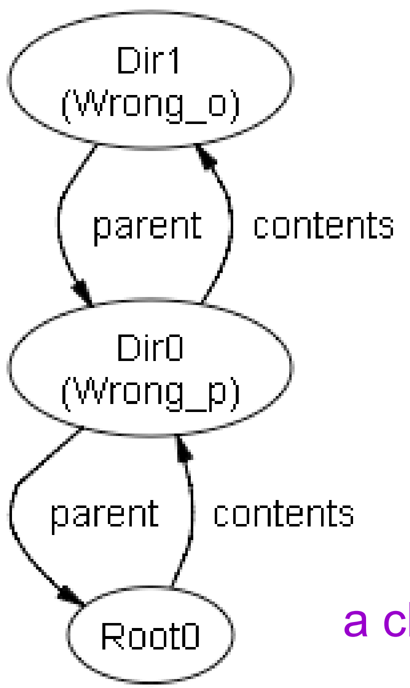
A Walkthrough: A File System I – Checking Assertions

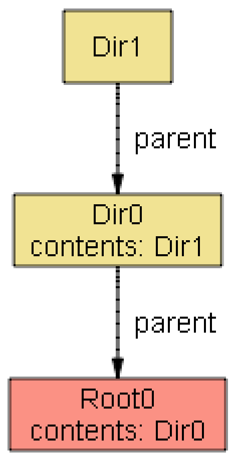
// an assertion with a counterexample

// any two non-root file system objects have the same parent.

assert Wrong { all o, p: (FSObject - Root) | (o.parent = p.parent) }

check Wrong for 3





a cleaner customization

check Wrong for 2

"no counterexamples found: Wrong may be valid.(00:02)",

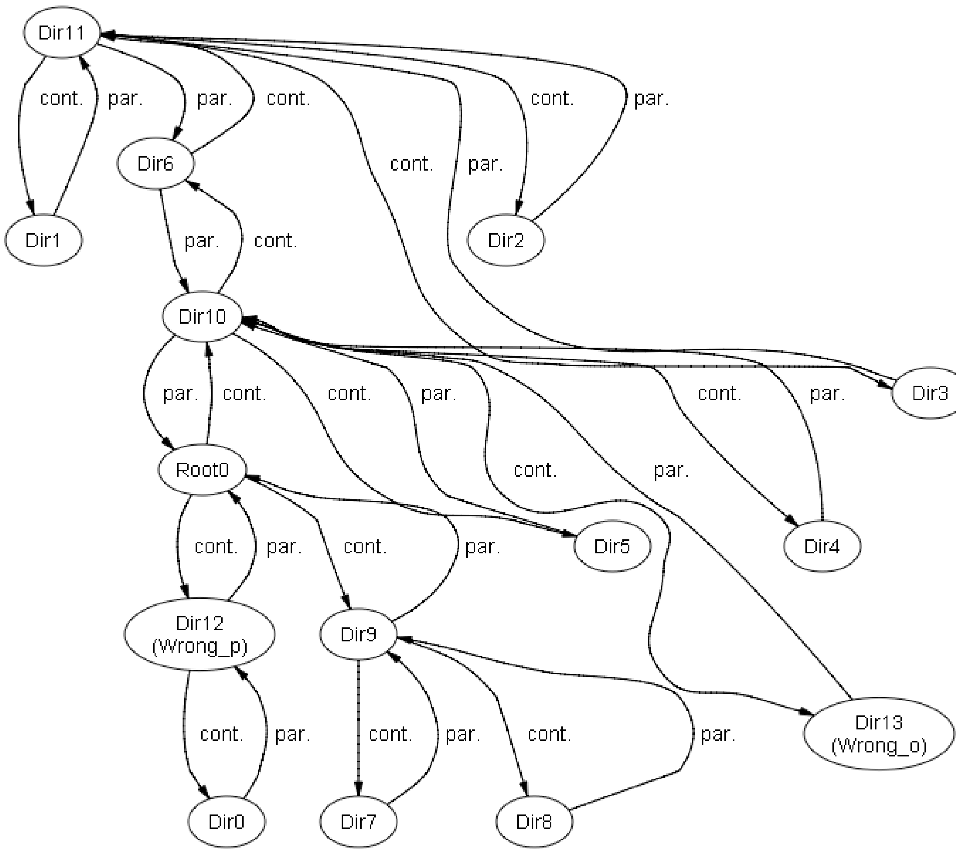
A Walkthrough: A File System I – Checking Assertions

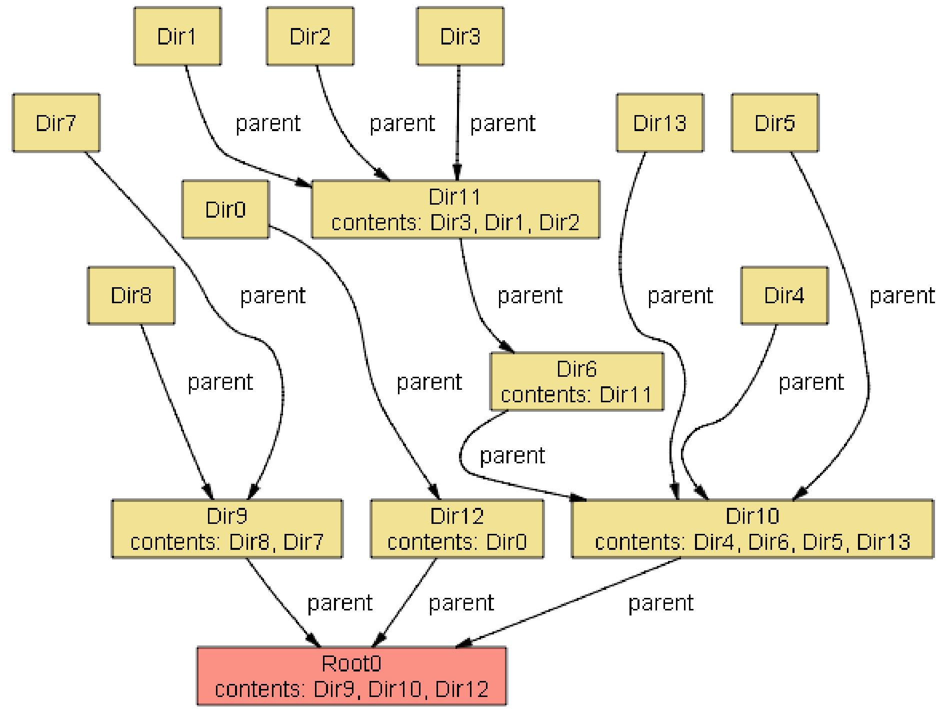
// an assertion with a counterexample

// any two non-root file system objects have the same parent.

assert Wrong { all o, p: (FSObject - Root) | (o.parent = p.parent) }

check Wrong for 15





a cleaner customization

An Alloy Model: 3 Levels

OO paradigm
Set theory
Atoms & relations

Example:

```
sig S extends E { F:T }

fact { all s:S | s.F in X }
```

OO paradigm

- **S is a class**
- **S extends its superclass E**
- **F is a field of S pointing to a T**
- **s is an instance of S**
- **. accesses a field**
- **s.F returns something of type T**

Set theory

- **S is a set (and so is E)**
- **S is a subset of E**
- **F is a relation which maps each S to some T**
- **s is an element of S**
- **. composes relations**
- **s.F composes the unary relation s with the binary relations F, returning a unary relation of type T**

Atoms & relations

- **S is an atom (and so is E)**
- **the containment relation maps E to S (among other things it does)**
- **F is a relation from S to T**
- **the containment relation maps S to s (among other things)**
- **. composes relations**
- **s.F composes the unary relation s with the binary relations F, resulting in a unary relation t, such that the containment relation maps T to t**

An Alloy Model: Logical Operations

!F // negations: not F
F && G // conjunction: F and G
F || G // disjunction: F or G

F => G // implication: same as !F || G
G <=> G // bi-implication: same as (F => G) && (G => F)
F => G,H // conditionsl: if F then G else H; same as (F => G) && (!F=> H)

Operator precedence:
not (!)
and (&&)
or (||)
implication/conditional (=>)
bi-implication (<=>)

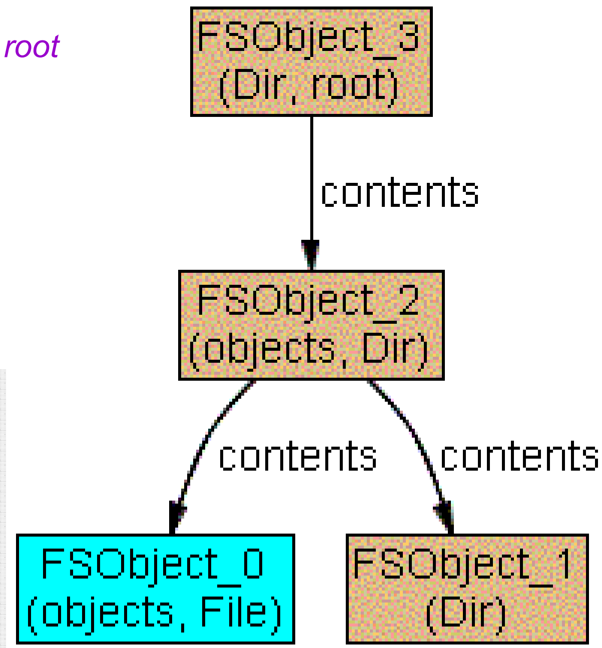
A Walkthrough: A File System II

```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // A file system object in the file system

// File system objects must be either directories or files.
sig File, Dir extends FSObject {}

sig FileSystem { // A File System
        root: Dir,
        objects: set FSObject,
        contents: Dir lone-> FSObject, // ternary relations; relational product operator ("->").
        parent: FSObject ->lone Dir }   // multiplicity markings: lone

        {no root.parent // root has no parent

        objects in root.*contents // objects are those reachable from the root

        parent = ~contents // parent is the inverse of contents }

pred example() {} // an empty predicate

// A run command, not a check command, generates solutions that are not
// counterexamples -- no claim has been made so there's nothing to disprove!
run example for exactly 1 FileSystem, 4 FSObject
// use "run" to detect overconstrained (over-specialized) or underconstrained (over-generalized) models
```

sig name { //fields of the signature }
{ //appended fact constraints }

A Walkthrough: A File System II - Problematic

```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // A file system object in the file system

// File system objects must be either directories or files.
sig File, Dir extends FSObject {}

// A File System
sig FileSystem {
        root: Dir,
        objects: set FSObject,
        contents: Dir lone-> FSObject, // ternary relations; relational product operator ("->").
        parent: FSObject ->lone Dir }   // multiplicity markings: lone

        {no root.parent // root has no parent

        objects in root.*contents // objects are those reachable from the root

        parent = ~contents // parent is the inverse of contents }

pred example() {} // an empty predicate

run example for exactly 1 FileSystem, 4 FSObject
```

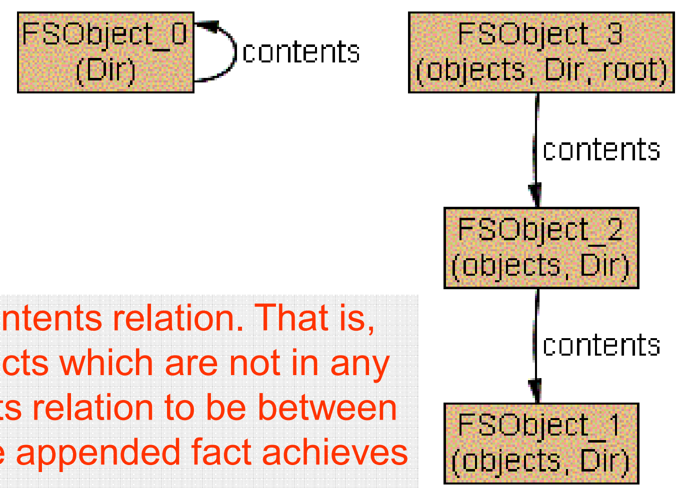


The problem with this instance is that the bottom-right Dir is reachable from the root which is not considered to be a object in the file system (it is not labeled 'objects'). That directory is reachable from the file system, but is not part of it!

objects in root.*contents ensures that all objects in the file system are reachable from the root, but fails to guarantee that all objects reachable from the root are in the file system.

A Walkthrough: A File System II Modified – But Still Problematic

```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // A file system object in the file system

// File system objects must be either directories or files.
sig File, Dir extends FSObject {}

// A File System
sig FileSystem {
        root: Dir,
        objects: set FSObject,
        contents: Dir lone-> FSObject, // ternary relations; relational product operator ("->").
        parent: FSObject ->lone Dir }   // multiplicity markings: lone

        {no root.parent // root has no parent

        // objects are those reachable from the root AND
        // all objects reachable from the root are in the file system.
        objects = root.*contents

        parent = ~contents // parent is the inverse of contents }

pred example() {} // an empty predicate

run example for exactly 1 FileSystem, 4 FSObject
```



Our previous problem is gone, but now there are stray tuples in the contents relation. That is, the contents relation is defining relationships between file system objects which are not in any file system. We can fix this by constraining all the tuples in the contents relation to be between objects that are in its file system. Adding the following constraint to the appended fact achieves this

contents in objects->objects

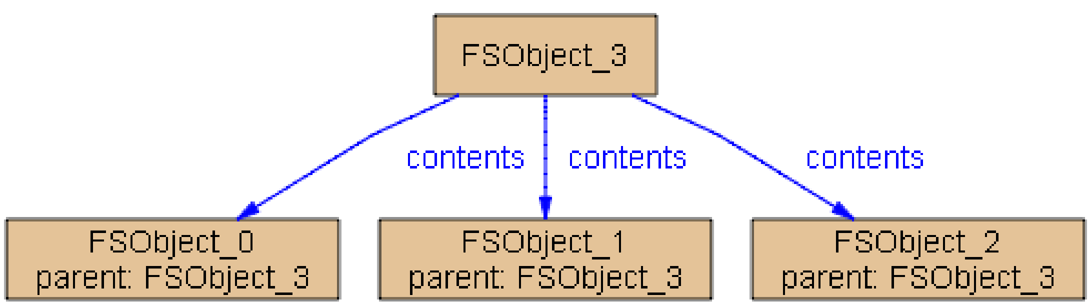
A Walkthrough: A File System II → III

```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // A file system object in the file system

// File system objects must be either directories or files.
sig File, Dir extends FSObject {}

sig FileSystem {// A File System
        root: Dir,
        objects: set FSObject,
        contents: Dir lone-> FSObject,
        parent: FSObject ->lone Dir }
        { no root.parent // root has no parent
        objects = root.*contents // objects are those reachable from the root
        contents in objects->objects // contents only defined on objects
        parent = ~contents // parent is the inverse of contents }

        pred example() {}

        run example for exactly 1 FileSystem, exactly 4 FSObject
```

A Walkthrough: A File System II → III

Enforcing some degree of non-triviality

```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // A file system object in the file system

sig File, Dir extends FSObject {} // File system objects must be either directories or files.

sig FileSystem {// A File System
        root: Dir,
        objects: set FSObject,
        contents: Dir lone-> FSObject,
        parent: FSObject ->lone Dir }
        { no root.parent // root has no parent
        objects = root.*contents // objects are those reachable from the root
        contents in objects->objects // contents only defined on objects
        parent = ~contents // parent is the inverse of contents }

        pred example() {}

        // run example for exactly 1 FileSystem, exactly 4 FSObject
         run example for all f: FSObject | some fs: FileSystem | f in fs.objects
```



A Walkthrough: A File System IV

semantically equivalent to File System III, but a bit more concise and arguably clearer.

```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // A file system object in the file system

// File system objects must be either directories or files.
sig File, Dir extends FSObject {}

// A File System
sig FileSystem {
        objects: set FSObject,
        root: Dir & objects,
        contents: (Dir & objects) one-> (objects - root), // "one" = exactly 1
        parent: (objects - root) ->one (Dir & objects) }
        {objects in root.*contents // objects are reachable from the root
          parent = ~contents // parent is the inverse of contents }

pred example() {}

run example for exactly 1 FileSystem, exactly 4 FSObject
```

A Walkthrough: A File System V - 1

```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // File system objects

sig File, Dir extends FSObject {}

sig FileSystem {// A File System
        root: Dir,
        objects: set FSObject,
        contents: Dir lone-> FSObject,
        parent: FSObject ->lone Dir }
        {no root.parent // root has no parent
         objects = root.*contents // objects are those reachable from the root
         contents in objects->objects // contents only defined on objects
         parent = ~contents // parent is the inverse of contents }
```

A Walkthrough: A File System V - 2

```
// Move FSObject f to Directory d
pred mv (fs, fs': FileSystem, f: FSObject, d: Dir) {
        (f + d) in fs.objects fs'.contents = fs.contents - f.(fs.parent)->f + d->f }

// Delete the file f
pred rm (fs, fs': FileSystem, f: File) {
        f in fs.objects fs'.contents = fs.contents - f.(fs.parent)->f }

// Delete the directory d
pred rmdir(fs, fs': FileSystem, d: Dir) {
        d in fs.(objects - root)
        no d.(fs.contents) //d is empty
        fs'.contents = fs.contents - d.(fs.parent)->d }

// Recursively delete the file system object f
pred rm_r(fs, fs': FileSystem, f: FSObject) {
        f in fs.(objects - root)
        let subtree = f.*(fs.contents) |
                fs'.contents = fs.contents - subtree.(fs.parent)->subtree }

run mv for 2 FileSystem, 4 FSObject
run rm for 2 FileSystem, 4 FSObject
run rmdir for 2 FileSystem, 4 FSObject
run rm_r for 2 FileSystem, 4 FSObject
```

A Walkthrough: A File System V - 3

```
// Moving doesn't add or delete any file system objects
assert moveAddsRemovesNone {
        all fs, fs': FileSystem, f: FSObject, d:Dir | mv(fs, fs', f, d) => fs.objects = fs'.objects }

// rm removes exactly the specified file
assert rmRemovesOneFile {
        all fs, fs': FileSystem, f: File | rm(fs, fs', f) => fs.objects - f = fs'.objects }

// rmdir removes exactly the specified directory
assert rmdirRemovesOneDir {
        all fs, fs': FileSystem, d: Dir | rmdir(fs, fs', d) => fs.objects - d = fs'.objects }

// rm_r removes exactly the specified subtree
assert rm_rRemovesSubtree {
        all fs, fs': FileSystem, f: FSObject | rm_r(fs, fs', f) => fs.objects - f.*(fs.contents) = fs'.objects }

// rm and rm_r same effect on files
assert rmAndrm_rSameForFiles {
        all fs, fs1, fs2: FileSystem, f: File | rm(fs, fs1, f) && rm_r(fs, fs2, f) => fs1.contents = fs2.contents
}

check moveAddsRemovesNone for 5 //passes
check rmRemovesOneFile for 5 //passes
check rmdirRemovesOneDir for 5 //counterexample!
check rm_rRemovesSubtree for 5 //counterexample!
check rmAndrm_rSameForFiles for 5 //passes

pred example() {}

run example for exactly 1 FileSystem, exactly 4 FSObject
```

A Walkthrough: A File System V - All

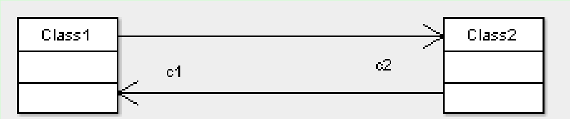
```
module models/examples/tutorial/filesystem

abstract sig FSObject {} // File system objects

sig File, Dir extends FSObject {}

sig FileSystem {// A File System
        root: Dir,
        objects: set FSObject,
        contents: Dir lone-> FSObject,
        parent: FSObject ->lone Dir }
        {no root.parent // root has no parent
         objects = root.*contents // objects are those reachable from the root
         contents in objects->objects // contents only defined on objects
         parent = ~contents // parent is the inverse of contents }

// Move FSObject f to Directory d
pred mv (fs, fs': FileSystem, f: FSObject, d: Dir) {
        (f + d) in fs.objects fs'.contents = fs.contents - f.(fs.parent)->f + d->f }

// Delete the file f
pred rm (fs, fs': FileSystem, f: File) {
        f in fs.objects fs'.contents = fs.contents - f.(fs.parent)->f }

// Delete the directory d
pred rmdir(fs, fs': FileSystem, d: Dir) {
        d in fs.(objects - root)
        no d.(fs.contents) //d is empty
        fs'.contents = fs.contents - d.(fs.parent)->d }

// Recursively delete the file system object f
pred rm_r(fs, fs': FileSystem, f: FSObject) {
        f in fs.(objects - root)
        let subtree = f.*(fs.contents) | fs'.contents = fs.contents - subtree.(fs.parent)->subtree }

run mv for 2 FileSystem, 4 FSObject
run rm for 2 FileSystem, 4 FSObject
run rmdir for 2 FileSystem, 4 FSObject
run rm_r for 2 FileSystem, 4 FSObject

// Moving doesn't add or delete any file system objects
assert moveAddsRemovesNone {
        all fs, fs': FileSystem, f: FSObject, d:Dir | mv(fs, fs', f, d) => fs.objects = fs'.objects }

// rm removes exactly the specified file
assert rmRemovesOneFile {
        all fs, fs': FileSystem, f: File | rm(fs, fs', f) => fs.objects - f = fs'.objects }

// rmdir removes exactly the specified directory
assert rmdirRemovesOneDir {
        all fs, fs': FileSystem, d: Dir | rmdir(fs, fs', d) => fs.objects - d = fs'.objects }

// rm_r removes exactly the specified subtree
assert rm_rRemovesSubtree {
        all fs, fs': FileSystem, f: FSObject | rm_r(fs, fs', f) => fs.objects - f.*(fs.contents) = fs'.objects }

// rm and rm_r same effect on files
assert rmAndrm_rSameForFiles {
        all fs, fs1, fs2: FileSystem, f: File | rm(fs, fs1, f) && rm_r(fs, fs2, f) => fs1.contents = fs2.contents }

check moveAddsRemovesNone for 5 //passes
check rmRemovesOneFile for 5 //passes
check rmdirRemovesOneDir for 5 //counterexample!
check rm_rRemovesSubtree for 5 //counterexample!
check rmAndrm_rSameForFiles for 5 //passes

pred example() {}

run example for exactly 1 FileSystem, exactly 4 FSObject
```

A Walkthrough: A File System VI

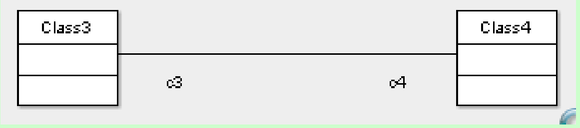
http://alloy.mit.edu/tutorial3/currentmodel-FS-VI.html

UML And Alloy

Prepared by Weimin Ma



Class Diagram (a)

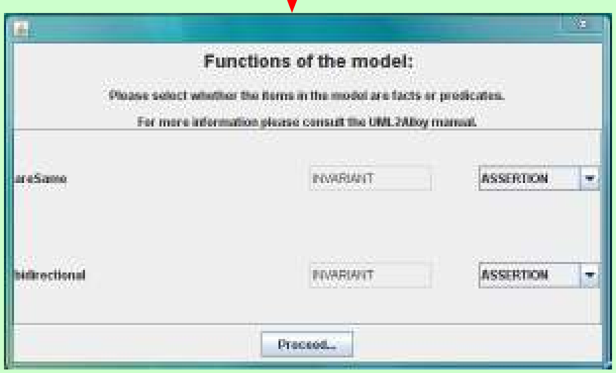


Class Diagram (b)

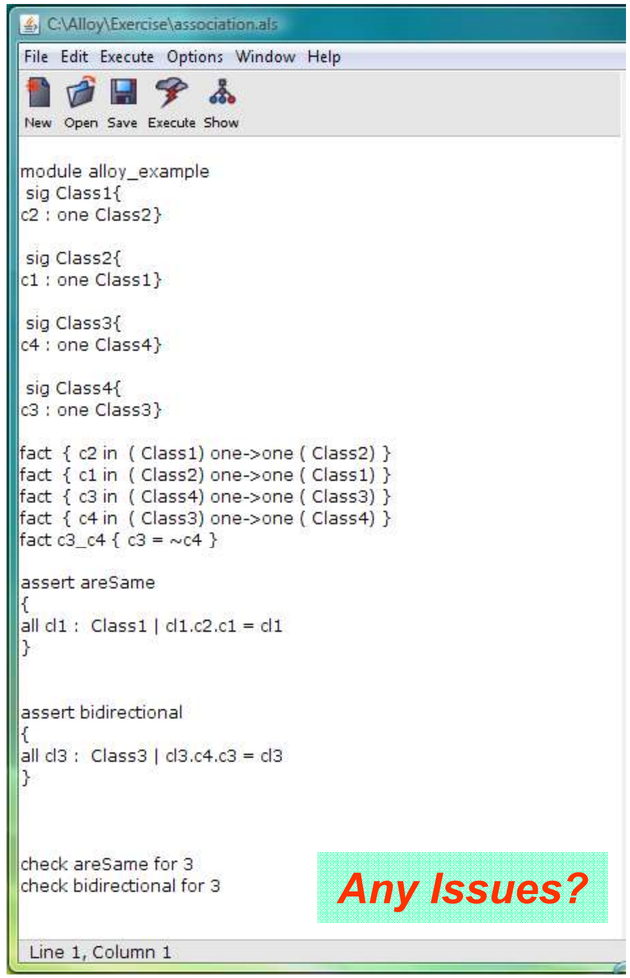
context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains

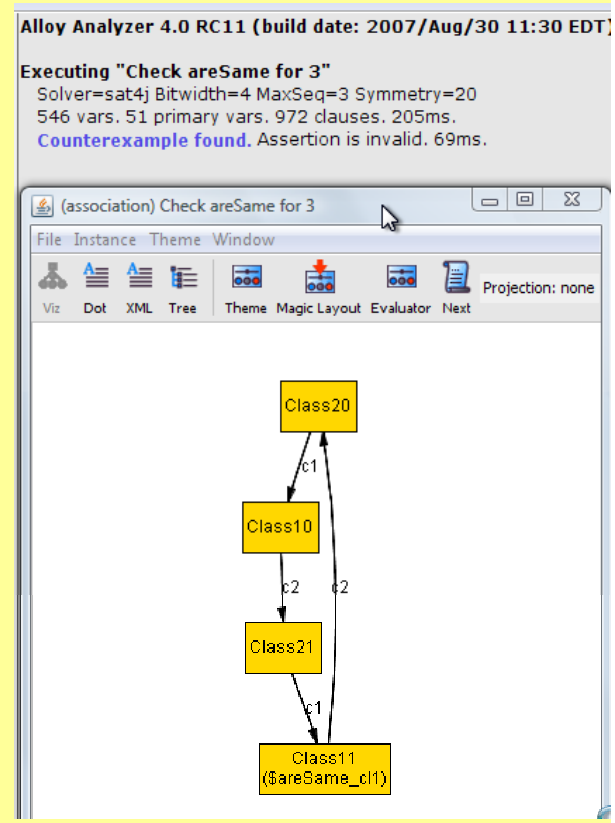


Translating UML to Alloy using UML2Alloy

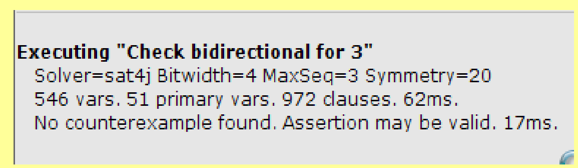


Any Issues?

Translated Alloy Model



Class Diagram (a) Execution Result



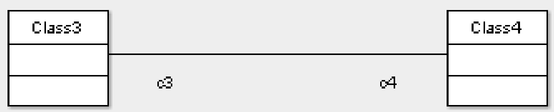
Class Diagram (b) Execution Result

Alloy Execution Result

UML And Alloy

OCL now part of model – class diagram (a)

Class Diagram (a)

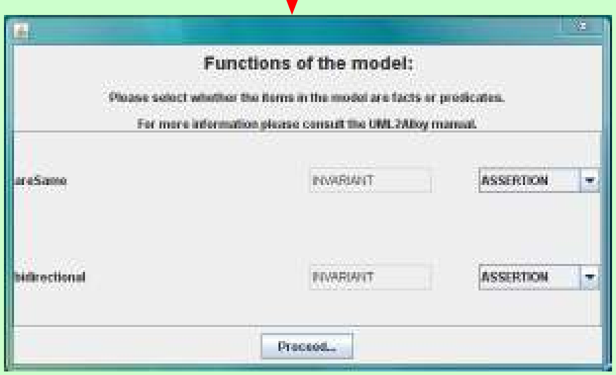


Class Diagram (b)

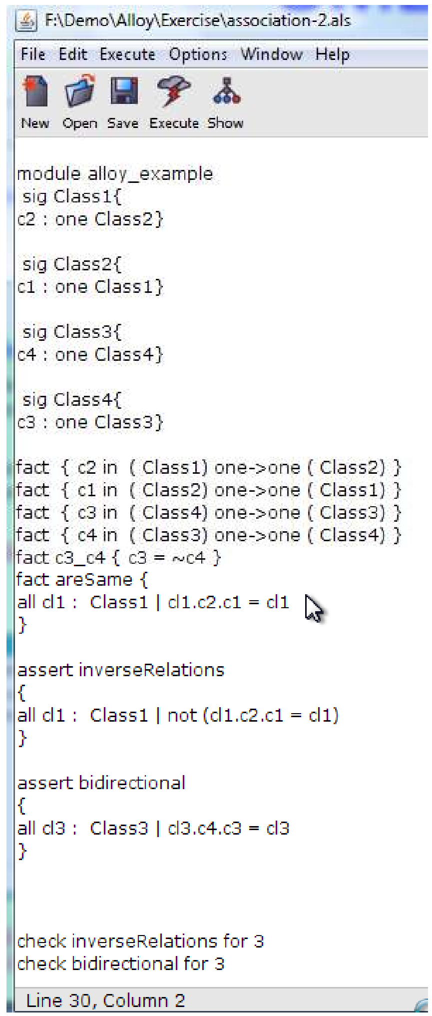
context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains



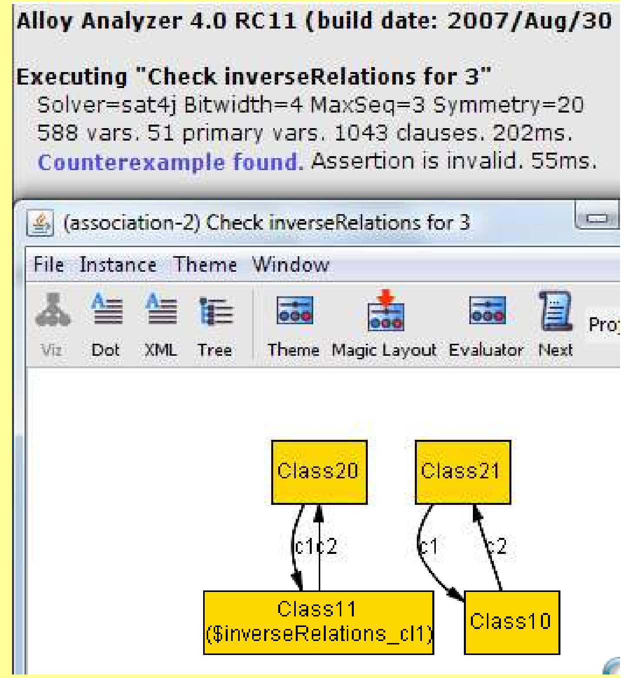
Translating UML to Alloy using UML2Alloy




```
module alloy_example
 sig Class1{
c2 : one Class2}

 sig Class2{
c1 : one Class1}

 sig Class3{
c4 : one Class4}

 sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }
fact areSame {
all cl1 : Class1 | cl1.c2.c1 = cl1
}

assert inverseRelations
{
all cl1 : Class1 | not (cl1.c2.c1 = cl1)
}

assert bidirectional
{
all cl3 : Class3 | cl3.c4.c3 = cl3
}


check inverseRelations for 3
check bidirectional for 3
```

Modified Alloy Model

"areSame" is part of UML model



Class Diagram (a) Execution Result

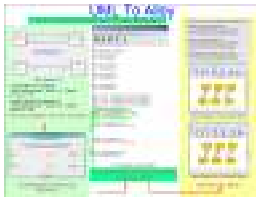
"areSame" is part of UML model

Executing "Check bidirectional for 3"
Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
588 vars. 51 primary vars. 1044 clauses. 41ms.
No counterexample found. Assertion may be valid. 8ms.

Class Diagram (b) Execution Result

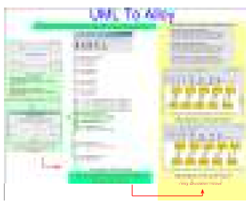
Alloy Execution Result

More on Run and Check



Run predicates

OCL constrains translated into ***predicates*** without being part of UML class

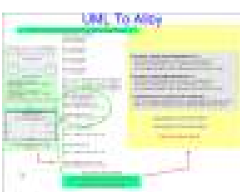


Run predicates

OCL constrains translated into ***facts (&=predicates)*** being part of UML class

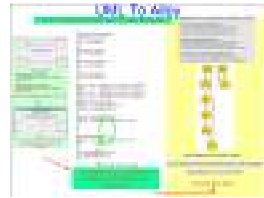
Check assertions

OCL constrains translated into ***assertions*** without being part of UML class



Check assertions

OCL constrains translated into ***facts (&=predicates)*** being part of UML class ***and assertions***

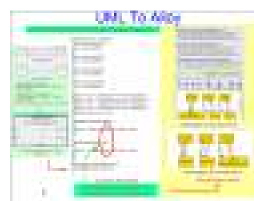


Run ~predicates

OCL constrains translated into ***negative predicates*** without being part of UML class

Run ~predicates

OCL constrains translated into ***negative predicates*** being part of UML class



Check ~assertions

OCL constrains translated into ***negative assertions*** without being part of UML class

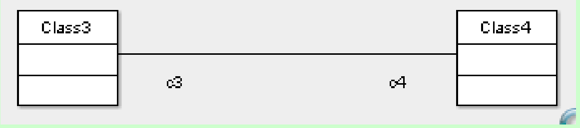
Check ~assertions

OCL constrains translated into ***facts (&=predicates)*** being part of UML class ***and negative assertions***

UML To Alloy

Predicate constrains are NOT part of class diagrams

Class Diagram (a)

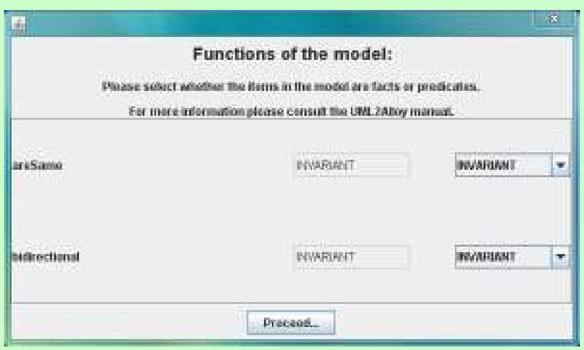


Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains



Translating UML to Alloy using UML2Alloy

I:\Demo\Alloy\Exercise\MyModel\pred_no_constrain.als

File Edit Execute Options Window Help

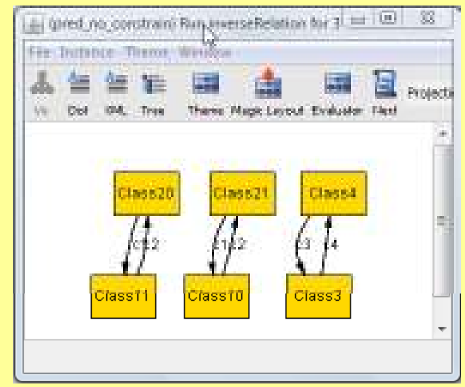
New Open Save Execute Show

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }
//fact { inverseRelation[ ] }
//fact { bidirectional[ ] }

pred inverseRelation ( ){
all cl1 : Class1 | cl1.c2.c1 = cl1
}

pred bidirectional ( ){
all cl3 : Class3 | cl3.c4.c3 = cl3
}

run inverseRelation for 3
run bidirectional for 3
```

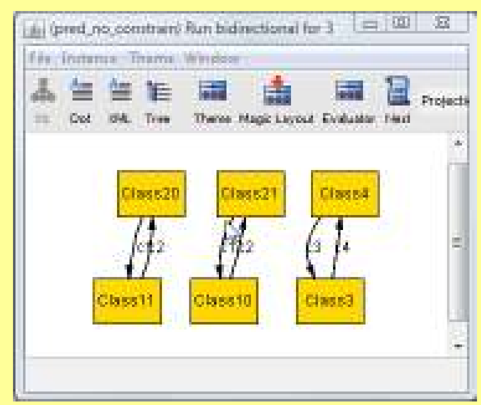
Translated Alloy Model

Constrains are NOT part of Alloy model as predicates

```
Alloy Analyzer 4.0 RC11 (build date: 2007/Aug/30 11:30

Executing "Run inverseRelation for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  543 vars. 48 primary vars. 933 clauses. 357ms.
  Instance found. Predicate is consistent. 106ms.

Executing "Run bidirectional for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  543 vars. 48 primary vars. 933 clauses. 85ms.
  Instance found. Predicate is consistent. 48ms.

2 commands were executed. The results are:
  #1: Instance found. inverseRelation is consistent.
  #2: Instance found. bidirectional is consistent.
```



Class Diagram (a) Execution Result

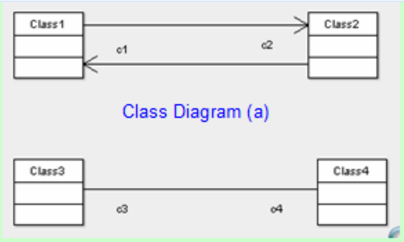


Class Diagram (b) Execution Result

Alloy Execution Result

UML To Alloy

Predicate constrains are part of class diagrams



Class Diagram (a)

Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains

Translating UML to Alloy using UML2Alloy

I:\Demo\Alloy\Exercise\MyModel\pred_constrain...

File Edit Execute Options Window Help

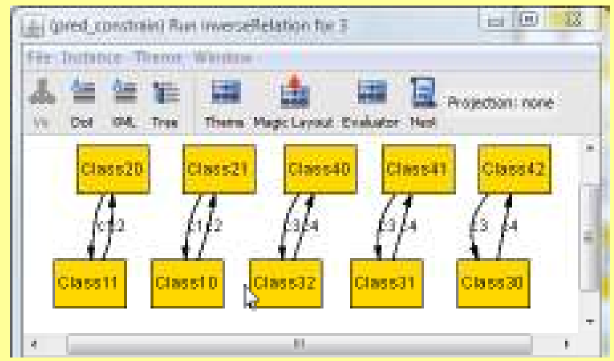
New Open Save Execute Show

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }
fact { inverseRelation[ ] }
fact { bidirectional[ ] }

pred inverseRelation ( ){
all cl1 : Class1 | cl1.c2.c1 = cl1
}

pred bidirectional ( ){
all cl3 : Class3 | cl3.c4.c3 = cl3
}

run inverseRelation for 3
run bidirectional for 3
```

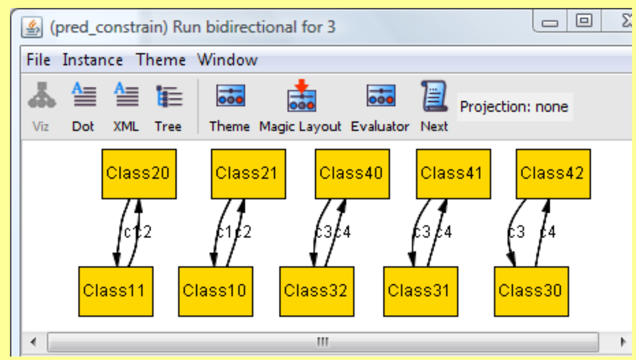
Translated Alloy Model

Constrains are part of Alloy model as predicates

```
Alloy Analyzer 4.0 RC11 (build date: 2007/Aug/30 11:30

Executing "Run inverseRelation for 3"
   Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
   585 vars. 48 primary vars. 1005 clauses. 456ms.
   Instance found. Predicate is consistent. 140ms.

Executing "Run bidirectional for 3"
   Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
   585 vars. 48 primary vars. 1005 clauses. 179ms.
   Instance found. Predicate is consistent. 36ms.

2 commands were executed. The results are:
   #1: Instance found. inverseRelation is consistent.
   #2: Instance found. bidirectional is consistent.
```



Class Diagram (a) Execution Result

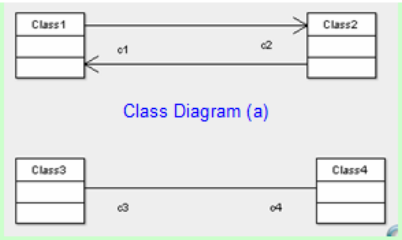


Class Diagram (b) Execution Result

Alloy Execution Result

UML To Alloy

Assertion constrains are NOT part of class diagrams



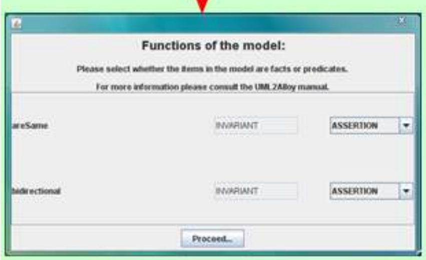
Class Diagram (a)

Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains



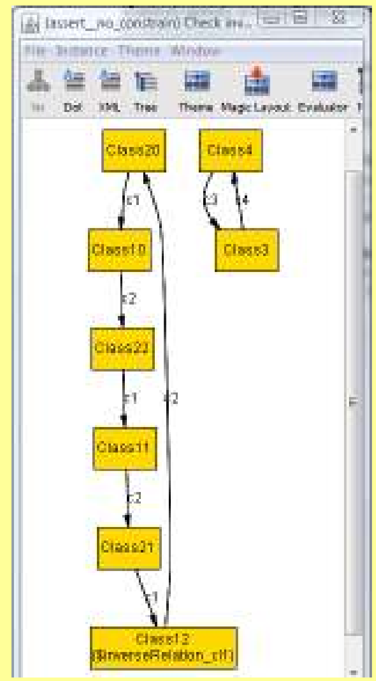
Translating UML to Alloy using UML2Alloy

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }

assert inverseRelation
{
all cl1 : Class1 | cl1.c2.c1 = cl1
}

assert bidirectional
{
all cl3 : Class3 | cl3.c4.c3 = cl3
}

check inverseRelation for 3
check bidirectional for 3
```

Translated Alloy Model

Constrains are NOT part of Alloy model as assertions

```
Executing "Check inverseRelation for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  550 vars. 51 primary vars. 980 clauses. 392ms.
  Counterexample found. Assertion is invalid. 125ms.

Executing "Check bidirectional for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  550 vars. 51 primary vars. 980 clauses. 113ms.
  No counterexample found. Assertion may be valid. 27ms.

2 commands were executed. The results are:
  #1: Counterexample found. inverseRelation is invalid.
  #2: No counterexample found. bidirectional may be valid.
```



Class Diagram (a) Execution Result

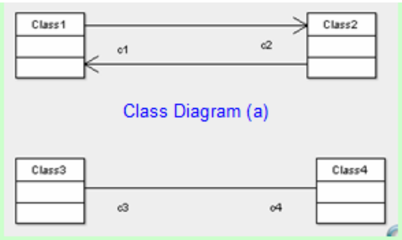
#2: No counterexample found. bidirectional may be valid.

Class Diagram (b) Execution Result

Alloy Execution Result

UML To Alloy

Assertion constrains are part of class diagrams



Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains

Translating UML to Alloy using UML2Alloy

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact  { c2 in  ( Class1) one->one ( Class2) }
fact  { c1 in  ( Class2) one->one ( Class1) }
fact  { c3 in  ( Class4) one->one ( Class3) }
fact  { c4 in  ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }
fact  { inverseRelation[  ] }
fact  { bidirectional[  ] }
pred inverseRelation (  ){
all cl1 :  Class1 | cl1.c2.c1 = cl1
}
pred bidirectional (  ){
all cl3 :  Class3 | cl3.c4.c3 = cl3
}

assert inverseRelation
{
all cl1 :  Class1 | cl1.c2.c1 = cl1
}

assert bidirectional
{
all cl3 :  Class3 | cl3.c4.c3 = cl3
}

check areSame for 3 but 0 int
check bidirectional for 3 but 0 int
```

Translated Alloy Model

Constrains are part of Alloy model as assertions

Executing "Check inverseRelation2 for 3"
Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
634 vars. 51 primary vars. 1124 clauses. 394ms.
No counterexample found. Assertion may be valid. 76ms.

Executing "Check bidirectional2 for 3"
Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
634 vars. 51 primary vars. 1124 clauses. 111ms.
No counterexample found. Assertion may be valid. 25ms.

2 commands were executed. The results are:
#1: No counterexample found. inverseRelation2 may be valid.
#2: No counterexample found. bidirectional2 may be valid.

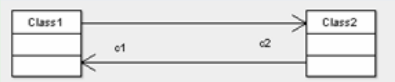
Class Diagram (a) Execution Result

Class Diagram (b) Execution Result

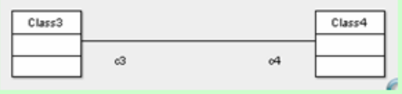
Alloy Execution Result

UML To Alloy

Predicate constrains are NOT part of class diagrams



Class Diagram (a)



Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains

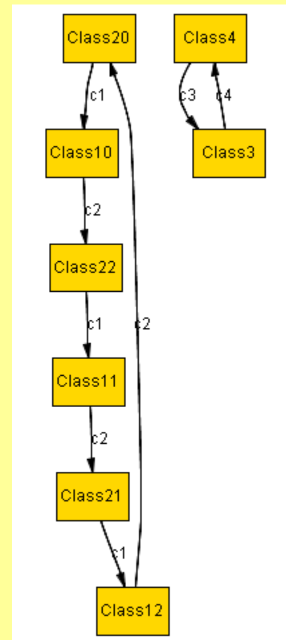
Translating UML to Alloy using UML2Alloy

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }
//fact { inverseRelation[ ] }
//fact { bidirectional[ ] }

pred inverseRelation ( ){
all cl1 : Class1 | not (cl1.c2.c1 = cl1)
}


pred bidirectional ( ){
all cl3 : Class3 | not (cl3.c4.c3 = cl3)
}

run inverseRelation for 3
run bidirectional for 3
```

Translated Alloy Model

Predicates to be checked are negation of predicate constrains

```
Alloy Analyzer 4.0 RC11 (build date: 2007/Aug/30 11:30 EDT

Executing "Run inverseRelation for 3"
   Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
   543 vars. 48 primary vars. 930 clauses. 383ms.
   Instance found. Predicate is consistent. 119ms.

Executing "Run bidirectional for 3"
   Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
   543 vars. 48 primary vars. 930 clauses. 131ms.
   No instance found. Predicate may be inconsistent. 18ms.

2 commands were executed. The results are:
   #1: Instance found. inverseRelation is consistent.
   #2: No instance found. bidirectional may be inconsistent.
```



Class Diagram (a) Execution Result

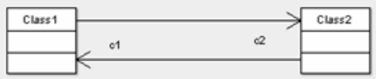
#2: No instance found. bidirectional may be inconsistent.

Class Diagram (b) Execution Result

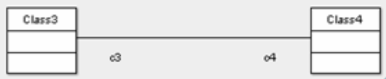
Alloy Execution Result

UML To Alloy

Predicate constrains are part of class diagrams



Class Diagram (a)



Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains

Translating UML to Alloy using UML2Alloy

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }
fact { inverseRelation[ ] }
fact { bidirectional[ ] }

pred inverseRelation ( ){
all cl1 : Class1 | cl1.c2.c1 = cl1
}

pred bidirectional ( ){
all cl3 : Class3 | cl3.c4.c3 = cl3
}

pred inverseRelation2 ( ){
all cl1 : Class1 | not (cl1.c2.c1 = cl1)
}

pred bidirectional2 ( ){
all cl3 : Class3 | not (cl3.c4.c3 = cl3)
}

run inverseRelation2 for 3
run bidirectional2 for 3
```

Translated Alloy Model

Predicate to check are negation of predicate constrains

Executing "Run inverseRelation2 for 3"
Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
588 vars. 48 primary vars. 1074 clauses. 95ms.
No instance found. Predicate may be inconsistent. 9ms.

Executing "Run bidirectional2 for 3"
Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
588 vars. 48 primary vars. 1074 clauses. 71ms.
No instance found. Predicate may be inconsistent. 28ms.

2 commands were executed. The results are:
#1: No instance found. inverseRelation2 may be inconsistent.
#2: No instance found. bidirectional2 may be inconsistent.

Class Diagram (a) Execution Result

Class Diagram (b) Execution Result

Alloy Execution Result

UML To Alloy

Assertion constrains are NOT part of class diagrams

Class Diagram (a)

Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains

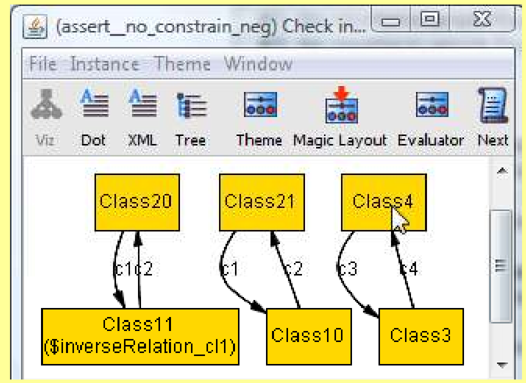
Translating UML to Alloy using UML2Alloy

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }

assert inverseRelation
{
all cl1 : Class1 | not (cl1.c2.c1 = cl1)
}

assert bidirectional
{
all cl3 : Class3 | not (cl3.c4.c3 = cl3)
}

check inverseRelation for 3
check bidirectional for 3
```

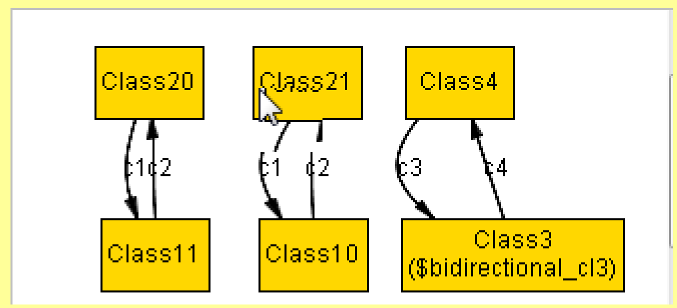
Translated Alloy Model

Assertions are negation of intended constrains

```
Executing "Check inverseRelation for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  550 vars. 51 primary vars. 979 clauses. 81ms.
  Counterexample found. Assertion is invalid. 38ms.

Executing "Check bidirectional for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  550 vars. 51 primary vars. 979 clauses. 67ms.
  Counterexample found. Assertion is invalid. 21ms.

2 commands were executed. The results are:
  #1: Counterexample found. inverseRelation is invalid.
  #2: Counterexample found. bidirectional is invalid.
```



Class Diagram (a) Execution Result

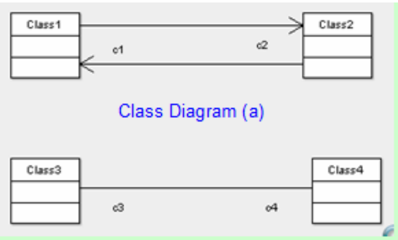


Class Diagram (b) Execution Result

Alloy Execution Result

UML To Alloy

Assertion constrains are part of class diagrams



Class Diagram (a)

Class Diagram (b)

context Class1 inv areSame :
Class1.allInstances -> forAll (cl1 : Class1 |
cl1.c2.c1 = cl1)

context Class3 inv bidirectional :
Class3.allInstances -> forAll (cl3 : Class3 |
cl3.c4.c3 = cl3)

UML Class Diagram with OCL Constrains

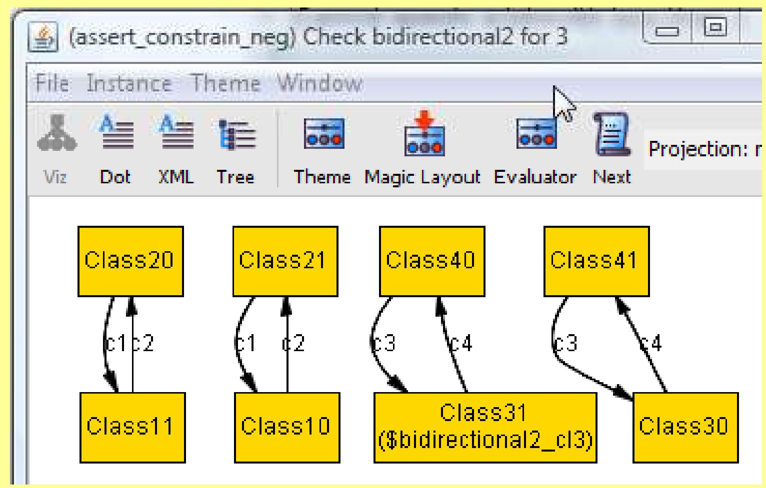
Translating UML to Alloy using UML2Alloy

```
module alloy_example
some sig Class1{
c2 : one Class2}

some sig Class2{
c1 : one Class1}

some sig Class3{
c4 : one Class4}

some sig Class4{
c3 : one Class3}

fact { c2 in ( Class1) one->one ( Class2) }
fact { c1 in ( Class2) one->one ( Class1) }
fact { c3 in ( Class4) one->one ( Class3) }
fact { c4 in ( Class3) one->one ( Class4) }
fact c3_c4 { c3 = ~c4 }
fact { inverseRelation[ ] }
fact { bidirectional[ ] }
pred inverseRelation ( ){
all cl1 : Class1 | cl1.c2.c1 = cl1
}
pred bidirectional ( ){
all cl3 : Class3 | cl3.c4.c3 = cl3
}

assert inverseRelation2
{
all cl1 : Class1 | not (cl1.c2.c1 = cl1)
}

assert bidirectional2
{
all cl3 : Class3 | not (cl3.c4.c3 = cl3)
}

check inverseRelation2 for 3
check bidirectional2 for 3
```

Translated Alloy Model

Assertions are negation of constrains

```
Executing "Check inverseRelation2 for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  634 vars. 51 primary vars. 1123 clauses. 83ms.
  Counterexample found. Assertion is invalid. 86ms.

Executing "Check bidirectional2 for 3"
  Solver=sat4j Bitwidth=4 MaxSeq=3 Symmetry=20
  634 vars. 51 primary vars. 1123 clauses. 78ms.
  Counterexample found. Assertion is invalid. 47ms.

2 commands were executed. The results are:
  #1: Counterexample found. inverseRelation2 is invalid.
  #2: Counterexample found. bidirectional2 is invalid.
```



Class Diagram (a) Execution Result

Class Diagram (b) Execution Result

Alloy Execution Result