

Software Model Checking Using Bogor

- a Modular and Extensible Model Checking Framework

*3rd Estonian Summer School in
Computer and System Science (ESSCaSS'04)*

Slide Set 01: Bogor Overview

<http://bogor.projects.cis.ksu.edu>

<http://www.cis.ksu.edu/~hatcliff/ESSCaSS04>

John Hatcliff

Matthew B. Dwyer

Robby

SAnToS Laboratory, Kansas State University, USA

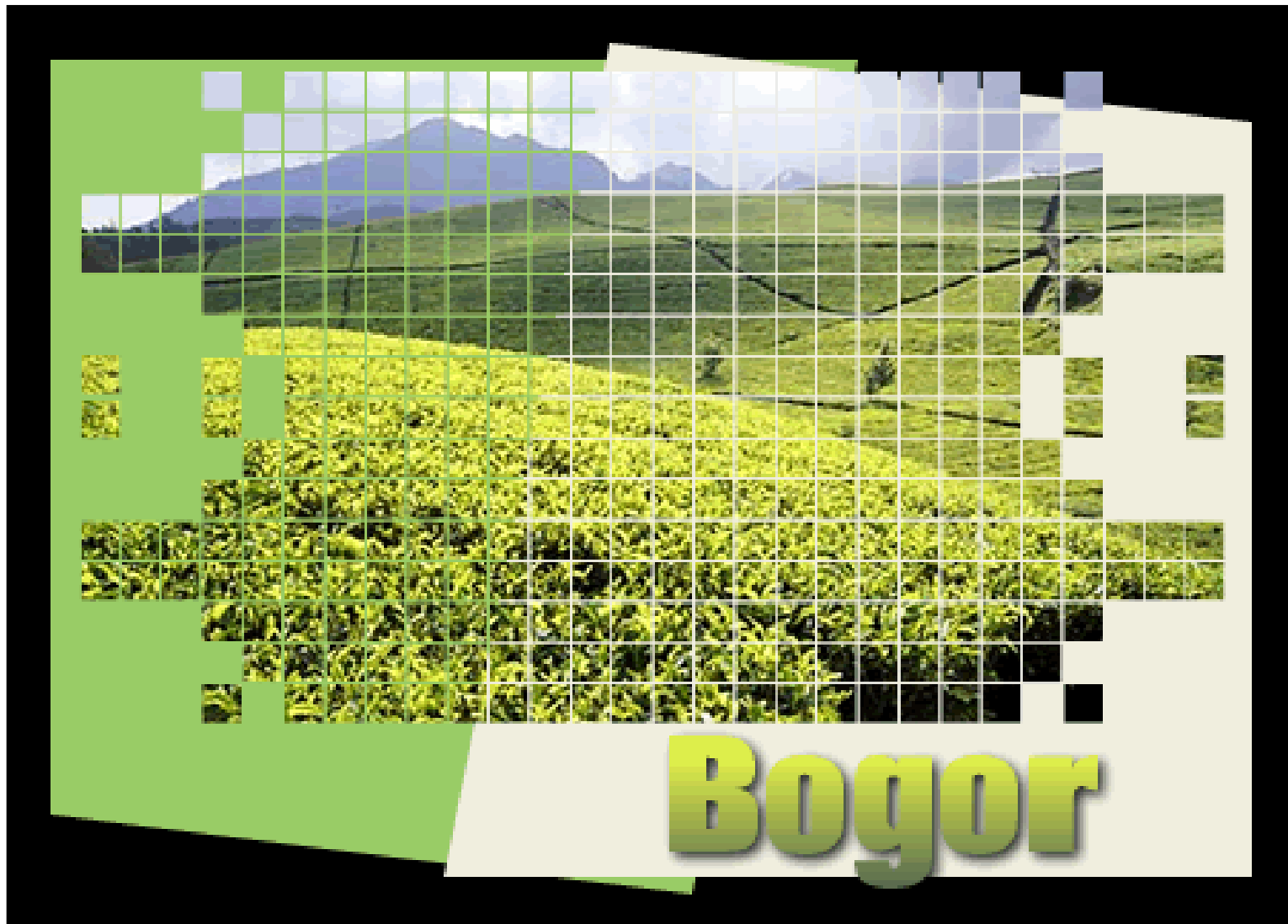
Support

US Army Research Office (ARO)
US National Science Foundation (NSF)
US Department of Defense
Advanced Research Projects Agency (DARPA)

Boeing
Honeywell Technology Center
IBM
Intel

Lockheed Martin
NASA Langley
Rockwell-Collins ATC
Sun Microsystems

Bogor



Bogor - Software Model Checking Framework

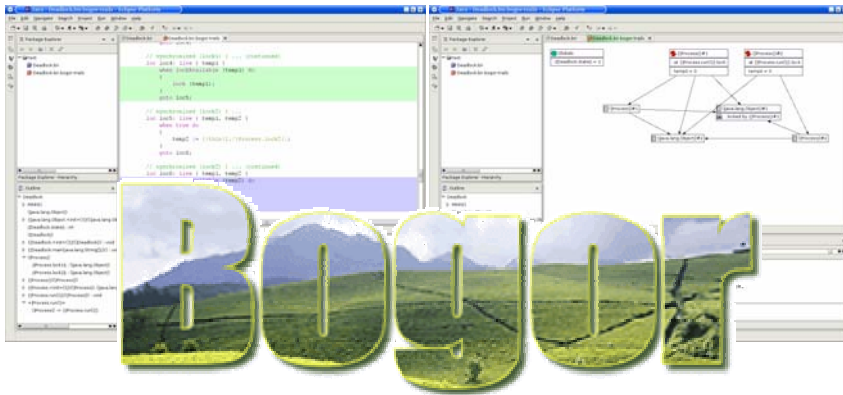
The screenshot displays the Eclipse IDE with the Bogor framework. The Package Explorer on the left shows the project structure: `test` containing `Deadlock.bir` and `Deadlock.bir.bogor-trails`. The Outline window shows the class hierarchy for `Deadlock`, including `MAIN()`, `(Ljava.lang.Object)`, `{|java.lang.Object.<init>()|}(|java.lang.Object)`, `/|Deadlock.state| : int`, `(|Deadlock|)`, `{|Deadlock.<init>()|}(|Deadlock|) : void`, `{|Deadlock.main(java.lang.String[])|}() : void`, `(|Process|)`, `/|Process.lock1| : (|java.lang.Object|)`, `/|Process.lock2| : (|java.lang.Object|)`, `{|Process|}(|Process|)`, `{|Process.<init>()|}(|Process|), (|java.lang.Object)`, `{|Process.run()|}(|Process|) : void`, and `+|Process.run()|+ (|Process|) -> {|Process.run()|}`.

The main editor shows a diagram of the model state. It includes a `Globals` box with `/|Deadlock.state| = 2`. Two process boxes are shown: `{|Process|}#1` at `{|Process.run()|}.loc6` with `temp0 = 0`, and `{|Process|}#2` at `{|Process.run()|}.loc4` with `temp0 = 0`. Arrows indicate that process #1 holds `(|java.lang.Object|)#1` (locked by process #1) and process #2 holds `(|java.lang.Object|)#2`. The diagram illustrates a deadlock state where both processes are blocked from proceeding.

The Counter-Example window at the bottom shows the path to the deadlock state: `/home/robby/Documents/Workspace/test/Deadlock`, `1`, `33/35`, `Globals`, `/|Deadlock.state|`, `Threads`, `{|Process|}#1 (Enabled, {|Process|})`, `{|Process|}#2 (Enabled, {|Process|})`, `{|Process.run()|}.loc4`, `[|this|] = (|Process|)#2`, `temp0 = 0`, and `temp1 = (|java.lang.Object|)#2`.

The Bogor logo is displayed at the bottom right of the image.

Bogor - Direct support for OO software



Extensive support for checking concurrent OO software

Direct support for...

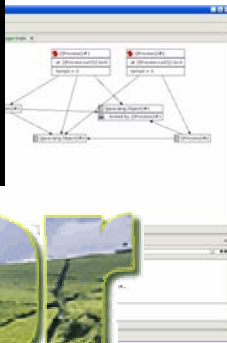
- *unbounded* dynamic creation of threads and objects
- automatic memory management (garbage collection)
- virtual methods, ...
- ..., exceptions, etc.
- *supports virtually all of Java*

Software targeted algorithms...

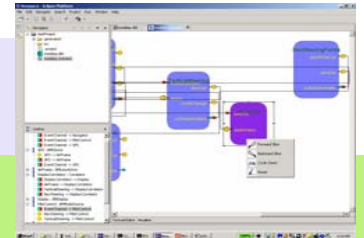
- thread & heap symmetry
- compact state representation
- partial order reduction techniques driven by
 - object escape analysis
 - locking information

Bogor - Eclipse-based Tool Components

Next generation of
Bandera Java Model-
checking Tool Set



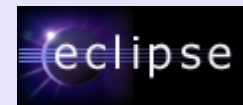
Cadena
CORBA Component Model
verification



SpEx
JML Verification, etc.



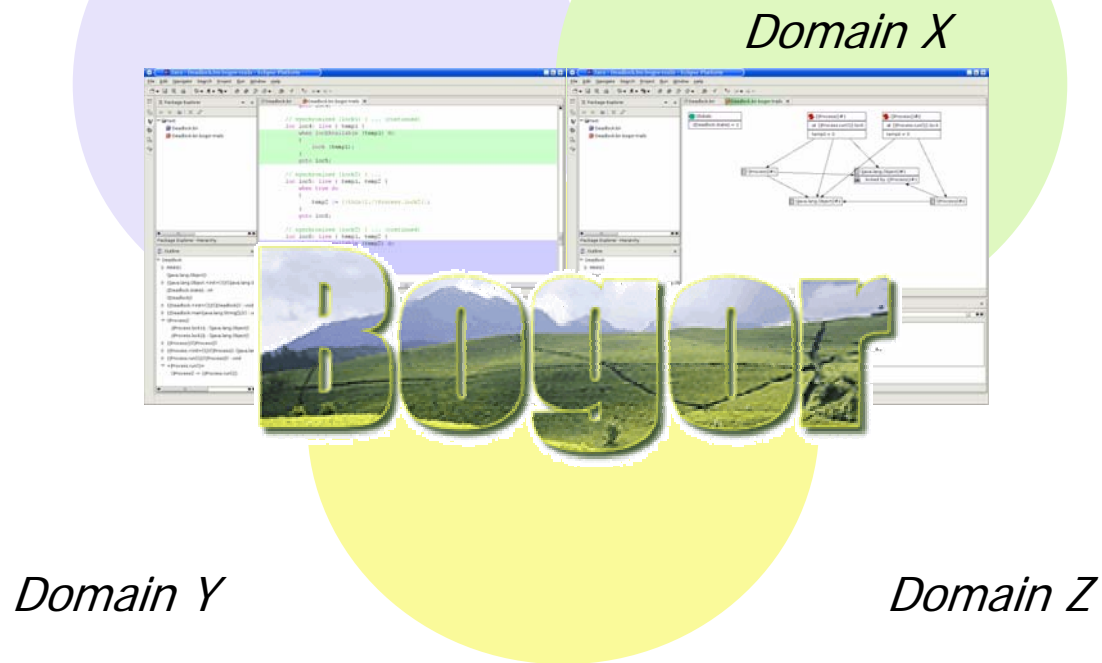
Tool Development
Framework



Architecture allows encapsulation/integration with other verification tools using IBM's *Eclipse* Integrated Development Environment

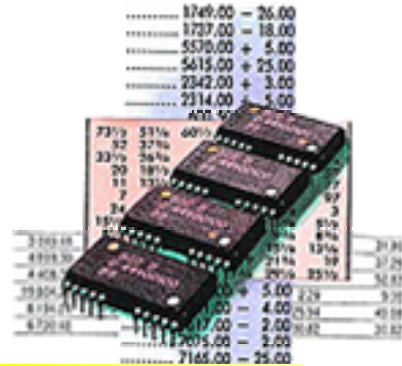
Bogor - Domain Specific Model-Checking

Modeling language and Algorithms
easily customized to different domains



Extensible modeling language and plug-in architecture allows Bogor to be customized to a variety of application domains

Variety of Application Domains



Hardware



Device Drivers



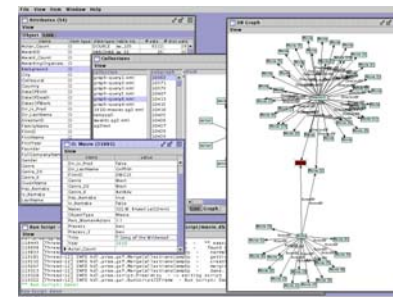
Avionics



Telephony



Automotive



GUI

Leveraging Domain Knowledge

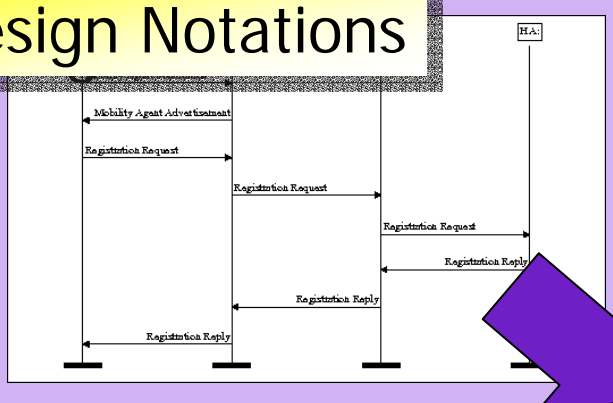


Lucent *Path Star*
Telephone Switch

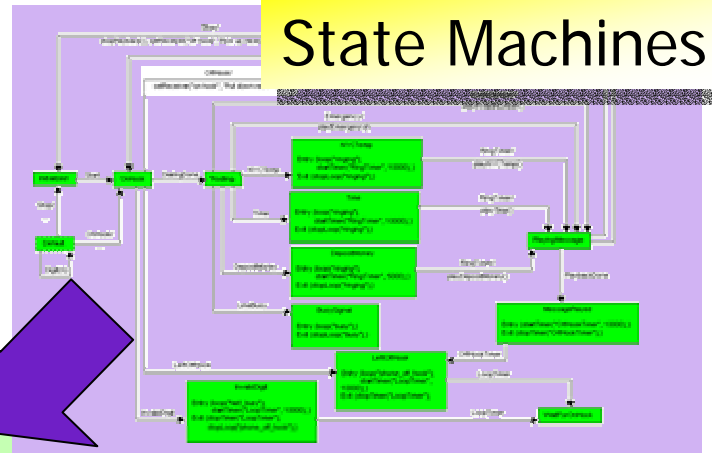
- Holzmann developed a customized model extraction from C to Spin
- Translation using pattern matching of *particular domain idioms*
- In essence, an abstract machine for a particular domain
- Very effective at finding subtle defects

Variety of System Descriptions

Design Notations



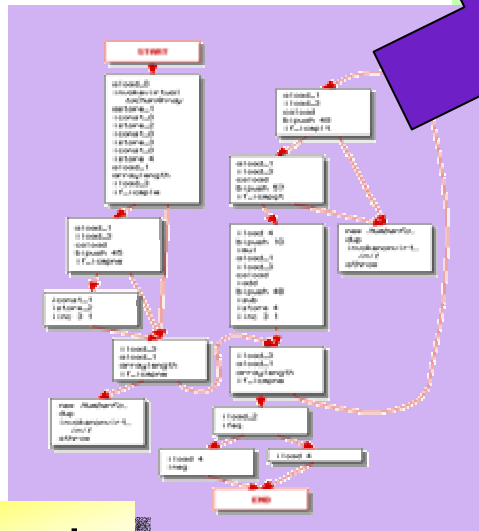
State Machines



Model Checker

Different levels of abstraction!

Byte code



```
Project1 - Microsoft Visual Basic [design] - [AMACS [Code] [Read Only]]
ControlTimer - Timer
If InStr(EnabledSystemOptions$, ",pH,") > 0 Then
    'pH
    CH = "p"
    NUM_AVG = 30
    RGE = "04" +/- 1V
    DLY = 50 'delay between readings in milliseconds
    Call ReadpH(CH, NUM_AVG, RGE, DLY)
End If
If AMACS.SystemStatus.Caption = "Stopped..." Then
    MousePointer = 0
    Exit Sub
End If
If InStr(EnabledSystemOptions$, ",Orp,") > 0 Then
    'ORP
    CH = "r"
    NUM_AVG = 30
    RGE = "03" +/-500mV
    DLY = 50 'delay between readings in milliseconds
    Call ReadOrp(CH, NUM_AVG, RGE, DLY)
End If
If AMACS.SystemStatus.Caption = "Stopped..." Then
    MousePointer = 0
    Exit Sub
End If
If InStr(EnabledSystemOptions$, "Temperature") > 0 Then
    'Temperature
    CH = "t"
End If
```

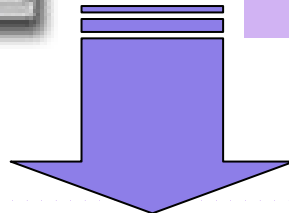
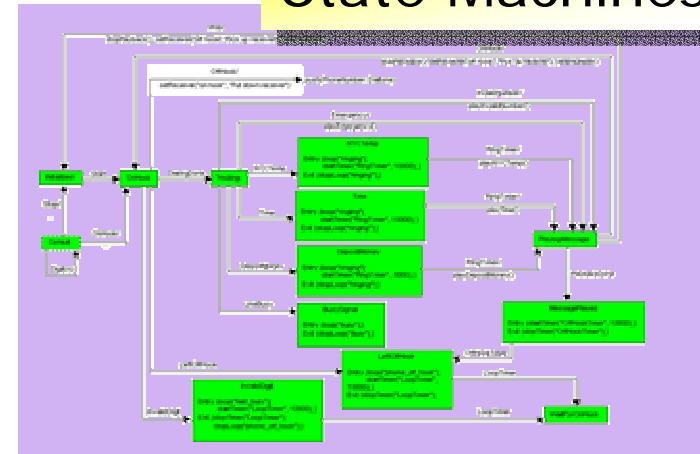
Source code

The Goal

Avionics



State Machines



Model-checking
Engine

Domain & Abstraction
Extensions

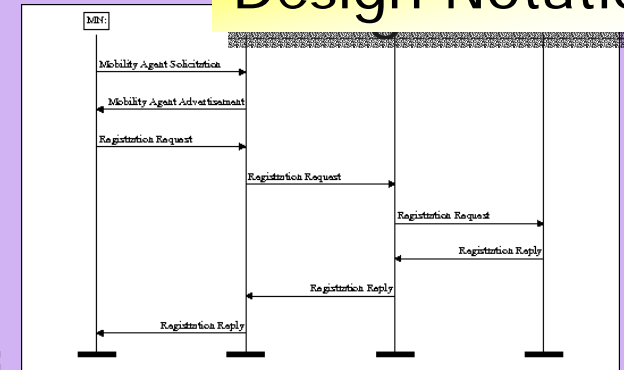
Abstract machine tailored to *domain* and *level of abstraction*

The Goal

Automotive



Design Notations



Model-checking
Engine

Domain & Abstraction
Extensions

Abstract machine tailored to domain and level of abstraction

Customization Mechanisms

Bogor -- Extensible Modeling Language

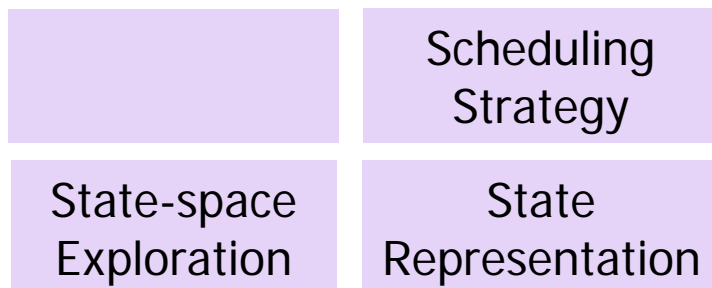
Threads,
Objects,
Methods,
Exceptions, etc.

+

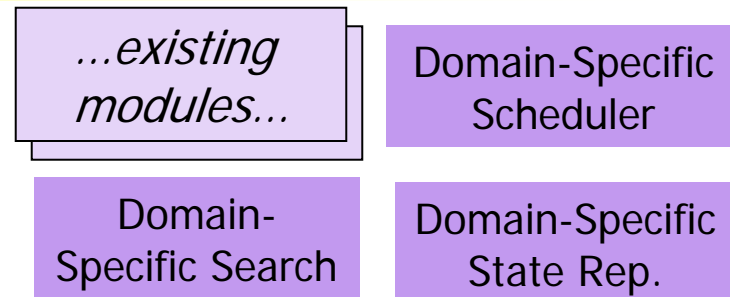
Domain-Specific
Abstractions

Core Modeling Language

Bogor -- Customizable Checking Engine Modules



Core Checker Modules



Customized Checker Modules

Outline



Overview

Concept of Bogor Extensions

- Extending the syntax
- Adding semantics via Java

● Bogor Modeling Language and UI

- Example: Dining philosophers
- Demo: Bogor UI and BIR Case Wizard

Conclusions

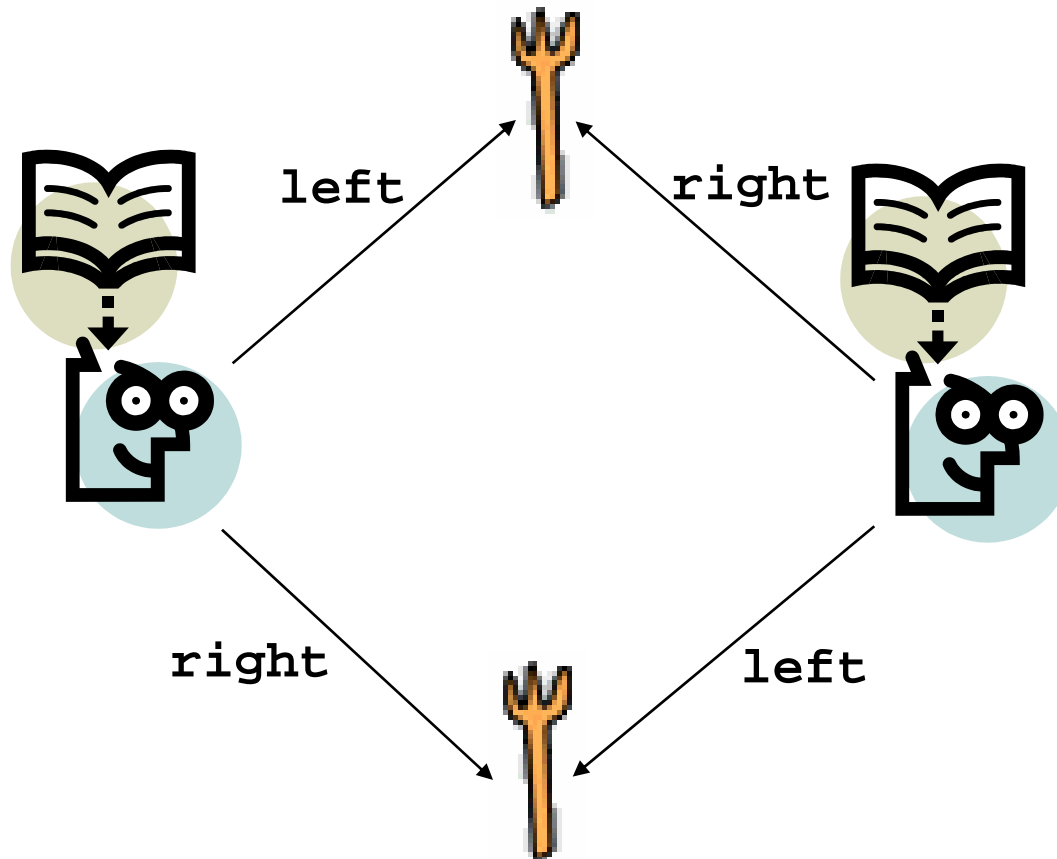
- The utility of a customizable model-checking platform

Bogor Modeling Language – BIR

BIR = Bandera Intermediate Representation

- Used as the intermediate language for the Bandera Tool Set for model-checking Java programs
- Guarded command language
 - **when** <condition> **do** <command>
- Native support for a variety of object-oriented language features
 - dynamically created objects and threads, exceptions, methods, inheritance, etc.

An Example – 2 Dining Philosophers



A BIR Example – 2 Dining Philosophers

```
system TwoDiningPhilosophers {
  record Fork { boolean isHeld; }

  main thread MAIN() {
    Fork fork1;
    Fork fork2;

    loc loc0:
      do {
        // create forks
        fork1 := new Fork;
        fork2 := new Fork;

        // start philosophers
        start Phil(fork1, fork2);
        start Phil(fork2, fork1);
      } return;
  }
}
```

```
thread Phil(Fork left, Fork right) {
  loc loc0: // take left fork
  when !left.isHeld do {
    left.isHeld := true;
  } goto loc1;

  loc loc1: // take right fork
  when !right.isHeld do
  { right.isHeld := true; }
  goto loc2;

  loc loc2: // put right fork
  do { right.isHeld := false; }
  goto loc3;

  loc loc3: // put left fork
  do { left.isHeld := false; }
  goto loc0;
}
}
```

A BIR Example – 2 Dining Philosophers

```
system TwoDiningPhilosophers {
  record Fork { boolean isHeld; }

  main thread MAIN() {
    Fork fork1;
    Fork fork2;

    loc loc0:
      do {
        // create forks
        fork1 := new Fork;
        fork2 := new Fork;

        // start philosophers
        start Phil(fork1, fork2);
        start Phil(fork2, fork1);
      } return;
  }
}
```

Uses a record to model forks

```
thread Phil(Fork left, Fork right) {
  loc loc0: // take left fork
  when !left.isHeld do {
    left.isHeld := true;
  } goto loc1;

  loc loc1: // take right fork
  when !right.isHeld do
  { right.isHeld := true; }
  goto loc2;

  loc loc2: // put right fork
  do { right.isHeld := false; }
  goto loc3;

  loc loc3: // put left fork
  do { left.isHeld := false; }
  goto loc0;
}
}
```

A BIR Example – 2 Dining Philosophers

```
system TwoDiningPhilosophers {  
  record Fork { boolean isHeld; }  
  
  -----  
  main thread MAIN() {  
    Fork fork1;  
    Fork fork2;  
  
    loc loc0:  
    do {  
      // create forks  
      fork1 := new Fork;  
      fork2 := new Fork;  
  
      // start philosophers  
      start Phil(fork1, fork2);  
      start Phil(fork2, fork1);  
    } return;  
  }  
}
```

Thread declarations

```
thread Phil(Fork left, Fork right) {  
  loc loc0: // take left fork  
  when !left.isHeld do {  
    left.isHeld := true;  
  } goto loc1;  
  
  loc loc1: // take right fork  
  when !right.isHeld do  
  { right.isHeld := true; }  
  goto loc2;  
  
  loc loc2: // put right fork  
  do { right.isHeld := false; }  
  goto loc3;  
  
  loc loc3: // put left fork  
  do { left.isHeld := false; }  
  goto loc0;  
}  
}
```

A BIR Example – 2 Dining Philosophers

```
system TwoDiningPhilosophers {
  record Fork { boolean isHeld; }

  main thread MAIN() {
    Fork fork1;
    Fork fork2;

    loc loc0:
      do {
        // create forks
        fork1 := new Fork;
        fork2 := new Fork;

        // start philosophers
        start Phil(fork1, fork2);
        start Phil(fork2, fork1);
      } return;
  }
}
```

Local variable declarations

```
thread Phil(Fork left, Fork right) {
  fork
  {
    left.isHeld := true;
  } goto loc1;

  loc loc1: // take right fork
  when !right.isHeld do
  { right.isHeld := true; }
  goto loc2;

  loc loc2: // put right fork
  do { right.isHeld := false; }
  goto loc3;

  loc loc3: // put left fork
  do { left.isHeld := false; }
  goto loc0;
}
}
```

A BIR Example – 2 Dining Philosophers

```
system TwoDiningPhilosophers {
  record Fork { boolean isHeld; }

  main thread MAIN() {
    Fork fork1;
    Fork fork2;

    loc loc0:
      do {
        // create forks
        Control locations
        // start philosophers
        start Phil(fork1, fork2);
        start Phil(fork2, fork1);
      } return;
  }
}
```

```
thread Phil(Fork left, Fork right) {
  loc loc0: // take left fork
  when !left.isHeld do {
    left.isHeld := true;
  } goto loc1;

  loc loc1: // take right fork
  when !right.isHeld do
  { right.isHeld := true; }
  goto loc2;

  loc loc2: // put right fork
  do { right.isHeld := false; }
  goto loc3;

  loc loc3: // put left fork
  do { left.isHeld := false; }
  goto loc0;
}
```

A BIR Example – 2 Dining Philosophers

Guarded transformations

*...aka “guarded transitions”,
“guarded commands”*

When condition
is true

```
thread Phil(Fork left, Fork right) {  
  loc loc0: // take left fork  
  when !left.isHeld do {  
    left.isHeld := true;  
  } goto loc1;  
  
  loc loc1: // take right fork  
  when !right.isHeld do  
  { right.isHeld := true; }  
  goto loc2;  
  
  loc loc2: // put right fork  
  do { right.isHeld := false; }  
  goto loc3;  
  
  loc loc3: // put left fork  
  do { left.isHeld := false; }  
  goto loc0;  
}  
}
```

Trivially true
guards

Execute these
statement(s)
atomically

A BIR Example – 2 Dining Philosophers

Demo

- Bogor BIR Editor
 - syntax highlighting
 - well-formed-ness checker
- Bogor Counter-example Display
 - states and transitions navigation
 - heap visualization
- Configuring Bogor

Outline



Overview

● Bogor Modeling Language and UI

- Example: Dining philosophers
- Demo: Bogor UI and BIR Case Wizard

Concept of Bogor Extensions

- Extending BIR language with new operations
- Supporting Java
- Functional sub-language

Conclusions

- The utility of a customizable model-checking platform

BIR: Extensible Modeling Language

Motivation

- Variety of application domains and system level descriptions often work at different level of abstractions
 - want to be able to bridge the gap between system descriptions and BIR with ease
- BIR extensions...
 - can be extended on-demand
 - minimize changes and maximize reuse of Bogor components
 - parser/lexer, symbol table, AST, type system, *etc.*

BIR Extensions

```
extension Channel for MyChannel {  
  // declaration of abstract types  
  typedef type<'a>;  
  
  // declaration of abstract expressions  
  expdef Channel.type<'a> create<'a>(int);  
  expdef boolean isEmpty<'a>(Channel.type<'a>);  
  expdef 'a getFirst<'a>(Channel.type<'a>);  
  
  // declaration of abstract actions/commands  
  actiondef send<'a>(Channel.type<'a>, 'a);  
  actiondef removeFirst<'a>(Channel.type<'a>);  
}
```

BIR allows introduction
of new abstract types
and operations

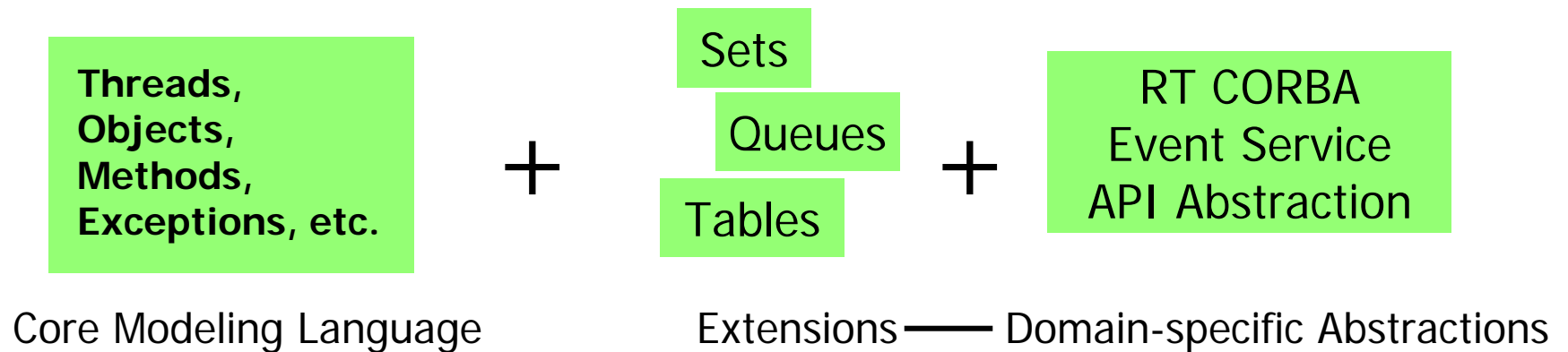
```
Channel.type<int> chan;  
int x;  
...  
chan := Channel.create<int>(5); // ch 5 slots  
...  
Channel.send<int>(chan, 0); // send 0  
...  
x := Channel.getFirst<int>(chan); // recv 1st  
Channel.removeFirst<int>(chan);
```

Sample usage

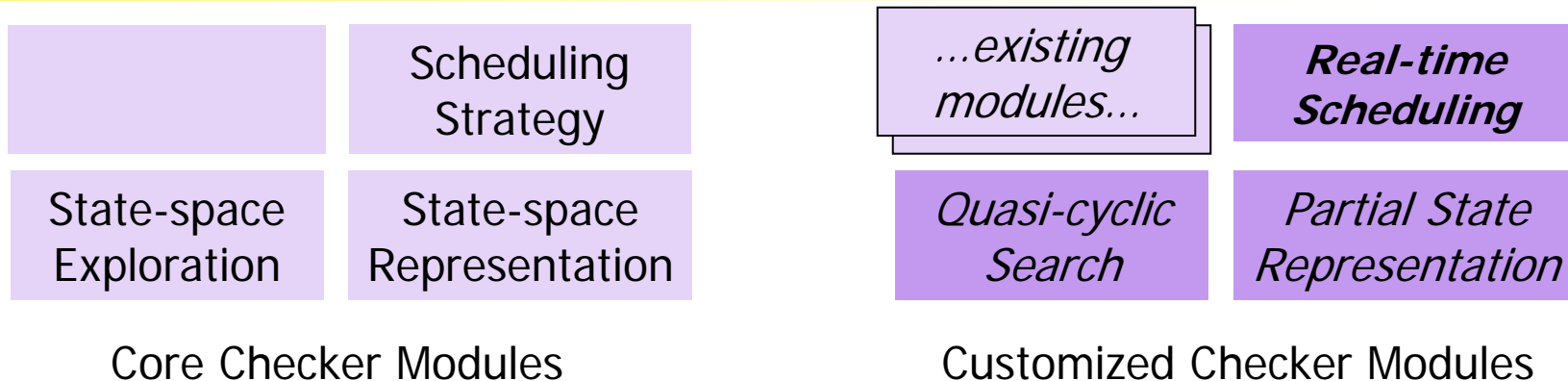
wait till next session!

Domain-Specific Model-Checking

Bogor -- Extensible Modeling Language



Bogor -- Customizable Checking Engine Modules

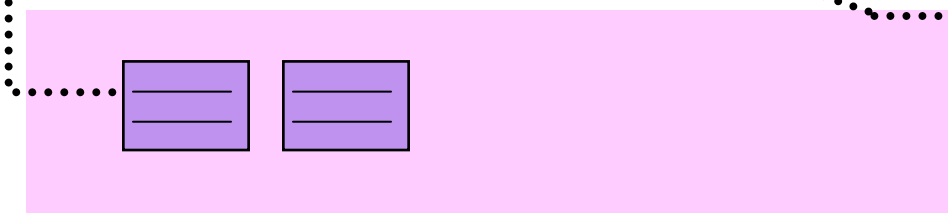


Extension Implementation

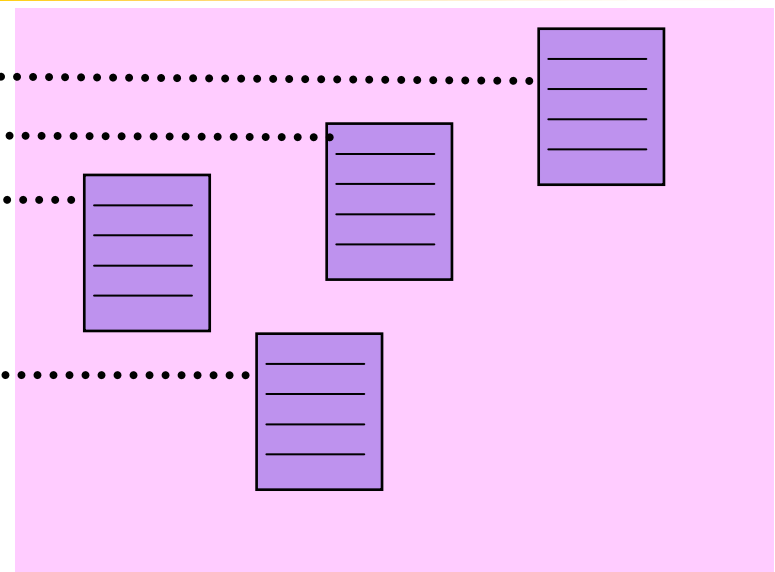
Extensions are implemented by associating each item in extension interface with Java methods that provide the semantics for the item (or state-vector storage representation in case of state).

```
extension Set for SetModule
{
  ••• typedef type<'a>;
  expdef Set.type<'a> create<'a>('a ...);
  expdef 'a choose<'a>(Set.type<'a>);
  actiondef add<'a>(Set.type<'a>, 'a);
  expdef boolean forAll('a -> boolean, Set.type<'a>);
}
```

Extension Implementation



Java implementation of set value and linearized (state-vector) representation.



Java methods implementing actions and expressions

Supporting Java

- BIR provides features commonly found in modern programming languages
 - Dynamic creation of objects and threads, automatic memory management, *etc.*
- Java-to-BIR translator
 - Uses the Soot framework from Sable Research at McGill University
 - Document:
http://projects.cis.ksu.edu/docman/?group_id=10

BIR Functional Sub-language

Motivation

- wants to allow complex queries of states while guaranteeing purity
 - very useful for specification purposes

Syntax and semantics

- similar to other functional languages (SML, *etc.*)
- ...but only supports first-order functions

BIR Functional Sub-language

```
record Node {  
  Node next;  
  int x;  
}  
  
fun sortedList(Node n)  
  returns boolean =  
  let  
    Node next = n.next  
  in  
    next == null ?  
      true  
    : (n.x <= next.x ?  
        sortedList(next)  
        : false);
```

Node for a linked-list
data structure

A recursive function to
determine whether a
given list is sorted
(ascending order)

Outline



Overview

● Concept of Bogor Extensions

- Extending BIR language with new operations
- Supporting Java
- Functional sub-language

Bogor Modeling Language and UI

- Example: Dining philosophers
- Demo: Bogor UI and BIR Case Wizard

Conclusions

- The utility of a customizable model-checking platform

Conclusions

- General purpose state-space reduction strategies have dramatically reduced the cost of model-checking
 - yet it is still quite expensive to apply in many cases
- To obtain further significant reductions, we believe that a variety of domain knowledge can be leveraged to improve model-checking applicability
 - incorporating knowledge about domain specific data structures, scheduling policies, state invariants, etc.
- Bogor is a platform that is specifically designed to...
 - ...be used to obtain domain-specific model-checking engines
 - ...be used for exploring a variety of research directions related to model-checking

Coming Soon for Bogor...

- Sophisticated counterexample display facilities
 - MSCs, abstractions of trace data, etc.
- Incorporation of a variety of forms of coverage information
- Support for a variety of forms of property specification/checking including...
 - Java Modeling Language (JML)
 - LTL/CTL via specification patterns
- Incorporation into next generation of Bandera

...many of these already implemented by not incorporated into distribution.