

Model Checking: Theory and Practice
Bogor Companion

Matthew B. Dwyer John Hatcliff Robby

August 5, 2004

Contents

0	Introduction	1
0.1	Getting Started with Bogor	1
0.1.1	Installing Bogor	1
0.1.2	Running Bogor	1
0.1.3	Organizing Your Workspace	2
0.2	Bogor Trackers and Forums	2
1	Foundations	3
1.1	BIR	3
1.1.1	A simple BIR system – no notes	3
1.1.2	States and transitions	3
1.2	Interleavings, Schedules, and Computation Trees	6
1.3	Executing (Simulating) BIR Systems with Bogor	7
1.4	Verifying BIR Systems with Bogor	10
A	Appendix	11
A.1	BIR Concrete Syntax	11

List of Figures

1.1	Bogor BIR Editor	4
1.2	A BIR system for illustrating instruction interleavings	7
1.3	User-guided Simulation: SumToN	8
1.4	Counter-example Display: SumToN	8
A.1	BIR Concrete Syntax	12

List of Tables

Chapter 0

Introduction

This document accompany the “Model Checking: Theory and Practice” book.¹ It provides additional commentaries for the materials in the book, some guided exercises, and additional instructions for using the Bogor software model checking framework. With the exception of this chapter, the rest of the chapters follow the same structure as the book.

0.1 Getting Started with Bogor

The Bogor software model checking framework is an ongoing project at Kansas State University. For more information about Bogor, visit the Bogor website.²

Bogor user interface is built around the Eclipse universal tool platform. For more information about Eclipse, visit the Eclipse website.³

0.1.1 Installing Bogor

Follow the instructions in the Bogor User Manual from the Bogor website to install and to start using Bogor in Eclipse. The Bogor manual is also included in the Bogor distribution. You can also view it using the Eclipse help system once you installed Bogor.

Although you can run Bogor using the Eclipse platform without JDT, it is recommended that you use the Eclipse SDK instead. Some of the exercises in the book requires you to extend the Bogor framework. Since Bogor is built around Eclipse, it is easier to develop Bogor extension in Eclipse.

0.1.2 Running Bogor

Bogor may require additional heap memory when model checking programs. If you have more RAM in your computer, you can give Bogor/Eclipse more

¹<http://book.projects.cis.ksu.edu/modelchecking>

²<http://bogor.projects.cis.ksu.edu>

³<http://www.eclipse.org>

memory as follows.

```
eclipse -vmargs -mxYYYm
```

Where `YYY` is the amount of memory that Bogor/Eclipse can use (in Megabytes – MB).

0.1.3 Organizing Your Workspace

It is advised that you assign your workspace to a different directory than the default one as follows.

```
eclipse -data directory -vmargs -mxYYYm
```

Where `directory` is the path of the directory that you want to use for your workspace.

0.2 Bogor Trackers and Forums

To submit Bogor bug reports, patches, and feature requests, go to the Bogor trackers webpage.⁴ You can use the Bogor forums⁵ for discussions about Bogor.

⁴http://projects.cis.ksu.edu/tracker/?group_id=8

⁵http://projects.cis.ksu.edu/forum/?group_id=8

Chapter 1

Foundations

1.1 BIR

The current implementation of BIR does not support all the specified features of BIR. For example:

- Type casting should be used for actions or expressions that involve integer types with different ranges. For example, the assignment action $x := 1$; has a type error if x not an int type (e.g., `int wrap(0, 255)`). That is, it should be $x := (\text{int wrap}(0, 255)) 1$; instead.

Note that BIR model file should have `.bir` as its extension.

The well-formed-ness checker included in the Bogor editor will notify you if you have syntax errors or type errors in your model. Figure 1.1 presents the Bogor well-formed-ness checker in action; it notifies you if you have a type casting error and no main thread error described above. Note that you can look at the Tasks View to see all of the errors in the model.

1.1.1 A simple BIR system – no notes

1.1.2 States and transitions

Exercises

2. Let us denote the PCs of the producer thread P and the consumer thread C as pc_P and pc_C , respectively. The following three states are unreachable from the initial state:
 - $[pc_P \mapsto 1, pc_C \mapsto 0, full \mapsto \text{"false"}, locked \mapsto \text{"false"}]$, `locked` is `"false"`, however, the Producer's PC is at `loc1` that is only reachable from `loc0` that sets `locked` to `"true"`.
 - $[pc_P \mapsto 0, pc_C \mapsto 1, full \mapsto \text{"false"}, locked \mapsto \text{"false"}]$, `locked` is `"false"`, however, the Consumer's PC is at `loc1` that is only reachable from `loc0` that sets `locked` to `"true"`.

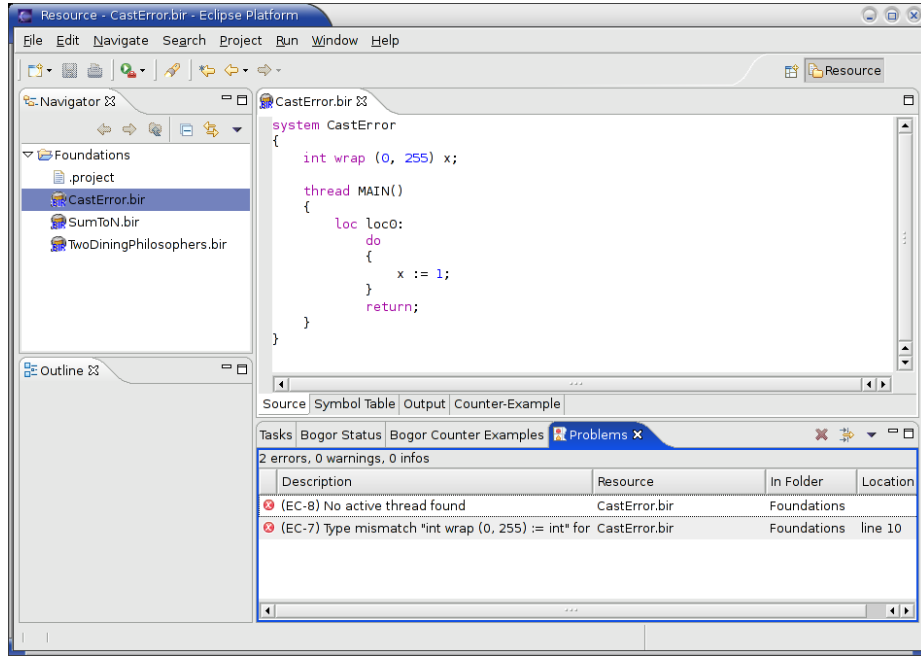


Figure 1.1: Bogor BIR Editor

- $[pc_P \mapsto 3, pc_C \mapsto 0, full \mapsto \text{"false"}, locked \mapsto \text{"false"}]$, both locked and full are "false", however, the Producer's PC is at loc3 that is only reachable from loc0 and loc2 that set locked and full to "true".

3. (a) The following three transitions are enabled:

- the transition $\alpha_{P:0}$ at state

$$[pc_P \mapsto 0, pc_C \mapsto 0, full \mapsto \text{"false"}, locked \mapsto \text{"false"}]$$

because its guard !locked holds,

- the transition $\alpha_{C:0}$ at state

$$[pc_P \mapsto 0, pc_C \mapsto 0, full \mapsto \text{"false"}, locked \mapsto \text{"false"}]$$

because its guard !locked holds, and

- the transition $\alpha_{P:1b}$ at state

$$[pc_P \mapsto 1, pc_C \mapsto 0, full \mapsto \text{"false"}, locked \mapsto \text{"true"}]$$

because its guard !full holds.

(b) The following three transitions are disabled:

- the transition $\alpha_{P:1a}$ at state

$$[\text{pc}_P \mapsto 1, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"true"}]$$

because its guard full does not hold,

- the transition $\alpha_{C:1a}$ at state

$$[\text{pc}_P \mapsto 0, \text{pc}_C \mapsto 1, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"true"}]$$

because its guard full does not hold, and

- the transition $\alpha_{P:1b}$ at state

$$[\text{pc}_P \mapsto 1, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"true"}]$$

because its guard !full does not hold.

- The state $s = [\text{pc}_P \mapsto 1, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"true"}]$ has $\text{current}(s) \neq \text{enabled}(s)$, because $\text{current}(s) = \{\alpha_{P:1a}, \alpha_{P:1b}, \alpha_{C:0}\}$ and $\text{enabled}(s) = \{\alpha_{P:1b}\}$.
- (a) The following sequence of six states forms an execution for the system:

$$\begin{array}{l} [\text{pc}_P \mapsto 0, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"false"}] \\ \xrightarrow{P:0} [\text{pc}_P \mapsto 1, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"true"}] \\ \xrightarrow{P:1b} [\text{pc}_P \mapsto 2, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"true"}] \\ \xrightarrow{P:2} [\text{pc}_P \mapsto 3, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"true"}] \\ \xrightarrow{P:3} [\text{pc}_P \mapsto 0, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"false"}] \\ \xrightarrow{P:0} [\text{pc}_P \mapsto 1, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"true"}]. \end{array}$$

- (b) The following sequence of six states does not form an execution for the system:

$$\begin{array}{l} [\text{pc}_P \mapsto 0, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"false"}] \\ \xrightarrow{P:0} [\text{pc}_P \mapsto 1, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"true"}] \\ \xrightarrow{P:1b} [\text{pc}_P \mapsto 2, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"false"}, \text{locked} \mapsto \text{"true"}] \\ \xrightarrow{P:2} [\text{pc}_P \mapsto 3, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"true"}] \\ \xrightarrow{C:0} [\text{pc}_P \mapsto 3, \text{pc}_C \mapsto 1, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"true"}] \\ \xrightarrow{P:3} [\text{pc}_P \mapsto 0, \text{pc}_C \mapsto 1, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"false"}], \end{array}$$

because the transition $\alpha_{C:0}$ is disabled at state:

$$[\text{pc}_P \mapsto 3, \text{pc}_C \mapsto 0, \text{full} \mapsto \text{"true"}, \text{locked} \mapsto \text{"true"}].$$

1.2 Interleavings, Schedules, and Computation Trees

Exercises

1. Another error schedule for the SumToN when $N = 2$ is as follows:

$$\begin{array}{l}
 [x \mapsto 1, t1 \mapsto 0, t2 \mapsto 0] \ . \\
 \xrightarrow{1:0} [x \mapsto 1, t1 \mapsto 1, t2 \mapsto 0] \\
 \xrightarrow{1:1} [x \mapsto 1, t1 \mapsto 1, t2 \mapsto 1] \\
 \xrightarrow{1:2} [x \mapsto 2, t1 \mapsto 1, t2 \mapsto 1] \\
 \xrightarrow{1:0} [x \mapsto 2, t1 \mapsto 2, t2 \mapsto 1] \\
 \xrightarrow{0:0} [x \mapsto 2, t1 \mapsto 2, t2 \mapsto 1]
 \end{array}$$

2. Any schedule that infinitely executes only the transitions of Thread1 and Thread2 will never violate the assertion for any value of N (or any schedule that begins with the transition of Thread0).

3. Another error schedule for the SumToN when $N = 2$ is as follows:

$$\begin{array}{l}
 [x \mapsto 1, t1 \mapsto 0, t2 \mapsto 0] \\
 \xrightarrow{1:0} [x \mapsto 1, t1 \mapsto 1, t2 \mapsto 0] \\
 \xrightarrow{1:1} [x \mapsto 1, t1 \mapsto 1, t2 \mapsto 1] \\
 \xrightarrow{2:0} [x \mapsto 1, t1 \mapsto 1, t2 \mapsto 1] \\
 \xrightarrow{1:2} [x \mapsto 2, t1 \mapsto 1, t2 \mapsto 1] \\
 \xrightarrow{2:1} [x \mapsto 2, t1 \mapsto 1, t2 \mapsto 2] \\
 \xrightarrow{2:2} [x \mapsto 3, t1 \mapsto 1, t2 \mapsto 2] \\
 \xrightarrow{3:0} [x \mapsto 3, t1 \mapsto 1, t2 \mapsto 2]
 \end{array}$$

7. An example of such a trace is as follows:

$$\begin{array}{l}
 [full \mapsto \text{"false"}, locked \mapsto \text{"false"}] \\
 \xrightarrow{P:0} [full \mapsto \text{"false"}, locked \mapsto \text{"true"}] \\
 \xrightarrow{P:1b} [full \mapsto \text{"false"}, locked \mapsto \text{"true"}] \\
 \xrightarrow{P:2} [full \mapsto \text{"true"}, locked \mapsto \text{"true"}] \\
 \xrightarrow{P:3} [full \mapsto \text{"true"}, locked \mapsto \text{"false"}] \\
 \xrightarrow{C:0} [full \mapsto \text{"true"}, locked \mapsto \text{"true"}] \\
 \xrightarrow{C:1a} [full \mapsto \text{"true"}, locked \mapsto \text{"true"}] \\
 \xrightarrow{C:2} [full \mapsto \text{"false"}, locked \mapsto \text{"true"}] \\
 \xrightarrow{C:3} [full \mapsto \text{"false"}, locked \mapsto \text{"false"}] \\
 \xrightarrow{P:0} \dots
 \end{array}$$

```

system SumToN {
  const PARAM { N = 1; }
  typealias byte int wrap (0,255);

  byte x := 1;
  byte t1;
  byte t2;

  active thread Thread1() {
    loc loc0:
    when x != (byte) 0 do {
      t1 := x;
    }
    goto loc1;

    loc loc1:
    do { t2 := x; }
    goto loc2;

    loc loc2:
    do { x := t1 + t2; }
    goto loc0;
  }

  active thread Thread2() {
    loc loc0:
    when x != (byte) 0 do {
      t1 := x;
    }
    goto loc1;

    loc loc1:
    do { t2 := x; }
    goto loc2;

    loc loc2:
    do { x := t1 + t2; }
    goto loc0;
  }

  active thread Thread0() {
    loc loc0:
    do {
      assert (x != (byte) PARAM.N);
    }
    return;
  }
}

```

Figure 1.2: A BIR system for illustrating instruction interleavings

1.3 Executing (Simulating) BIR Systems with Bogor

Figure 1.2 presents the BIR code for the `SumToN` example in [?]. Run your Eclipse/Bogor, and create a simple project called `Chapter1`. Create a new file in the project called `SumToN.bir`, and copy and paste the content of Figure 1.2. (*Note: for the next example, we have modified the BIR constant `N` to 2.*) After that, run the Bogor user-guided simulation mode by right-clicking the BIR editor and then selecting “User-guided simulation.”

Figure 1.3 presents the screen shot of Bogor running in guided simulation mode. Note that it stops at the first non-deterministic choice. You can select the next transition to execute in the Bogor “Choose” dialog. Choose “Thread#1...” by double-clicking it to expose the assertion error in the model. As this point, the first temporary variable `t1` has been assigned the value of `x`. Choose “Thread#1...” twice more to bring the value of `x` to 2. Now the transition in `Thread0` can be provoked to expose an error. Choose “Thread#0...” once to make the model checker execute the assertion `assert (x != (byte) Param.N)`. Once Bogor encounters an error, it stops the simulation mode and writes a trail file called `SumToN.bir.bogor-trails`. Now, close the simulation mode, and double-click the trail file to open the Bogor counter-example display.

Bogor’s counter-example viewer allows you to go through each step and see how the transitions affect the state and finally lead the assertion failue. In

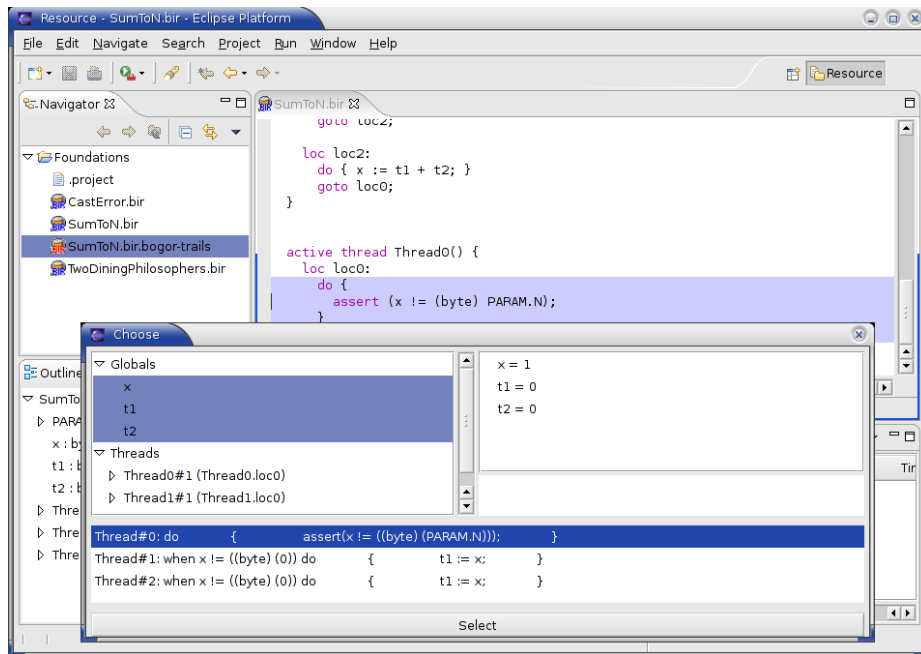


Figure 1.3: User-guided Simulation: SumToN

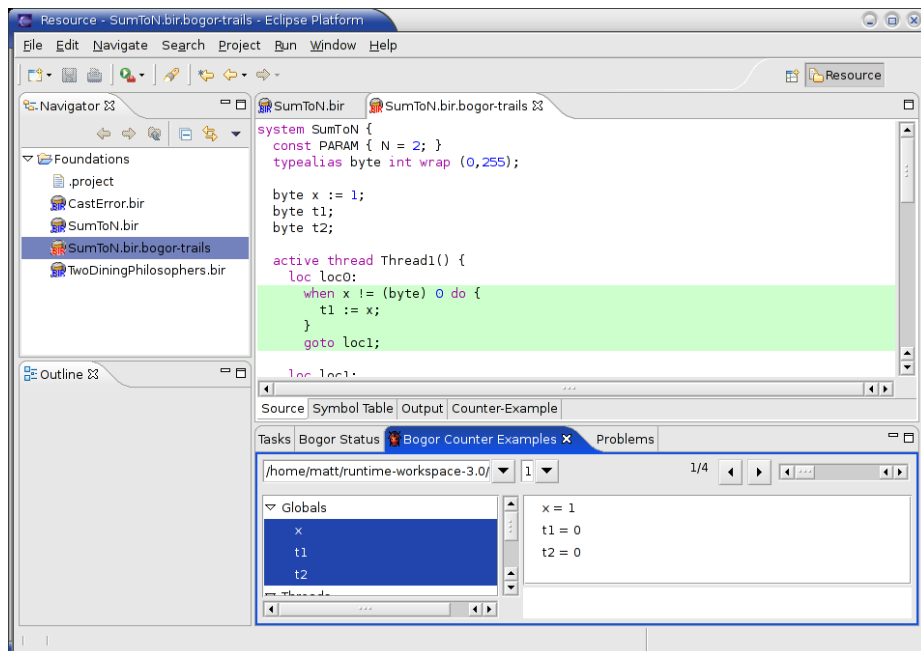


Figure 1.4: Counter-example Display: SumToN

Figure 1.4 we see the system as it first starts, with all global variable values fresh at 0 and `Thread1` poised to assign to the first temporary variable `t1`.

Exercises

- For small values of `N`, Bogor’s random simulation mode does often find assertion violations in a small number of steps. Below we list the results of informally performing random executions of `SumToN`. The results listed with an *A* are assertion violations from `Thread0`.

N	Depth of error			
1	3 (A)	3 (A)	3 (A)	3 (A)
2	88 (D)	5 (A)	46 (D)	58 (D)
3	49 (D)	43 (D)	40 (D)	49 (D)
4	55 (D)	37 (D)	43 (D)	61 (D)

A note about the results listed as *D*: the `SumToN` system actually contains another type of error besides `Thread0`’s assertion violation. The `loc0` guards on both `Thread1` and `Thread2` prevent these threads from making progress when `x = 0`. Because the domains over which our global variables range are limited (only 8 bits per byte), their values “wrap around.” Thus when `x` is assigned the result of `128 + 128`, it gets the new value 0. All threads are then stuck, and the resulting condition is called *deadlock*. This is in fact much more likely to occur in `SumToN` than an assertion violation when $N \geq 2$; consequently, the simulations usually encounter a deadlock and terminate before a “real” error happens.

Try removing the `loc0` guard on the two worker threads to see if Bogor can quickly find assertion violations with, for example, $N = 3$.

- Finding a non-deadlock error schedule with $N = 5$ is extremely difficult. Using the automated analysis of Bogor (which has not yet been introduced here) we were able to find an assertion violation after 395 transitions—we will not list it here.

Using a guided simulation, there is no way to be certain that 395-step error schedules are minimal without manually trying every schedule with 394 or fewer steps.

- This is a similar situation to the solution for Exercise 2. With $N = 7$, the the shortest of the violations which our automated tool could find required 604 steps. Again, we cannot be certain that this schedule is minimal without manually testing all shorter ones.

1.4 Verifying BIR Systems with Bogor

The basic depth-first search algorithm

Exercises

1. For the state labeled (2), suppose the following:
 - $s_0 = [\text{pc}_1 \mapsto 0, \text{pc}_2 \mapsto 0, \text{fork1} \mapsto \text{"false"}, \text{fork2} \mapsto \text{"false"}]$,
 - $s_1 = [\text{pc}_1 \mapsto 1, \text{pc}_2 \mapsto 0, \text{fork1} \mapsto \text{"true"}, \text{fork2} \mapsto \text{"false"}]$,
 - $s_2 = [\text{pc}_1 \mapsto 2, \text{pc}_2 \mapsto 0, \text{fork1} \mapsto \text{"true"}, \text{fork2} \mapsto \text{"true"}]$,
 - $s_3 = [\text{pc}_1 \mapsto 3, \text{pc}_2 \mapsto 0, \text{fork1} \mapsto \text{"true"}, \text{fork2} \mapsto \text{"false"}]$,
 - $s_4 = [\text{pc}_1 \mapsto 0, \text{pc}_2 \mapsto 1, \text{fork1} \mapsto \text{"false"}, \text{fork2} \mapsto \text{"true"}]$, and
 - $s_5 = [\text{pc}_1 \mapsto 1, \text{pc}_2 \mapsto 1, \text{fork1} \mapsto \text{"true"}, \text{fork2} \mapsto \text{"true"}]$.

Assuming that the algorithm visits the tree from right to left, then the data structures are as follows: $\text{state} = s_3$, $\text{seen} = \{s_0, s_1, s_2\}$, and $\text{stack} = [s_4, s_5]$ (i.e., s_5 is at the top of the stack).

Appendix A

Appendix

A.1 BIR Concrete Syntax

$\langle system \rangle$::=	"system" $\langle system-id \rangle$ "{" $\langle system-member \rangle^*$ "}"
$\langle system-member \rangle$::=	$\langle type-alias \rangle$ $\langle global-var \rangle$ $\langle thread \rangle$
$\langle type-alias \rangle$::=	"typealias" $\langle type-alias-id \rangle$ $\langle type \rangle$ ";"
$\langle global-var \rangle$::=	$\langle type \rangle$ $\langle var-id \rangle$ ";"
$\langle type \rangle$::=	"boolean" "int" $\langle int-range \rangle?$ $\langle type-alias-id \rangle$
$\langle int-range \rangle$::=	"wrap"? "(" $\langle int-lit \rangle$ "," $\langle int-lit \rangle$ ")"
$\langle thread \rangle$::=	"thread" $\langle thread-id \rangle$ $\langle instance \rangle?$ "(" ")" "{" $\langle location \rangle^+$ "}"
$\langle instance \rangle$::=	"[" $\langle int-lit \rangle$ "]"
$\langle location \rangle$::=	"loc" $\langle loc-id \rangle$ ":" $\langle transformation \rangle^+$
$\langle transformation \rangle$::=	$\langle guard \rangle?$ "do" $\langle visibility \rangle?$ "{" $\langle action \rangle^*$ "}" $\langle jump \rangle$
$\langle guard \rangle$::=	"when" $\langle exp \rangle$
$\langle visibility \rangle$::=	"visible" "invisible"
$\langle jump \rangle$::=	"goto" $\langle loc-id \rangle$ ";" "return" ";"
$\langle exp \rangle$::=	$\langle lit-exp \rangle$ $\langle var-exp \rangle$ $\langle unary-exp \rangle$ $\langle binary-exp \rangle$ $\langle paren-exp \rangle$ $\langle cast-exp \rangle$
$\langle lit-exp \rangle$::=	"true" "false" $\langle int-lit \rangle$
$\langle var-exp \rangle$::=	$\langle var-id \rangle$
$\langle unary-exp \rangle$::=	$\langle unary-op \rangle$ $\langle exp \rangle$
$\langle unary-op \rangle$::=	"+" "-" "!"
$\langle binary-exp \rangle$::=	$\langle exp \rangle$ $\langle binary-op \rangle$ $\langle exp \rangle$
$\langle binary-op \rangle$::=	"+" "-" "*" "/" "%" "==" "!=" ">" ">=" "<" "<=" "&&" " "
$\langle paren-exp \rangle$::=	"(" $\langle exp \rangle$ ")"
$\langle cast-exp \rangle$::=	"(" $\langle type \rangle$ ")" $\langle exp \rangle$

Figure A.1: BIR Concrete Syntax