THESIS TITLE
SECOND LINE IF NECESSARY

by

AUTHOR NAME

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
March 2010

Abstract

This is my abstract.

Acknowledgments

Blah blah blah.

Statement of Originality

Only required by CHEM, COMPUTING, GEOL, MATH and Physics (Ph.D. ONLY!).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Section

1.1.1 SubSection

SubSubSection

Paragraph

SubParagraph

1.2 Motivation

...the current de facto standard being the Unified Modeling Language (UML) [1]...

1.3 Problem

1.4 Objective

Note: These are the section headings that I decided to use. Check out several recent theses to decide how you want to lay out your introduction (and conclusion) chapters.

1.4.1 Hypothesis

1.5 Contributions

1.6 Organization of Thesis

We proceed by introducing conformance checking and discussing related work in the next chapter. We discuss the Alloy language and the Alloy Analyzer tool in Chapter 3. Chapter 4 describes our Embee tool, from the user's perspective, with several running examples. Implementation details and the analysis of the tool are presented in Chapter 5. Chapter 6 concludes and outlines future work.

Chapter 2

Background

2.1 UML

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering.

2.2 Conformance Checking

2.2.1 Multiple Definitions

- checking “whether an implementation conforms to some given design” [3]
- ensuring “that the actual software (the detailed design and code) conforms to the architecture” [2]
- etc. etc.

For our research, we are adopting the following definitions of conformance checking:

Conformance checking *is the process of comparing...*

We further refine our definition with the following caveats:

1. Our version of conformance checking .
2. We distinguish between *checking* and *ensuring*...

Don't forget to discuss related work!!!

Chapter 3

Alloy

3.1 The Alloy Language

Alloy is...

Quantifiers There are five quantifiers available in Alloy:

Quantifier	Meaning
`all x : e | F`	universal, `F` is true for every `x` in `e`
`some x : e | F`	existential, `F` is true for some `x` in `e`
`no x : e | F`	`F` is true for no `x` in `e`
`sole x : e | F`	`F` is true for at most one `x` in `e`
`one x : e | F`	`F` is true for exactly one `x` in `e`

Signatures and Fields

The simple signature `sig A {}` introduces `A` as a basic type with a set of atoms of that type. `A` refers to the set of atoms; the type is inferred by Alloy and cannot be referenced explicitly.

```
sig A {}
sig B {
    f : A
}
```

3.1.1 Example

An excerpt from an Alloy specification of a singly-linked list is presented in Listing 3.1.

Listing 3.1 Excerpt of a simple Alloy specification for a singly-linked list

```
sig Node {                    sig List {
    next : option Node            first : Node
}                             }{
                                  all n : Node | n in first.*next
                                  no n : Node | n in n.^next
                              }
```

Chapter 4

Embee: User Perspective

Listing 4.1 Alloy specification of a singly-linked list using only binary relations

---

```
module List

sig Node {
    next : option Node
}

sig List {
    first : Node
}

fact NodeInOneList {
    all n : Node | one l : List | n in (l.first).*next
}

fact NoCycle {
    all n : Node | n ! in n.^next
}

fun Show() {}

run Show for 4
```

4.0.2 Phase 1: High-Level Static Mapping

...Phase 1 simply generates the default static mapping and presents it to the user, as shown in Figure 4.1(a). We have modified the map file as shown in Figure 4.1(b).

```
List = List
List$first = List.first
Node = Node
Node$next = Node.next
```

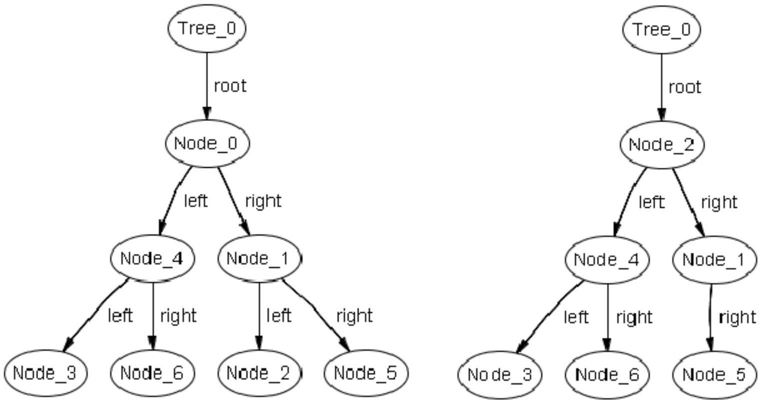
(a) Default static mapping

```
List = SimpleList
List$first = SimpleList.first
Node = Node
Node$next = Node.next
```

(b) Modified static mapping

Figure 4.1: Excerpt of high-level static mapping file before and after modification

Figure 4.2 shows the tree before and after deletion, with correctly implemented code. Figure 4.3 on the next page shows the tree after deletion of the root, when the `root = n2` statement is not executed.



(a) Before deletion (b) After correct deletion

Figure 4.2: Visualization of tree before and after correct deletion of the root node

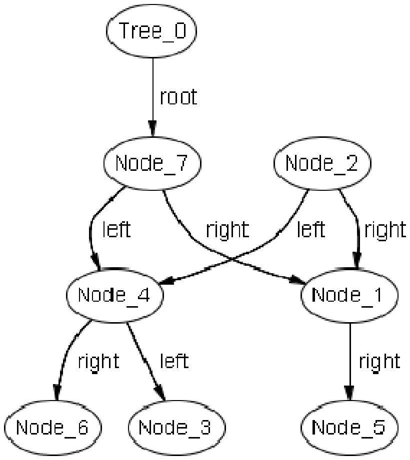


Figure 4.3: Visualization of tree after deletion of the root node, with an error of omission in the code. `Node_7` represents the temporary node in the `swapNodes()` method.

Chapter 5

Embee: Implementation and Analysis

Listing 5.1 Excerpt from `StateDumperThreads.java`, showing how to connect to a second virtual machine executing the target code. In this example, the target class is referenced by `javaClassName`. The JPDA classes can be accessed by including the `tools.jar` archive in the program's classpath; this archive is found in the Java installation's `lib` directory

```
//import com.sun.jdi.Bootstrap; com.sun.jdi.VirtualMachine; com.sun.jdi.connect.Connector;
//com.sun.jdi.connect.LaunchingConnector;
...
LaunchingConnector connect = Bootstrap.virtualMachineManager().defaultConnector();
Map connectorArguments = connect.defaultArguments();
Connector.Argument main = (Connector.Argument) connectorArguments).get("main");
main.setValue(javaClassName);
...
VirtualMachine vm = connect.launch(connectorArguments);
```

```
sig Node {
  next : Node
}
```

(a) Specification of binary `next` relation

```
class Node {
  Node next;
}
```

(b) Implementation of binary relation in (a)

```
sig Tree {
  next : Node -> Node
}
```

(c) Specification of ternary `next` relation

Figure 5.1: Sample specification and implementation of a binary relation; sample specification of a ternary relation

5.1 Complexity and Performance

5.1.1 Definition of Terms

The following terms...:

scope	The maximum number of objects...
$\boldsymbol{R}$	The number of relations...
$\boldsymbol{r_i}$	The i^{th} relation in the specification, $1 \leq i \leq R$.
arity$(\boldsymbol{r_i})$	The arity of relation r_i...
$\boldsymbol{N}$	The total number...

Given the calculated arities of a particular specification's relations, and the scope at a specific breakpoint, Equation 5.1 can be used to determine N.

$$N = S \times scope + \sum_{i=1}^{R} scope^{arity(r_i)} \tag{5.1}$$

The combined complexity of all four steps is

$$O(N) + O(nN) + O(N^2) + O(F)$$

Again, these steps are completed once for every breakpoint in the target program's execution; therefore, the overall upper bound becomes

$$\begin{aligned} & b \times O(N) + b \times O(nN) + b \times O(N^2) + b \times O(F) \\ = \; & O(bN + bnN + bN^2 + bF) \end{aligned}$$

The vector $[x_0\ x_1]$ represents the two possible atoms of type `X`. With our naming scheme, x_0 represents `X_0` and x_1 represents `X_1`. The binary relation itself is represented by a two-dimensional bit matrix where a 1 in position $[i,j]$ means that there is a mapping between the i^{th} atom of `X` and the j^{th} atom of `Y`:

$$\begin{bmatrix} r_{00} & r_{01} \\ r_{10} & r_{11} \end{bmatrix} \quad \begin{bmatrix} \texttt{X\_0->Y\_0} & \texttt{X\_0->Y\_1} \\ \texttt{X\_1->Y\_0} & \texttt{X\_1->Y\_1} \end{bmatrix}$$

Now, consider a fact stating that relation `r` is total, i.e.,

```
all x : X | some y : Y | x.r = y
```

The CNF formula for our example fact, in scope 2, is

$$\neg(((x_0 \wedge r_{00}) \vee (x_1 \wedge r_{10})) \wedge \neg((x_0 \wedge r_{01}) \vee (x_1 \wedge r_{11})))\wedge$$
$$\neg(\neg((x_0 \wedge r_{00}) \vee (x_1 \wedge r_{10})) \wedge ((x_0 \wedge r_{01}) \vee (x_1 \wedge r_{11})))$$

Table 5.1 contains...

Table 5.1: Running times for each phase and total running time of Embee

Test Case			Running Time (m:ss)				
Object Model	Scope	Number of Breakpoints	Phase 1	Phase 2	Phase 3		Total
					First 16	Last 4	
List	20	20	0:07	0:32	0:12	06:39	07:30
Graph	20	19[a]	0:07	1:27	0:35	44:10	46:19
Tree	20	20	0:04	1:20	0:21	06:04	07:49

[a] Breakpoints occur after the addition of each edge, i.e., the first breakpoint does not occur until the second node is added.

...upper bound on Embee's performance:

$$\text{upper bound is} \begin{cases} O(bN^2) & \text{if } scope \leq 16 \\ O(bF) & \text{if } scope > 16 \end{cases}$$

Chapter 6

Summary and Conclusions

6.1 Summary

6.2 Future Work

6.3 Conclusion

Bibliography

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[2] Gert Florijn. RevJava: Design critiques and architectural conformance checking for Java software. Technical report, Software Engineering Research Centre (SERC), May 2002.

[3] Roel Wuyts. *A Logic Meta-Programming Approach to Support the Co-Evolution of Object-Oriented Design and Implementation.* PhD thesis, Vrije Unversiteit Brussel (VUB), January 2001.

Appendix A

Alloy Analyzer

A.1 Documentation

Package: `alloy.api`

`AlloyRunner`

This class provides...

To do an analysis...

`analyzeCommand`	Run the actual...
`prepareSpec`	Parse...
`translateCommand`	Translate...

Appendix B

Additional Analysis

B.1 Calculation of Arity

Examples of arity calculations are shown in Table B.1. These calculations can be performed using either the equations listed in Figure B.1 on the next page...

Table B.1: Example arity calculations

relation r_i	$arity(r_i)$	arity equations used
`f : A`	2	(B.1), (B.2a)
`f : option A`	2	(B.1), (B.2a)
`f : A -> A`	3	(B.1), (B.2b), (B.3a)
`f : A -> ? B`	3	(B.1), (B.2b), (B.3a)
`f : A -> B -> C`	4	(B.1), (B.2b), (B.3c), (B.3a)
`f : A -> B ? -> ! C`	4	(B.1), (B.2b), (B.3a), (B.3c)
`f : A -> B -> C -> D`	5	(B.1), (B.2b), (B.3c), (B.3a)

Given:

v is a variable of type `<var>`, i.e., an `id` (identifier)
m is a multiplicity expression of type `<multexpr>`
r is a relation of type `<relation>`, i.e., $r = v : m$
e_1, e_2, ... are expressions of type `<expr>` in m
x is an optional set multiplicity modifier of type `<setmult>`
y, z are optional relation multiplicity modifiers of type `<mult>`

The arity equations are:

$$\text{arity}(r) = 1 + \text{arity}(m), \text{ where } r \text{ is of the form } v : m \tag{B.1}$$

$$\mathit{arity}(m) = \begin{cases} 1 & \text{if } m \text{ is of the form } x\ v \quad \text{(B.2a)} \\ \mathit{arity}(e_1) + \mathit{arity}(e_2) & \text{if } m \text{ is of the form } e_1\ y \texttt{ -> } z\ e_2 \quad \text{(B.2b)} \end{cases}$$

$$\mathit{arity}(e) = \begin{cases} 1 & \text{if } e \text{ is of the form } \texttt{id} \quad \text{(B.3a)} \\ \mathit{arity}(e_1) & \text{if } e \text{ is of the form } (e_1) \quad \text{(B.3b)} \\ \mathit{arity}(e_1) + \mathit{arity}(e_2) & \text{if } e \text{ is of the form } e_1 \texttt{ -> } e_2 \quad \text{(B.3c)} \end{cases}$$

Figure B.1: Equations to compute arity of relations

B.2 Comparison of N

B.2.1 Reasoning about N in terms of n

It is possible to determine an upper bound on the size of N, relative to the size of n. To do this, we re-examine Equation 5.1.

From Equation 5.1, we have:

$$N = S \times \mathit{scope} + \sum_{i=1}^{R} \mathit{scope}^{\mathit{arity}(r_i)}$$

In the worst-case, the scope is equal to the total number of objects that exist at a particular breakpoint, i.e., $scope = n$.

$$N = Sn + \sum_{i=1}^{R} n^{arity(r_i)}$$

We can expand the summation to

$$N = Sn + n^{arity(r_1)} + n^{arity(r_2)} + ... + n^{arity(r_R)}$$

Because...

$$O(N) = O(Sn) + O(n^{arity(r_1)}) + O(n^{arity(r_2)}) + ... + O(n^{arity(r_R)})$$

We assume that all R relations in the specification have the same arity, and that this arity is represented by a value $x \geq 2$. Therefore...

$$\begin{aligned} O(N) &= O(Sn) + R \times O(n^x) \\ &= O(Sn) + O(Rn^x) \end{aligned} \tag{B.4}$$

Equation B.4 demonstrates...

Because both S and R are finite numbers, it is possible to further reduce Equation B.4 to

$$\begin{aligned} O(N) &= O(n) + O(n^x) \\ &= O(n^x) \end{aligned} \tag{B.5}$$

Therefore...

B.3 Estimation of F

For example, Table B.2 contains the values of F...

Table B.2: Estimate of Boolean formula size, determined by number of Boolean operators (“and”, “or”, “not”)

Example 1 - List				
scope	N	0 Facts	1 Fact	2 Facts
1	4	23	34	43
2	12	197	657	729
3	24	671	13,799	15,200
4	40	1,731	91,435	96,771

Example 2 - Graph					
scope	N	Facts	1 Fact	2 Facts	3 Facts
1	—	—	—	—	—
2	16	185	1,005	1,783	2,181
3	42	674	66,722	118,250	142,328
4	88	1,787	635,811	1,153,063	1,319,611

Example 3 - Tree						
scope	N	0 Facts	1 Fact	2 Facts	3 Facts	4 Facts
1	7	39	78	93	103	104
2	22	367	1,601	2,487	2,629	2,715
3	45	1,283	38,528	73,472	76,196	76,568
4	76	3,359	234,595	456,459	466,979	468,087

B.3.1 Test Series

Table B.3 summarizes...

Table B.3: Test series for evaluating the running time of conformance checking

Series Name	Example	S	R	arity(r_i)	Number of Facts	*scope* $= n$	N	Number of Tests
E1F0	1 *List*	2	2	2, 2	0	1,2,...,40	4 - 3,280	40
E1F1					1	1,2,...,32	4 - 1,984	32
E1F2					2	1,2,...,31	4 - 1,984	31
E2F0	2 *Graph*	2	2	2, 3	0	2,3,...,40	16 - 65,680	39
E2F1					1	2,3,...,40	16 - 33,856	39
E2F2					2	2,3,...,34	16 - 33,856	33
E2F3					3	2,3,...,24	16 - 14,448	23
E3F0	2 *Tree*	3	4	2, 2, 2, 2	0	1,2,...,40	7 - 6,520	40
E3F1					1	1,2,...,40	7 - 6,520	40
E3F2					2	1,2,...,32	7 - 4,192	32
E3F3					3	1,2,...,32	7 - 4,192	32
E3F4					4	1,2,...,32	7 - 4,192	32
Total Number of Tests (Conformance Checks)								412