Bogor: A Flexible Framework for Creating Software Model Checkers

SAnToS Laboratory, Kansas State University, USA

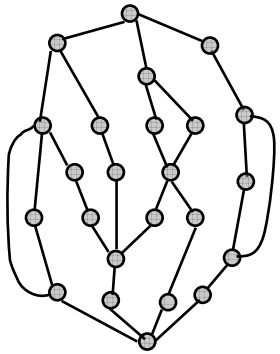
http://bogor.projects.cis.ksu.edu

Robby Matthew B. Dwyer John Hatcliff

Support

US Army Research Office (ARO)
US National Science Foundation (NSF)
US Department of Defense
Advanced Research Projects Agency (DARPA)
Lockheed Martin ATL (Cherry Hill, NJ)
US Air Force Office of Scientific Research (AFOSR)
IBM Eclipse

Model Checking



Finite-state model

$\Box(\Phi \rightarrow \Diamond\, \Omega)$

Specification

Model Checker

OK

or

Error trace

```
Line 5: …
Line 12: …
Line 15:…
Line 21:…
Line 25:…
Line 27:…
   …
Line 41:…
Line 47:…
```

Why Try to Use Model Checking for Software?

- Automatically check, e.g.,
  - invariants, simple safety & liveness properties
  - absence of dead-lock and live-lock,
  - complex event sequencing properties,

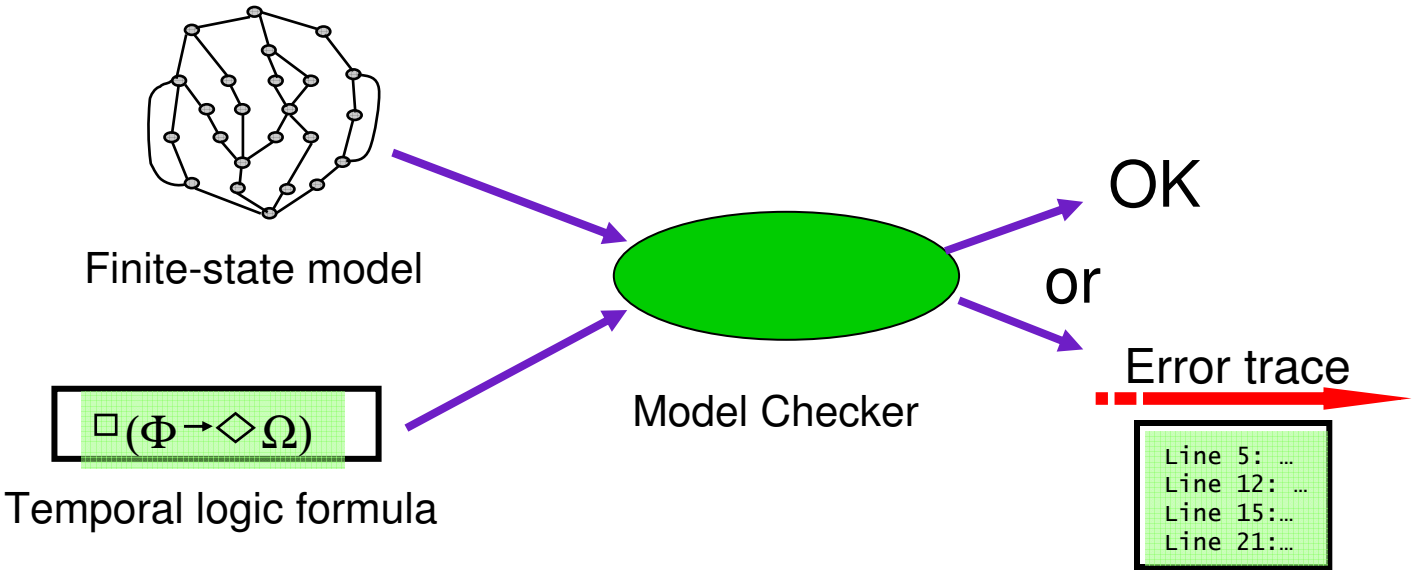
> "Between the window open and the window close, button X can be pushed at most twice."

- In contrast to testing, gives complete coverage – even in presence of concurrency -- by exhaustively exploring all paths in system,
- It's been used for years with good success in hardware and protocol design

This suggests that model-checking can complement existing software quality assurance techniques.

What makes model-checking software difficult?

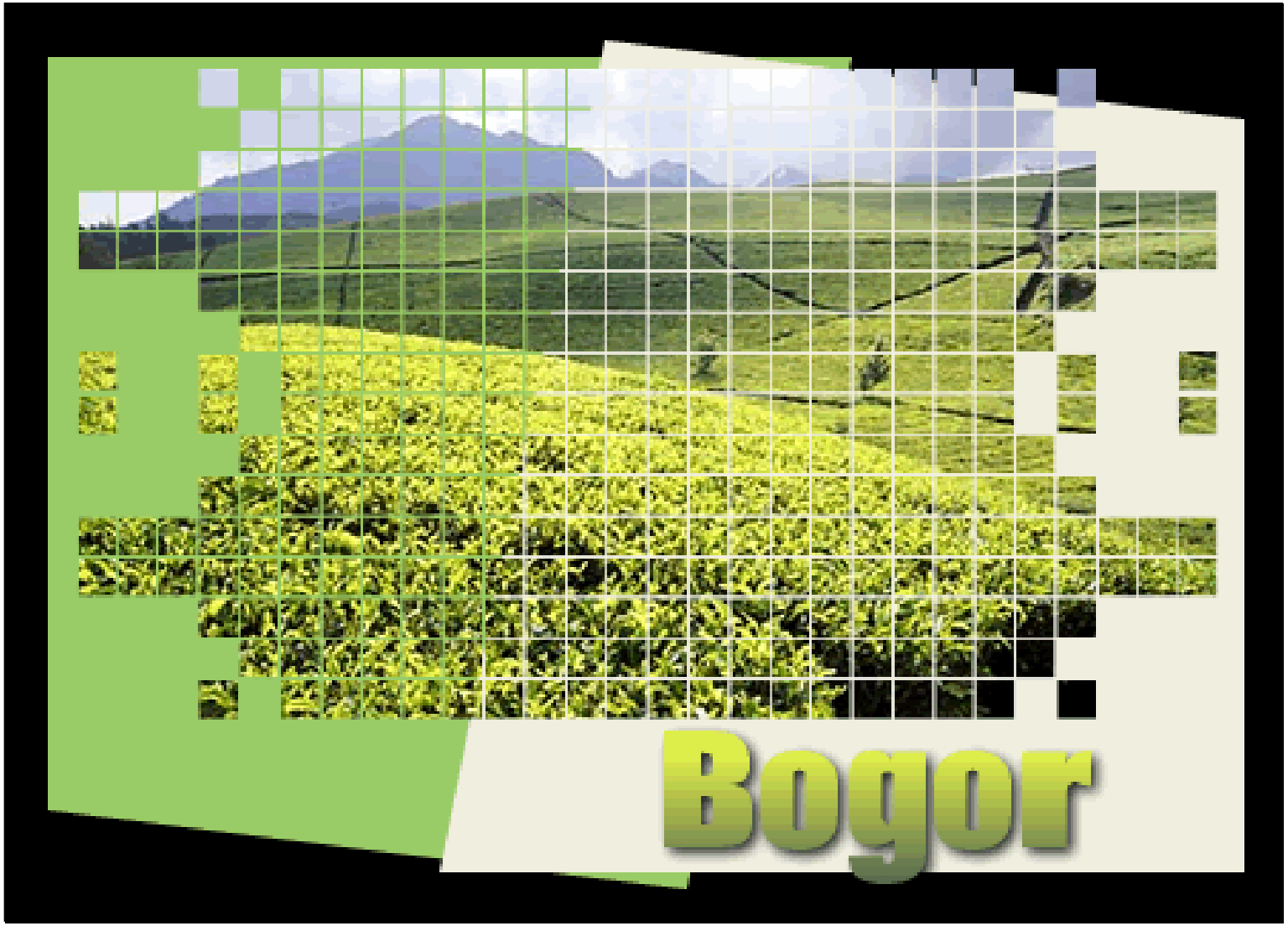
Note: These slides are from my ICSE 2000 talk…



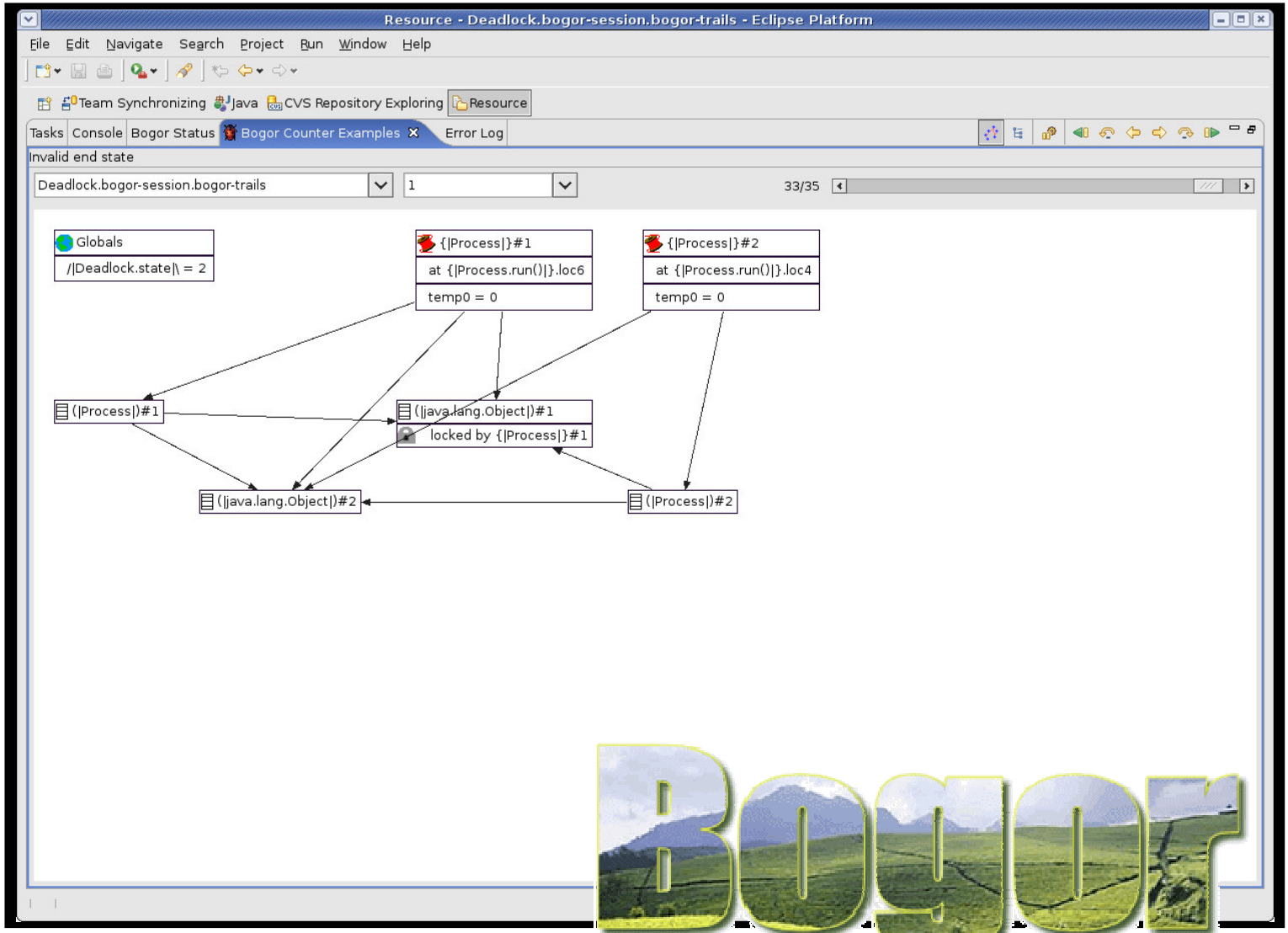
Problems using existing checkers:

- Model construction
- Property specification
- State explosion
- Output interpretation

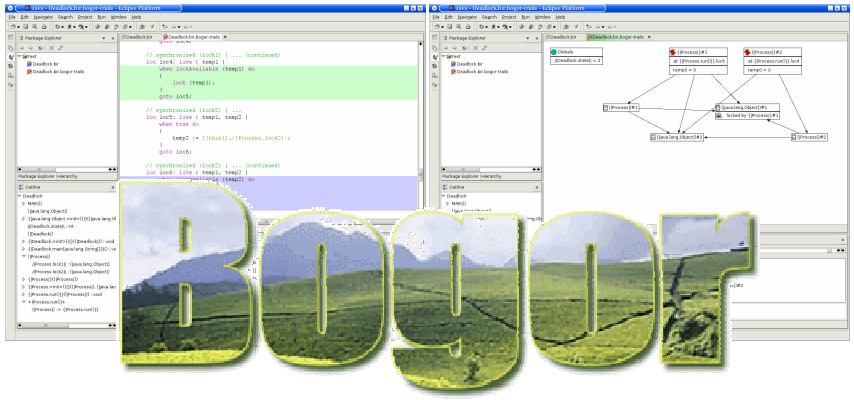
Bogor



Bogor – Software Model Checking Framework



Bogor – Direct support for OO software



Extensive support for checking concurrent OO software

Direct support for…

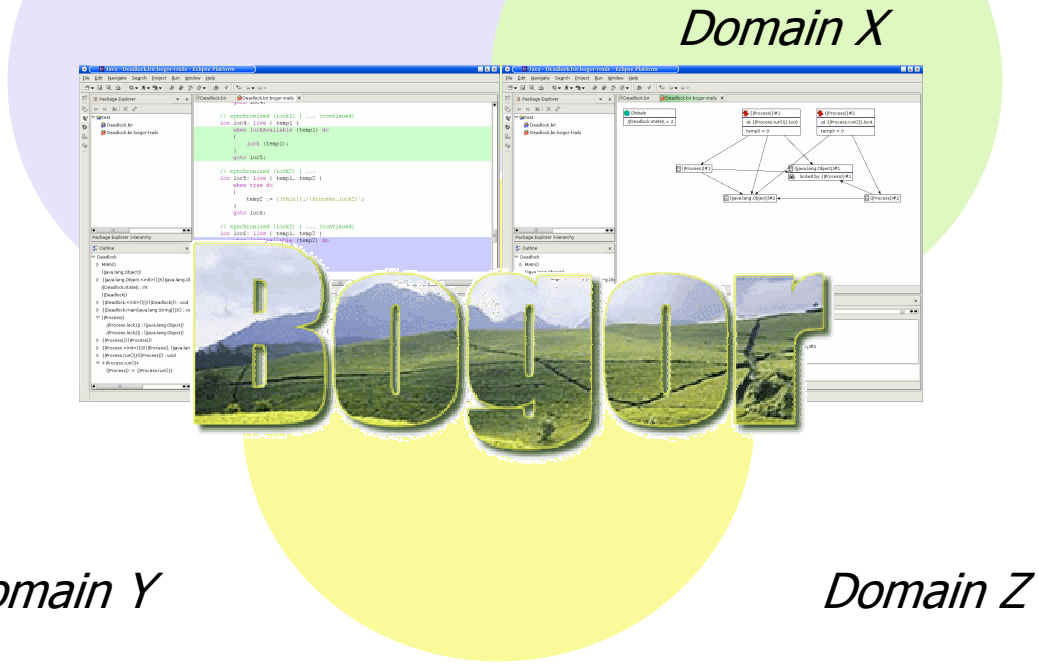
- *unbounded* dynamic creation of threads and objects
- automatic memory management (garbage collection)
- virtual methods, …
- …, exceptions, etc.
- *supports virtually all of Java*

Software targeted algorithms…

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

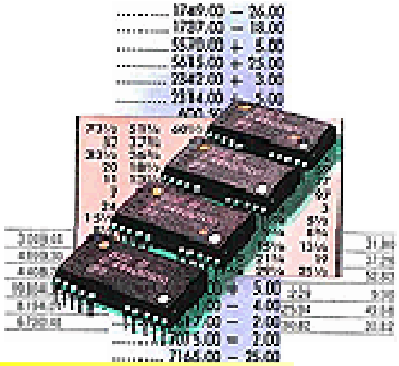
Bogor – Domain Specific Model-Checking

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

System Modeling Problem – Variety of Application Domains



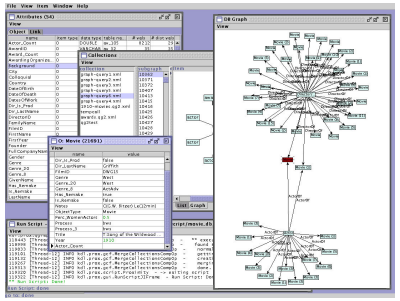
Hardware

Device Drivers

Avionics

Telephony

Automotive



GUI

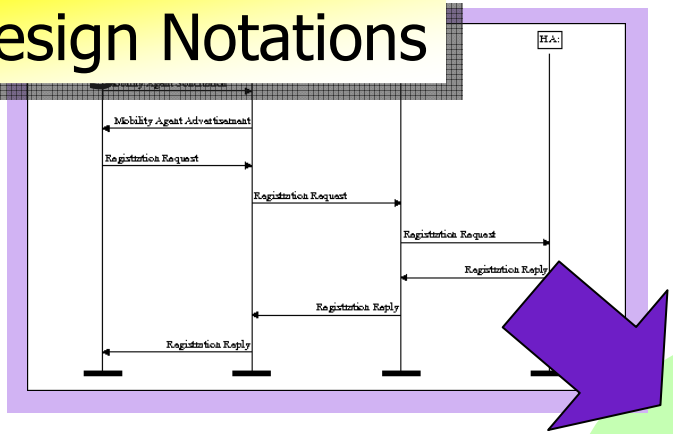
Leveraging Domain Knowledge

Lucent *Path Star* Telephone Switch

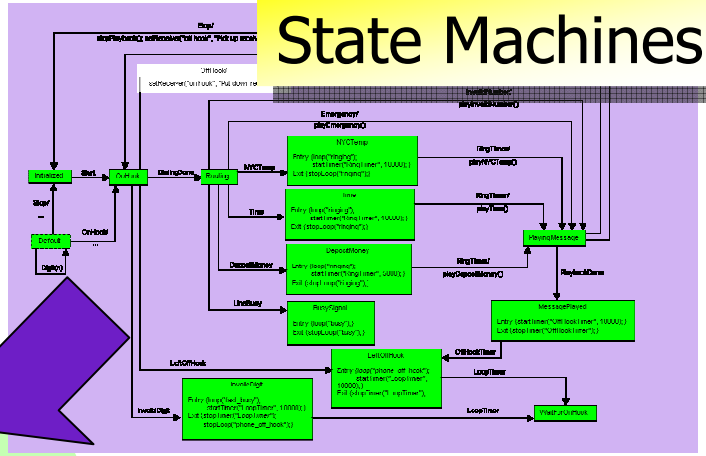
- Holzmann developed a customized model extraction from C to Spin for *telephone switching software*
- Translation using pattern matching of *particular domain idioms*
- In essence, an abstract machine for a particular domain
- Very effective at finding subtle defects

System Modeling Problem – Variety of System Descriptions

Design Notations

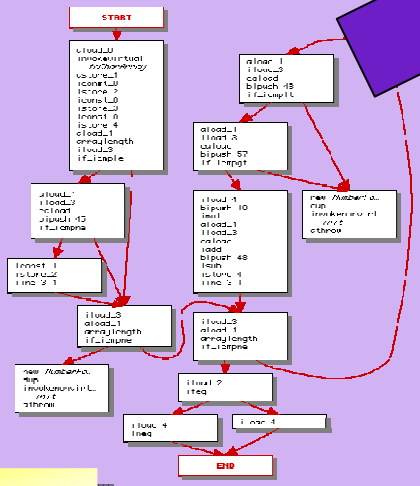


State Machines



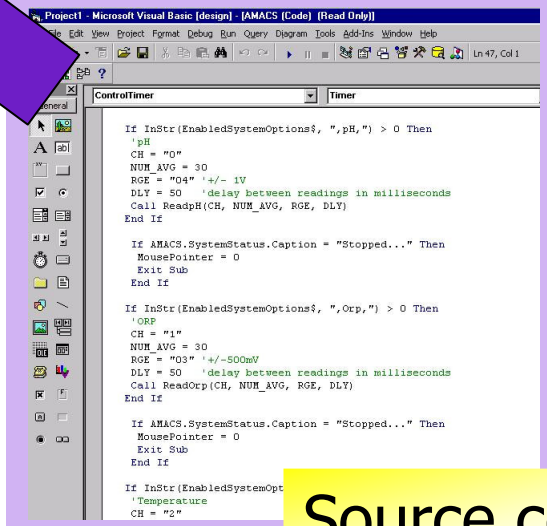
Model Checker

Byte code



Different levels of abstraction!

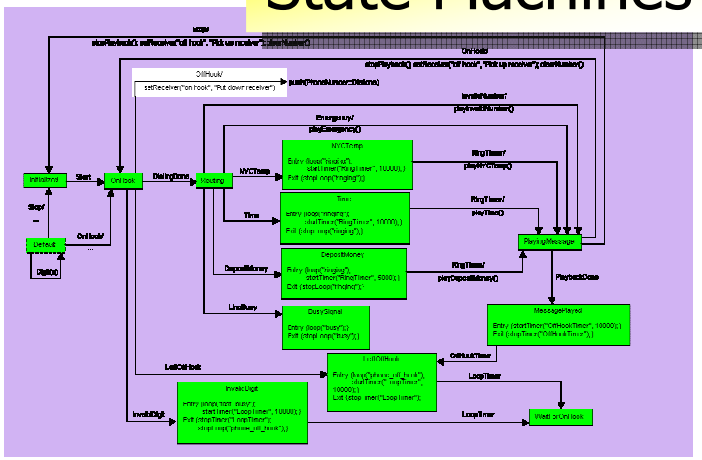
Source code



The Goal

Avionics

State Machines



Model-checking Engine

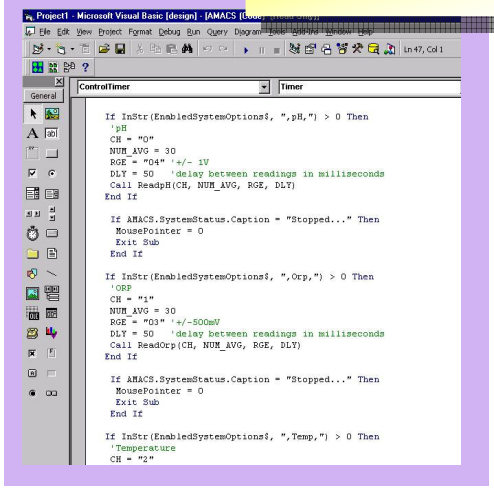
Domain & Abstraction Extensions

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

The Goal

Device Drivers

Source code



Model-checking Engine

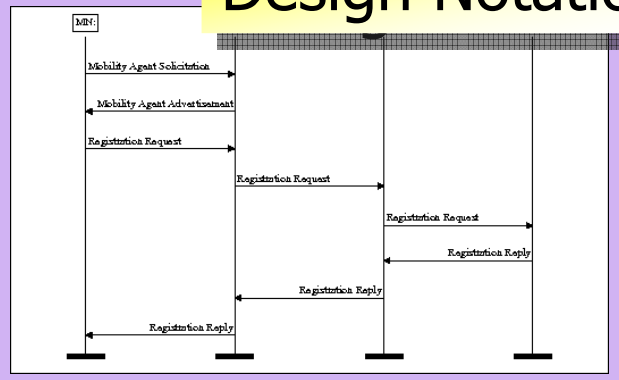
Domain & Abstraction Extensions

Abstract machine tailored to domain and level of abstraction

The Goal

Automotive

Design Notations



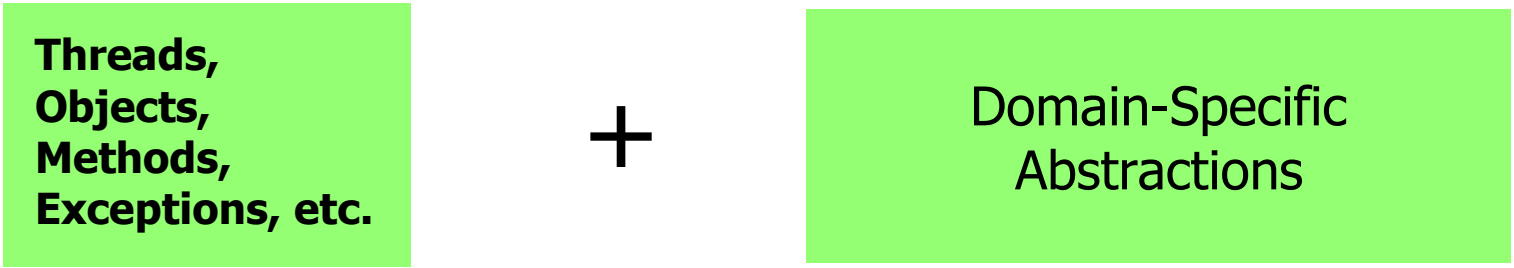
Model-checking Engine

Domain & Abstraction Extensions

Abstract machine tailored to domain and level of abstraction

Customization Mechanisms

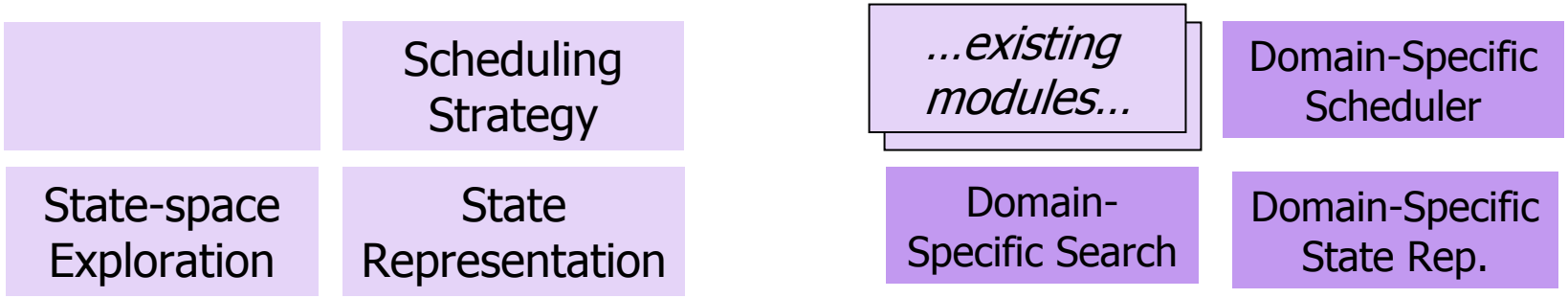
Bogor -- Extensible Modeling Language



Core Modeling Language

Add new commands & expressions

Bogor -- Customizable Checking Engine Modules



Core Checker Modules

Customized Checker Modules

Outline

--- A Flexible Framework for Creating Software Model Checkers

Bogor Language and UI

- Demo/Example: Dining philosophers
- Support for OO Features

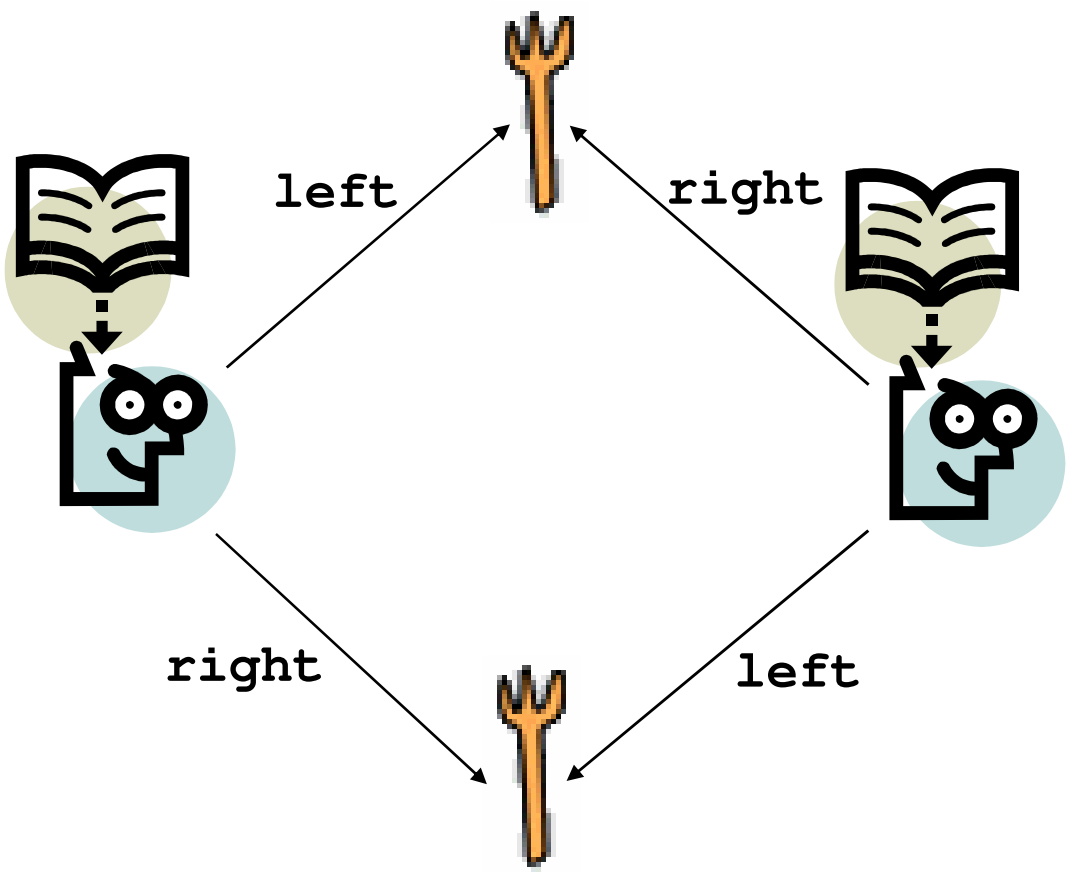
Customization Facilities

- Adding new commands & expressions to the modeling language
- Customizing core model checking algorithms

Example Customizations

- Component designs for avionics
- Java/C# Virtual Machines
  - synergistic integration with static analysis
- Symbolic Execution
  - synergistic integration with theorem proving & testing

An Example — 2 Dining Philosophers



How can we represent this system in BIR (the Bandera Intermediate Representation) – Bogor's modeling language?

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;
  }
```

Global data;
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;
References to Fork objects

```
  thread Phil(Fork left, Fork right) {
                                    fork
                                    {
                                    ;

    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;
  }
```

Dynamic allocation of forks

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;
  }
```

Dynamic allocation of threads; parameterized on forks

A BIR Example — 2 Dining Philosophers

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

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

    loc loc0:
      do {
        // create forks
        new Fork(false, fork1);
        new Fork(false, fork2);

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

Control locations

```
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;
  }
}
```

Execute these statement(s) atomically

High-level BIR

Motivation

- while low-level BIR is suitable as a target for automatic translation, it is too cumbersome to use for manual modeling
- High-level BIR provides constructs similar to most modern programming languages such as loops, try-catch, etc.
  - high-level BIR constructs are translated to low-level BIR before model checking starts

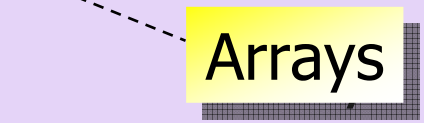
Bounded Buffer

```
record Object { lock l; }

record BoundedBuffer extends Object {
  Object[] buffer; int head; int tail; int bound;
}


function add(BoundedBuffer this, Object o) {
  lock (this.l);

  while this.tail == this.head do
    wait (this.l);
  end
  this.buffer[this.head] := o;
  this.head := (this.head + 1) % this.bound;
  notifyAll (this.l);

  unlock (this.l);
}
```

Inheritance

Bounded Buffer

```
record Object { lock l; }

record BoundedBuffer extends Object {
  Object[] buffer; int head; int tail; int bound;
}

function add(BoundedBuffer this, Object o) {
  lock (this.l);

  while this.tail == this.head do
    wait (this.l);
  end
  this.buffer[this.head] := o;
  this.head := (this.head + 1) % this.bound;
  notifyAll (this.l);

  unlock (this.l);
}
```



Bounded Buffer

```
record Object { lock l; }

record BoundedBuffer extends Object {
  Object[] buffer; int head; int tail; int bound;
}


function add(BoundedBuffer this, Object o) {
  lock (this.l);

  while this.tail == this.head do
    wait (this.l);
  end
  this.buffer[this.head] := o;
  this.head := (this.head + 1) % this.bound;
  notifyAll (this.l);

  unlock (this.l);
}
```

Locking primitives

Bounded Buffer

```
record Object { lock l; }

record BoundedBuffer extends Object {
  Object[] buffer; int head; int tail; int bound;
}


function add(BoundedBuffer this, Object o) {
  lock (this.l);

  while this.tail == this.head do
    wait (this.l);
  end
  this.buffer[this.head] := o;
  this.head := (this.head + 1) % this.bound;
  notifyAll (this.l);

  unlock (this.l);
}
```

Wait/notify

Bounded Buffer

```
record Object { lock l; }

record BoundedBuffer extends Object {
  Object[] buffer; int head; int tail; int bound;
}


function add(BoundedBuffer this, Object o) {
  lock (this.l);

  while this.tail == this.head do
    wait (this.l);
  end
  this.buffer[this.head] := o;
  this.head := (this.head + 1) % this.bound;
  notifyAll (this.l);

  unlock (this.l);
}
```

High-level control structures

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

Outline

--- A Flexible Framework for Creating Software Model Checkers

Bogor Language and UI

- Demo/Example: Dining philosophers
- Support for OO Features

Customization Facilities

- Adding new commands & expressions to the modeling language
- Customizing core model checking algorithms

Example Customizations

- Component designs for avionics
- Java/C# Virtual Machines
  - synergistic integration with static analysis
- Symbolic Execution
  - synergistic integration with theorem proving & testing

Customization Mechanisms

Bogor -- Extensible Modeling Language

Threads, Objects, Methods, Exceptions, etc.

+

Domain-Specific Abstractions

Core Modeling Language

Add new commands & expressions

Bogor -- Customizable Checking Engine Modules

	Scheduling Strategy	*...existing modules...*	Domain-Specific Scheduler
State-space Exploration	State Representation	Domain-Specific Search	Domain-Specific State Rep.

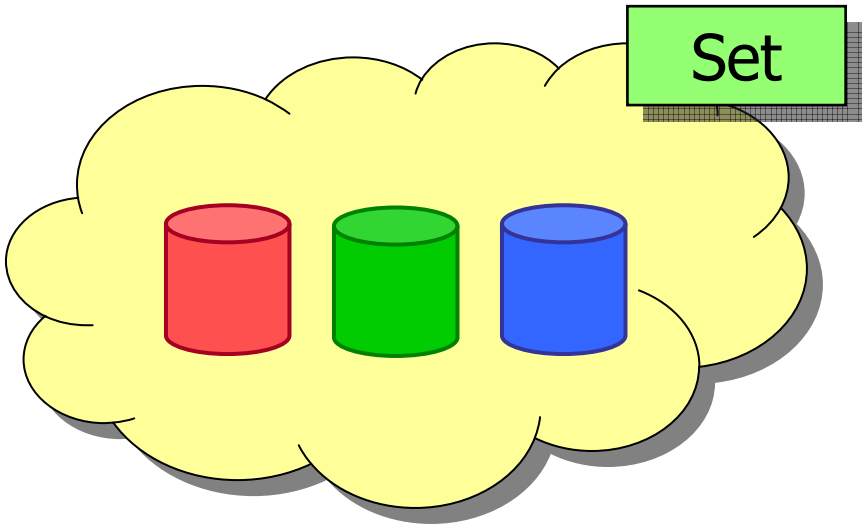
Core Checker Modules

Customized Checker Modules

Bogor Language Extensions

BIR does not include a *set* data type as a primitive.
Let's imagine adding *symmetric sets* as a BIR extension.

- add an element
- remove an element
- non-deterministically select an element
- universal quantification of predicate across all elements



Domain-Specific Modeling

Bogor -- Extensible Modeling Language

APIs generated automatically; just fill in semantics

Threads, Objects, Methods, Exceptions, etc.

+

Set.type<'a>

```
class SetValue implements IValue {
  byte[] linearize(…) {
    Bogor API calls…
  } …
}
```

create()

selectElement()

add()

⋮

forAll()

+

```
IValue selectElement(…) {
  Bogor API calls…
}
```

⋮

```
IValue forAll(…) {
  Bogor API calls…
}
```

No need to extend parser!

Core Modeling Language

New Bogor types and primitives

Java implementation of new values and new primitives inside model-checker

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

A new *type* to represent polymorphic symmetric sets

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef add<'a>(Set.type<'a>, 'a);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef add<'a>(Set.type<'a>, 'a);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

Variable arity function for creating symmetric sets

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef add<'a>(Set.type<'a>, 'a);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

Non-deterministically pick an element of the set to return

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef add<'a>(Set.type<'a>, 'a);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

Commands to add and remove elements

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

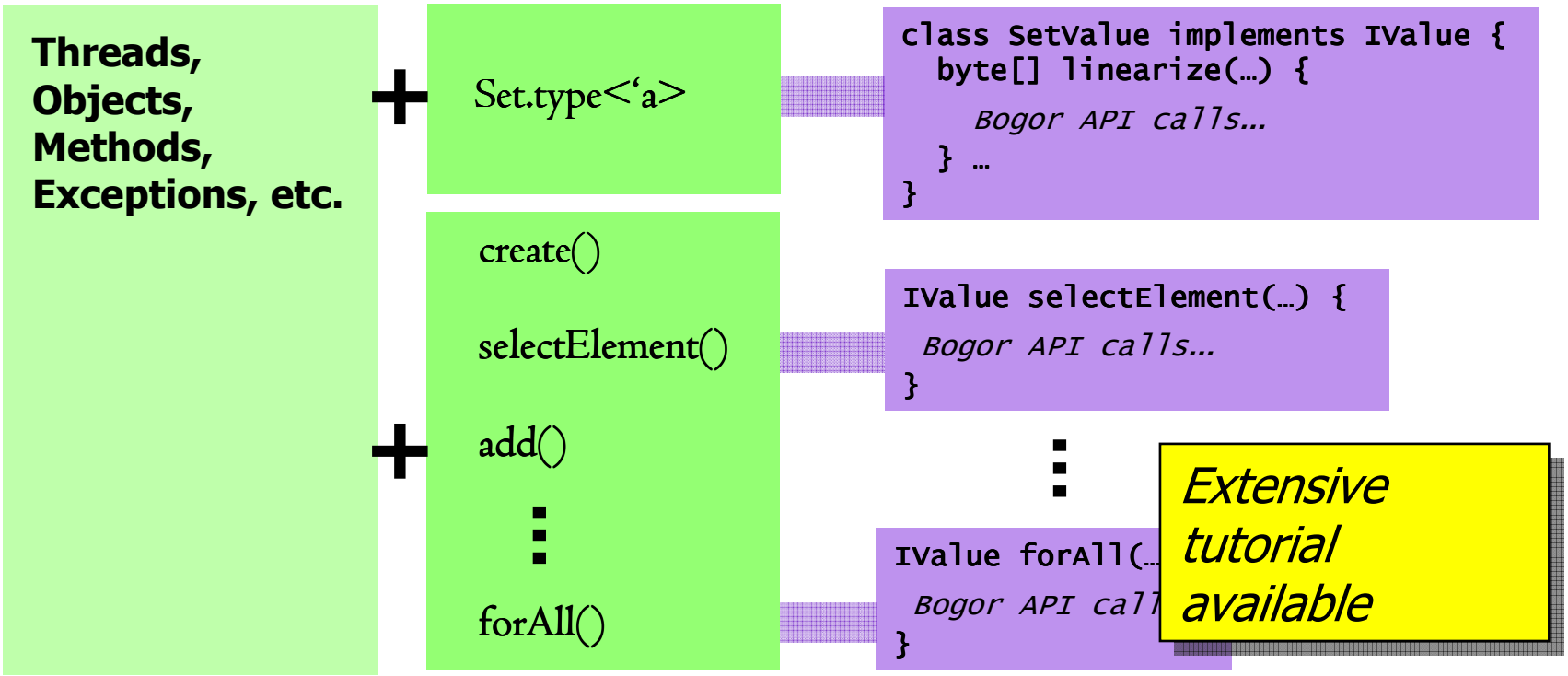
Higher-order function implements quantification over set elements

Predicate on set element

Set value to iterate over

Domain-Specific Modeling

Bogor -- Extensible Modeling Language



Core Modeling Language

New Bogor types and primitives

Java implementation of new values and new primitives inside model-checker

Customization Mechanisms

Bogor -- Extensible Modeling Language

Threads, Objects, Methods, Exceptions, etc.

+

Domain-Specific Abstractions

Core Modeling Language

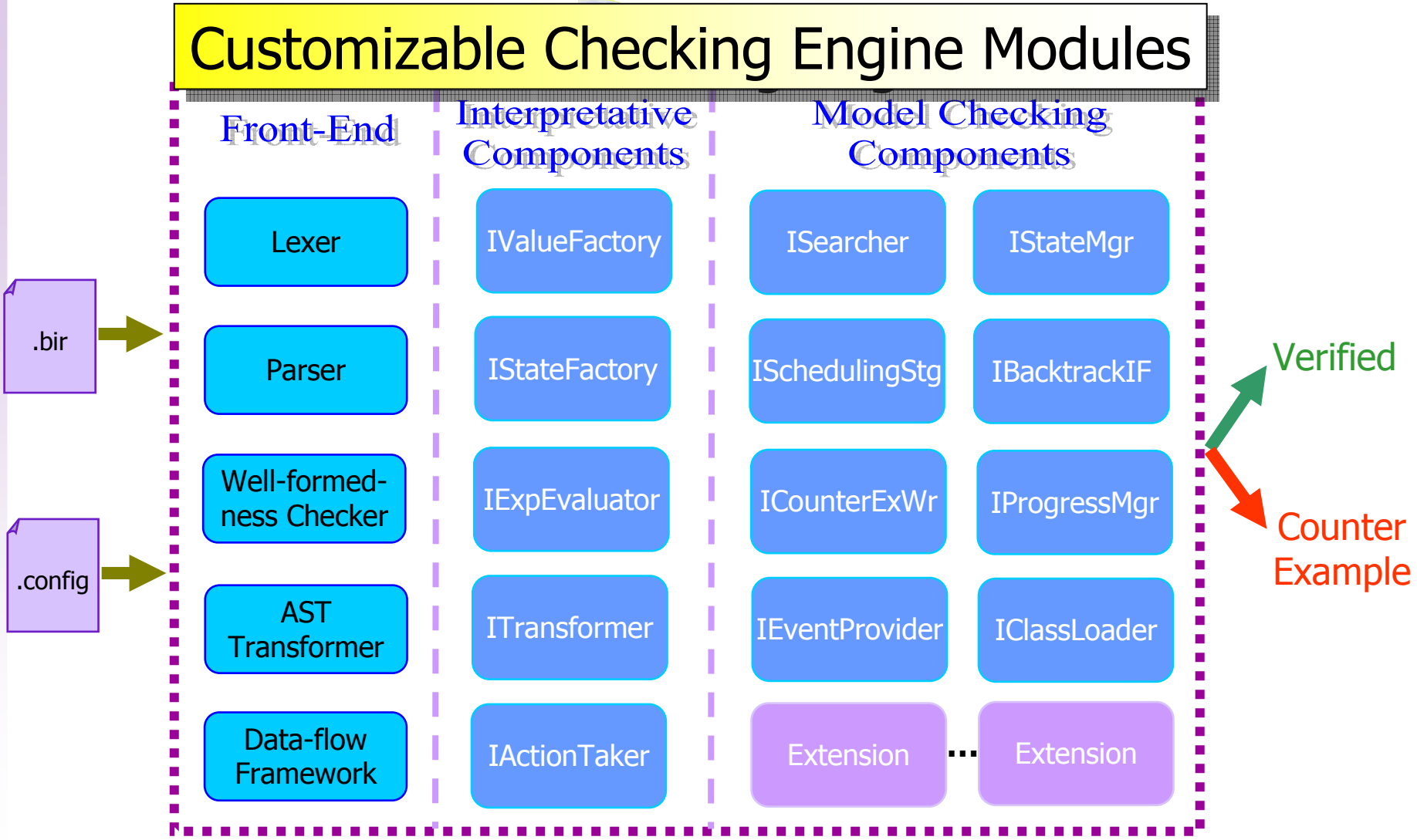
Bogor -- Customizable Checking Engine Modules

	Scheduling Strategy	*...existing modules...*	Domain-Specific Scheduler
State-space Exploration	State Representation	Domain-Specific Search	Domain-Specific State Rep.

Core Checker Modules

Customized Checker Modules

Internal Architecture



…modular components with clean and well-designed API using design patterns

Bogor Configuration

A Bogor configuration is a key-value set

Keys for component interfaces

Java class implementation for each interface

```
IActionTaker                = DefaultActionTaker
IExpEvaluator               = DefaultExpEvaluator
ISchedulingStrategist       = DefaultSchedulingStrategist
ISearcher                   = DefaultSearcher
IStateManager               = DefaultStateManager
ITransformer                = DefaultTransformer
IBacktrackingInfoFactory    = DefaultBacktrackingInfoFactory
IStateFactory               = DefaultStateFactory
IValueFactory               = DefaultValueFactory

ISearcher.maxErrors         = 1
…
```

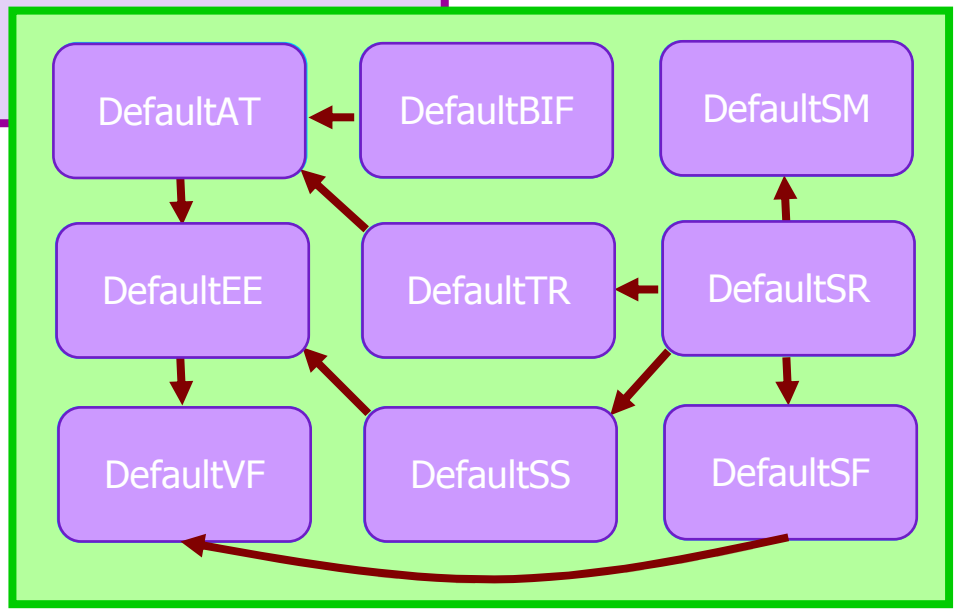
Options for components

Bogor Initialization

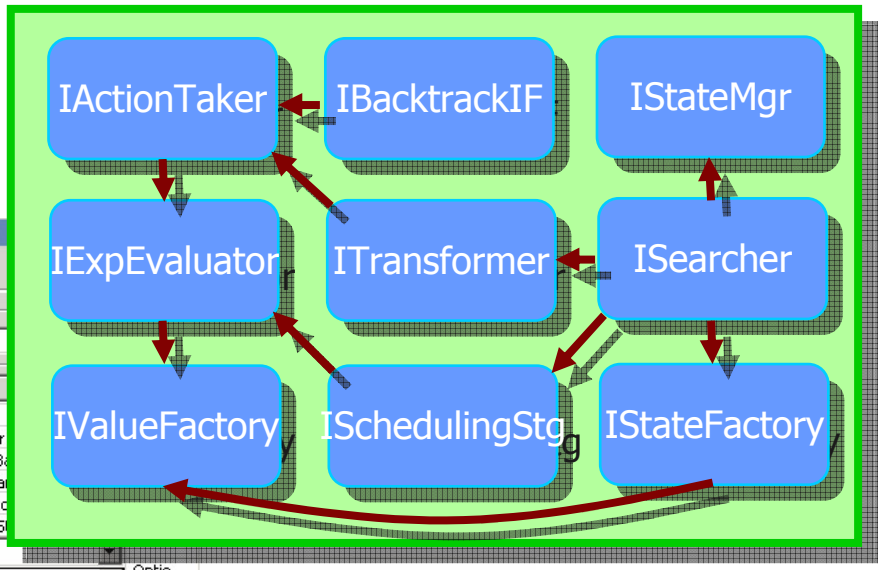
```
IActionTaker              = DefaultActionTaker
IExpEvaluator             = DefaultExpEvaluator
ISchedulingStrategist     = DefaultSchedulingStrategist
ISearcher                 = DefaultSearcher
IStateManager             = DefaultStateManager
ITransformer              = DefaultTransformer
IBacktrackingInfoFactory  = DefaultBacktrackingInfoFactory
IStateFactory             = DefaultStateFactory
IValueFactory             = DefaultValueFactory

ISearcher.maxErrors       = 1
…
```

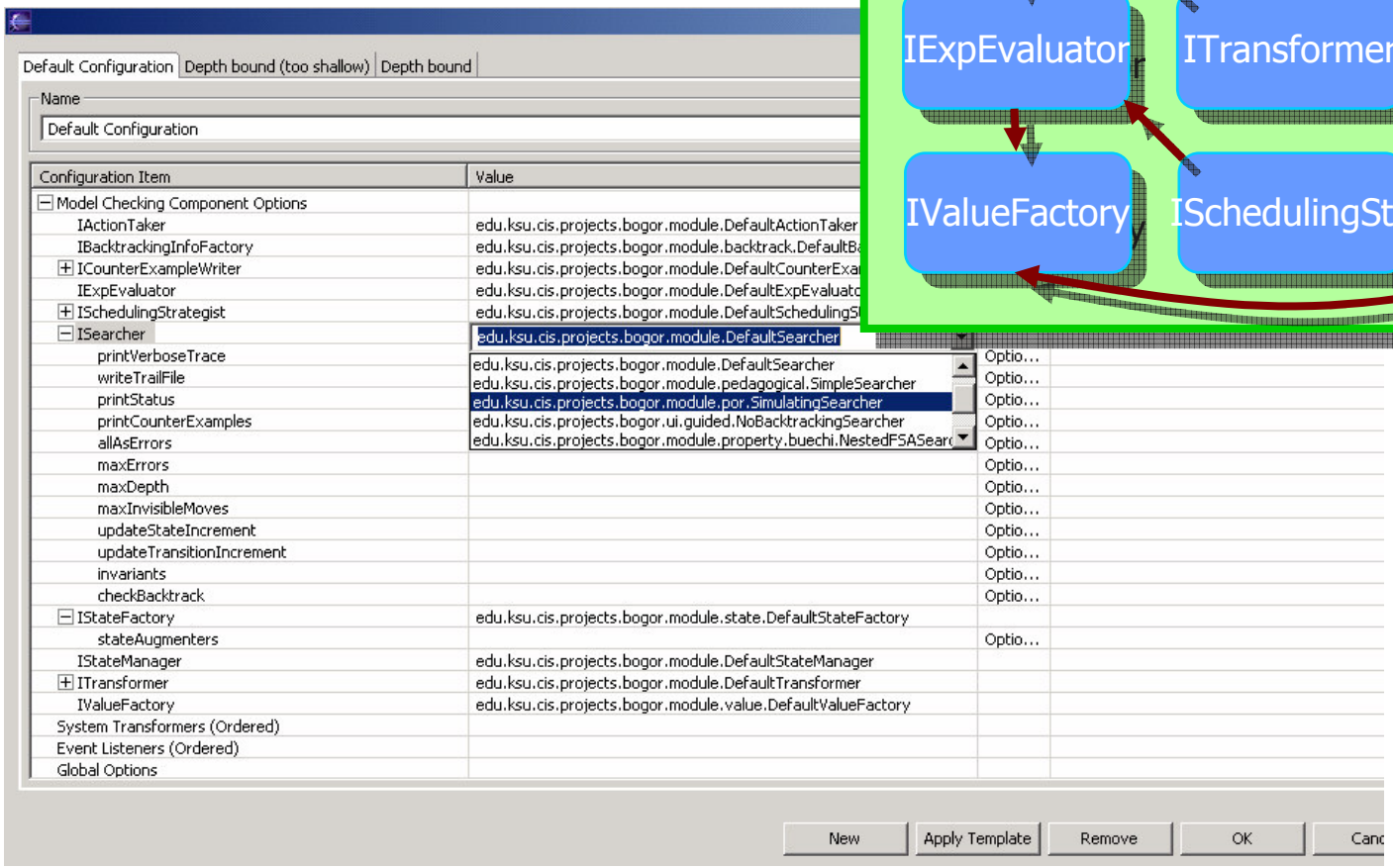
Given a configuration, Bogor instantiate the specified components

Options are passed to each component, and connections are established



Demo


IActionTaker
IBacktrackIF
IStateMgr
IExpEvaluator
ITransformer
ISearcher
IValueFactory
ISchedulingStg
IStateFactory


Default Configuration
Depth bound (too shallow)
Depth bound
Name
Default Configuration
Configuration Item
Value
Model Checking Component Options
IActionTaker
edu.ksu.cis.projects.bogor.module.DefaultActionTaker
IBacktrackingInfoFactory
ICounterExampleWriter
IExpEvaluator
ISchedulingStrategist
ISearcher
edu.ksu.cis.projects.bogor.module.DefaultSearcher
printVerboseTrace
writeTrailFile
printStatus
printCounterExamples
allAsErrors
maxErrors
maxDepth
maxInvisibleMoves
updateStateIncrement
updateTransitionIncrement
invariants
checkBacktrack
IStateFactory
edu.ksu.cis.projects.bogor.module.state.DefaultStateFactory
stateAugmenters
IStateManager
edu.ksu.cis.projects.bogor.module.DefaultStateManager
ITransformer
edu.ksu.cis.projects.bogor.module.DefaultTransformer
IValueFactory
edu.ksu.cis.projects.bogor.module.value.DefaultValueFactory
System Transformers (Ordered)
Event Listeners (Ordered)
Global Options
edu.ksu.cis.projects.bogor.module.DefaultSearcher
edu.ksu.cis.projects.bogor.module.pedagogical.SimpleSearcher
edu.ksu.cis.projects.bogor.module.por.SimulatingSearcher
edu.ksu.cis.projects.bogor.ui.guided.NoBacktrackingSearcher
edu.ksu.cis.projects.bogor.module.property.buechi.NestedFSASearc
Optio...
New
Apply Template
Remove
OK

Outline

--- A Flexible Framework for Creating Software Model Checkers

Bogor Language and UI

- Demo/Example: Dining philosophers
- Support for OO Features

Customization Facilities

- Adding new commands & expressions to the modeling language
- Customizing core model checking algorithms

Example Customizations

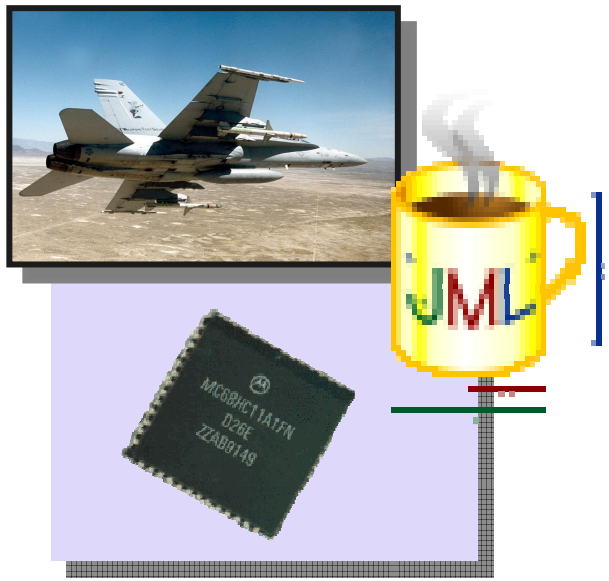
- Component designs for avionics
- Java/C# Virtual Machines
  - synergistic integration with static analysis
- Symbolic Execution
  - synergistic integration with theorem proving & testing

Domain Customizations

We are aware of more than 31 substantive extensions to Bogor that have been built by 19 people, only one of whom was the primary Bogor developer.

Examples

- Component-based avionics systems in *Cadena*
- Checking Java Modeling Language (JML) specifications
- Compositional Reasoning using Symbolic Execution
- Test Case Generation
- Multi-agent Systems
- Modeling MPI (U Mass)
- Model checking Motorola M68HC11 machine code (BYU)



Bogor Extension Examples

See paper!

Property Checking Modules

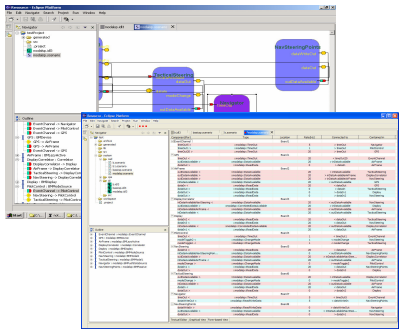
- Invariants
- Regular expression/finite-state automata (1083)
- An automata-theoretic Linear Temporal Logic (1011) checker
- Computation-tree Logic (1418) checker based on alternating tree automata
- We have also implemented a checker extension for the Java Modeling Language (3721).

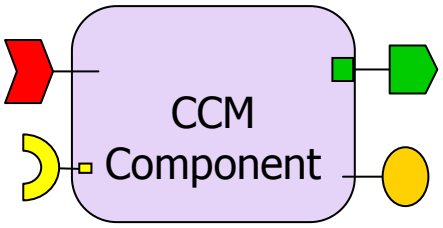
Avionics Mission Control Systems

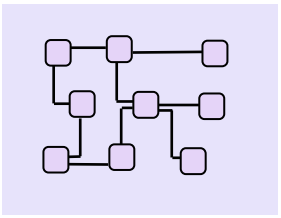
- Mission-control software for Boeing military aircraft
- Boeing's Bold Stroke Avionics Middleware
  - ...built on top of ACE/TAO RT CORBA

CADENA

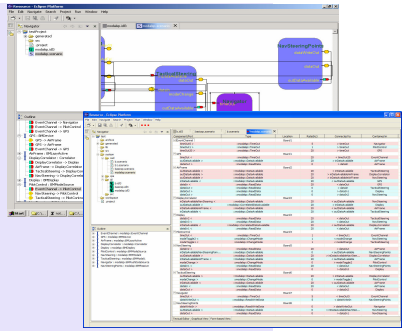
Development and analysis environment for systems built using CORBA Component Model

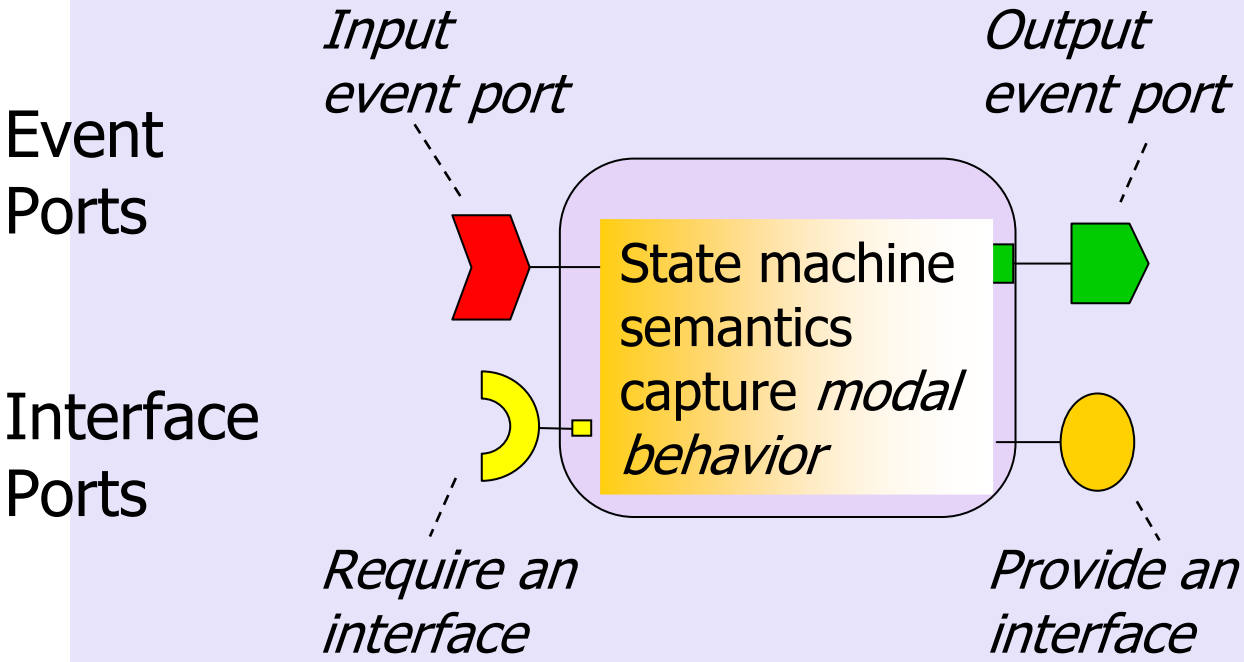






Component-based Design

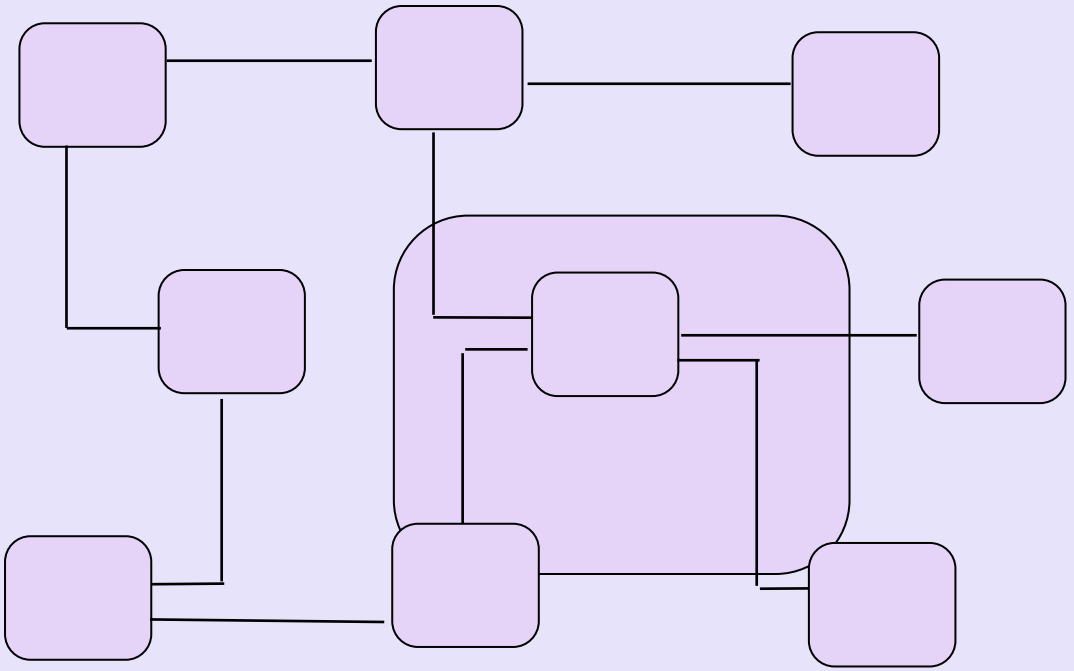


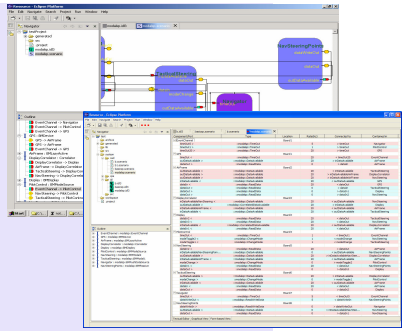


Component Development

Cadena development environment allows model-based development of Bold Stroke applications using the CORBA Component Model (CCM)

Component-based Design



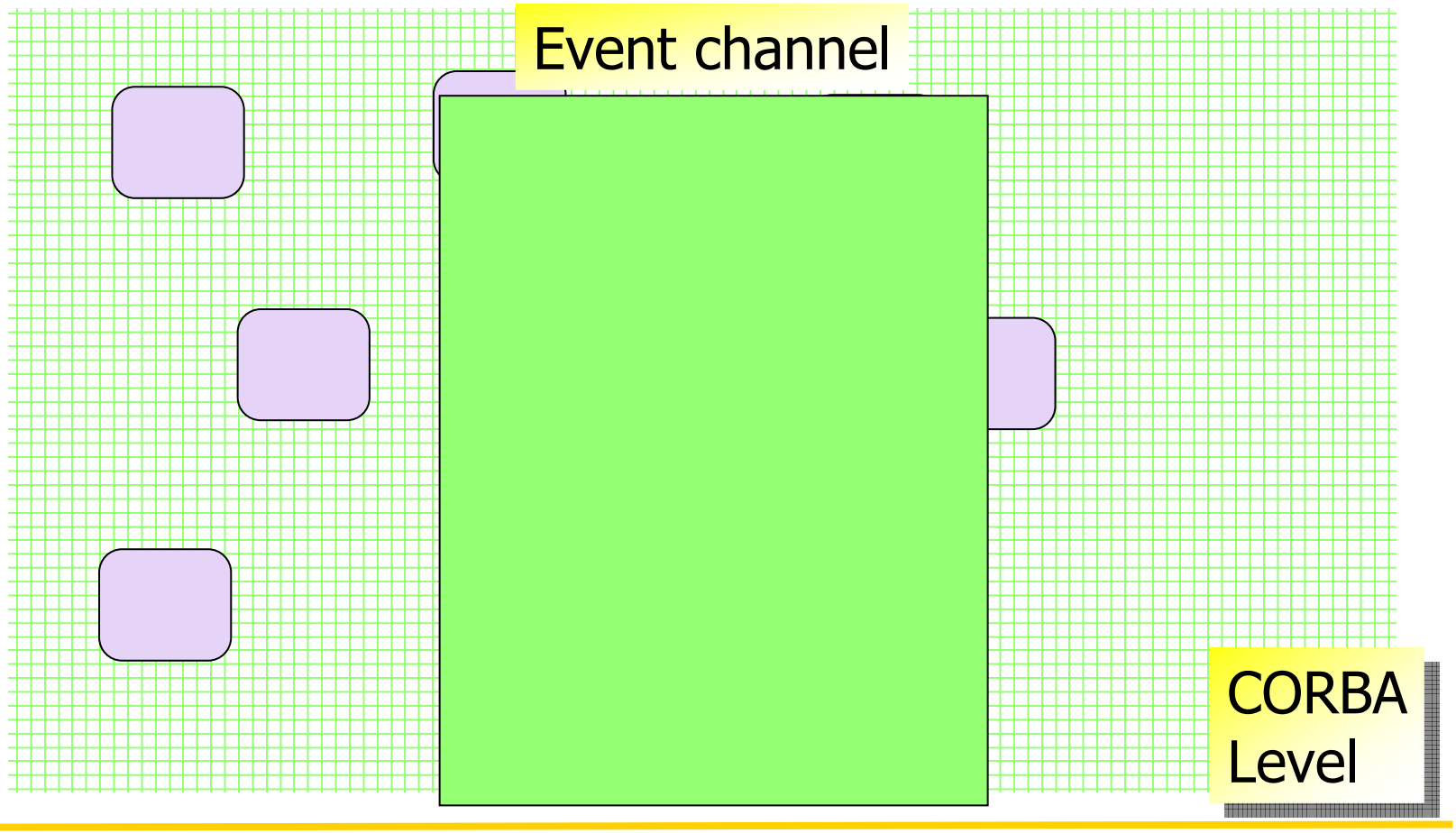


…100's of components

Must take into account the semantics of the underlying RT middleware!

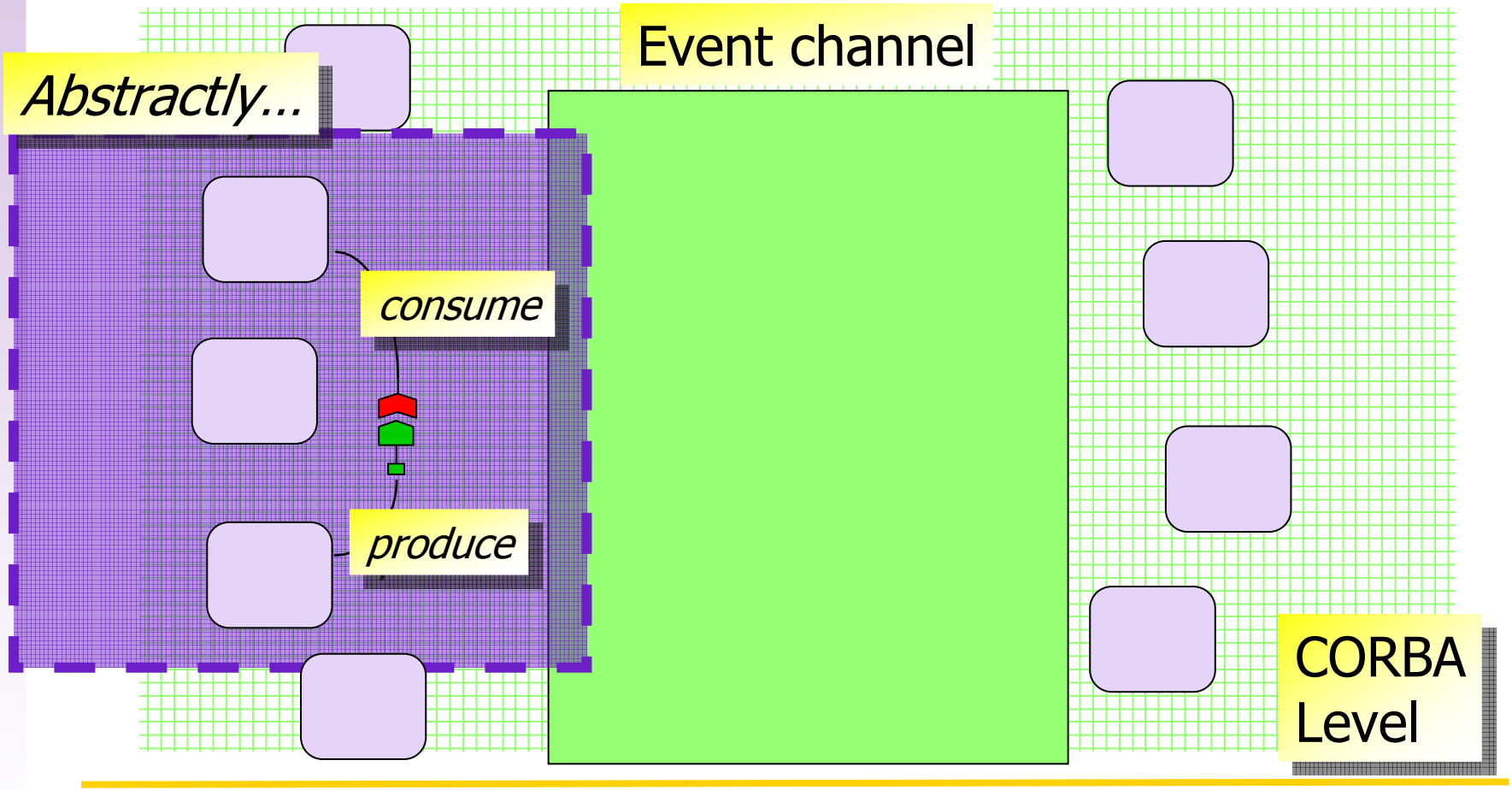
How can model checking be used to use about global modal control flow properties of the system, e.g., if component A is in mode M and sends event E, component B will eventually transition to mode N?

RT Middleware-based Implementation



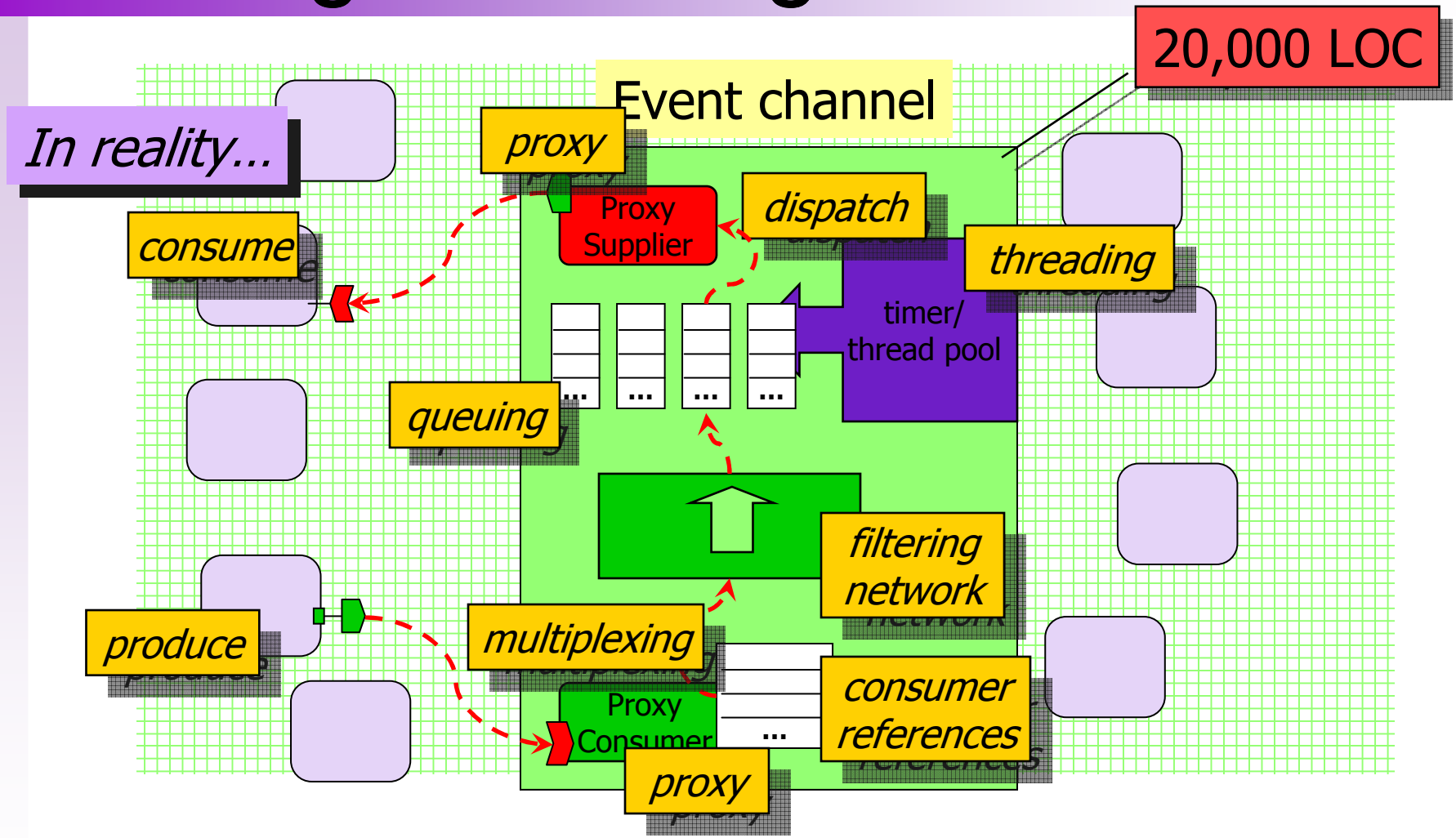
Real Time Event Channel
(from ACE/TAO RT CORBA)

RT Middleware-based Implementation



Real Time Event Channel
(from ACE/TAO RT CORBA)

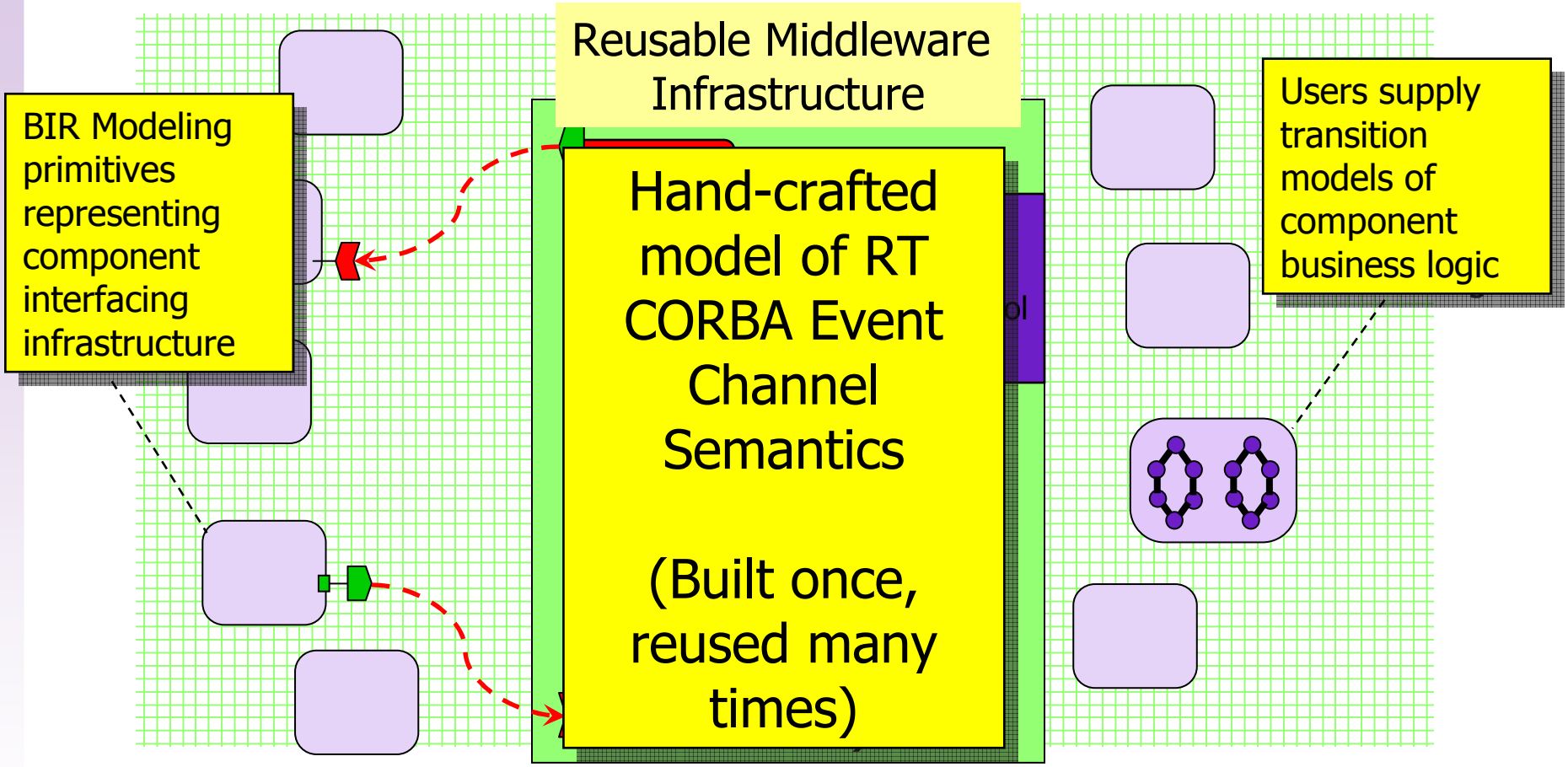
Boeing Threading Model



Periodic event driven system based on rate-monotonic scheduling

Domain-Specific Modeling Environment

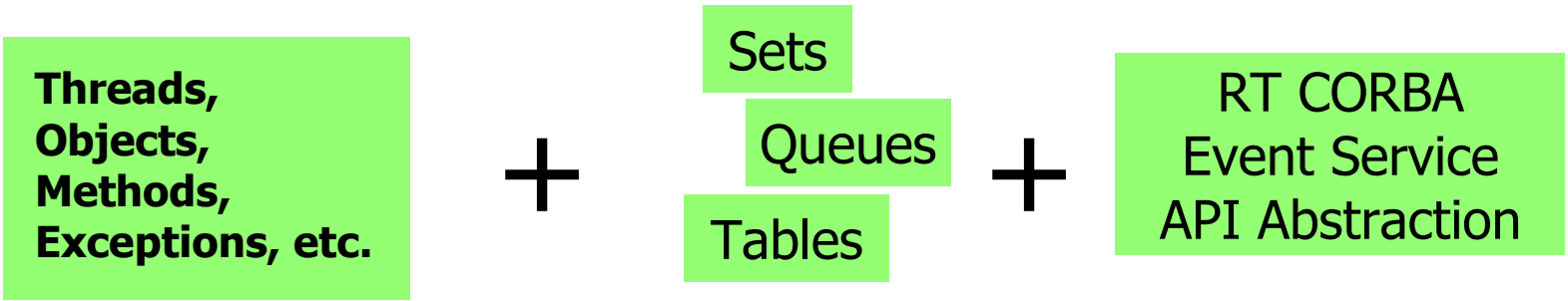
Theme



- Systems in the product line are built by deploying components on top of reuseable middleware infrastructure

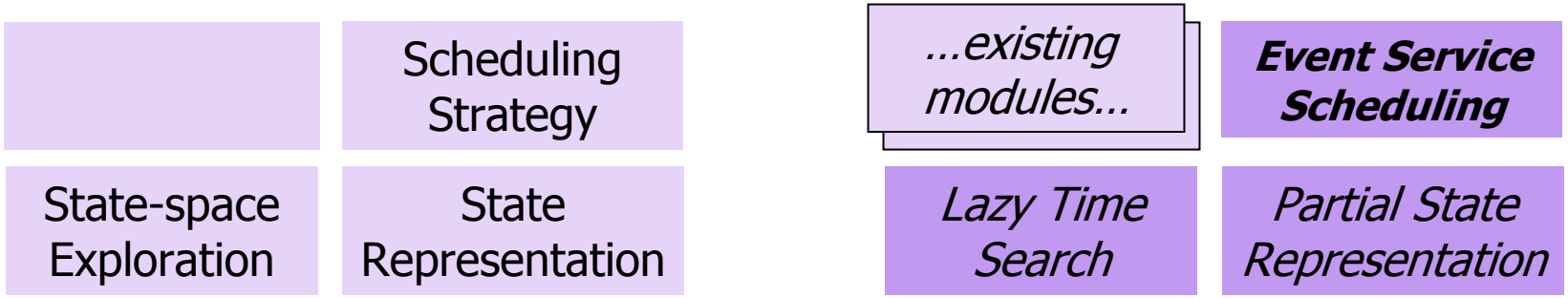
Bogor Customized To Cadena

Bogor -- Extensible Modeling Language



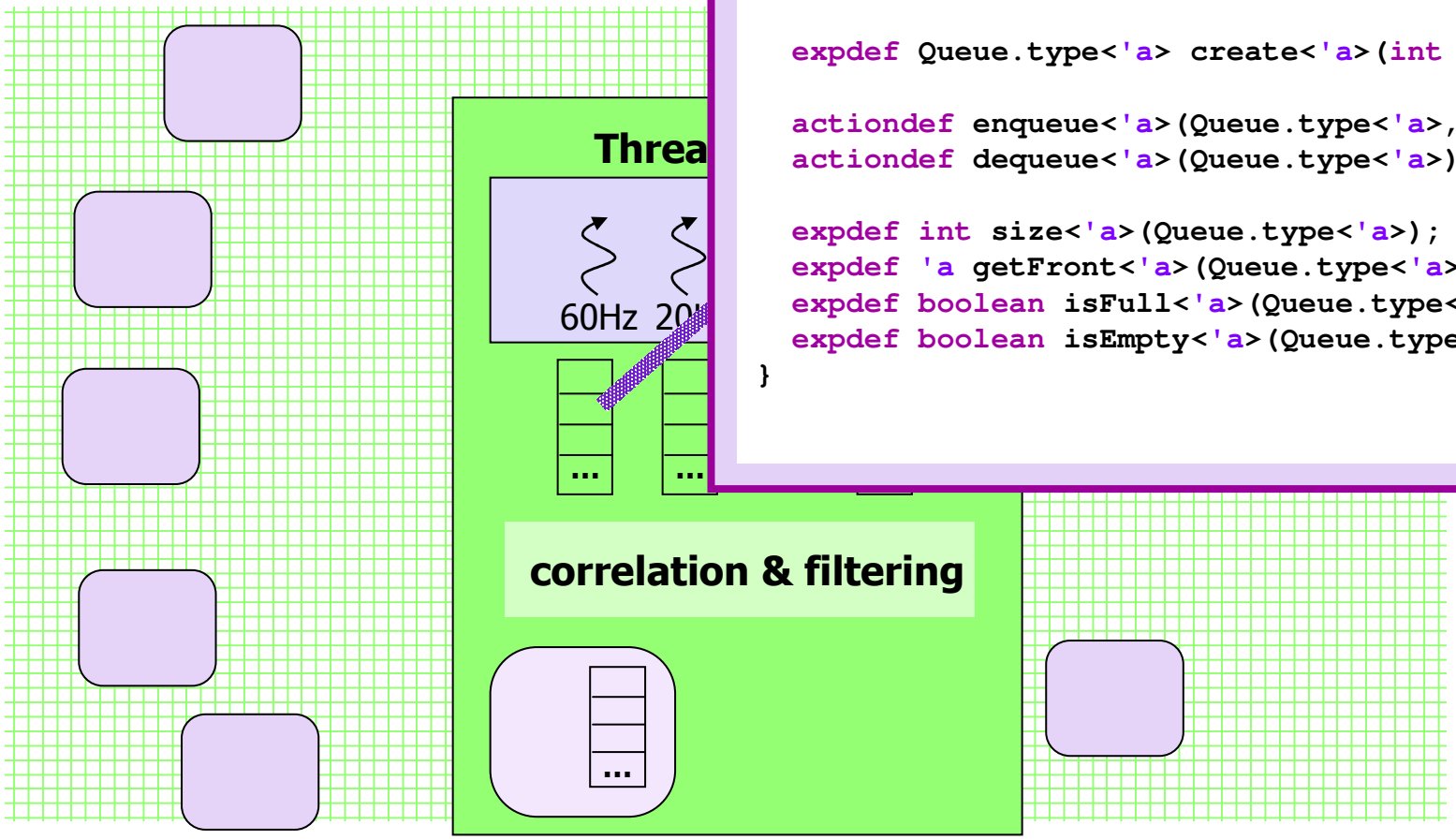
Core Modeling Language — Extensions — Domain-specific Abstractions

Bogor -- Customizable Checking Engine Modules



Core Checker Modules — Customized Checker Modules

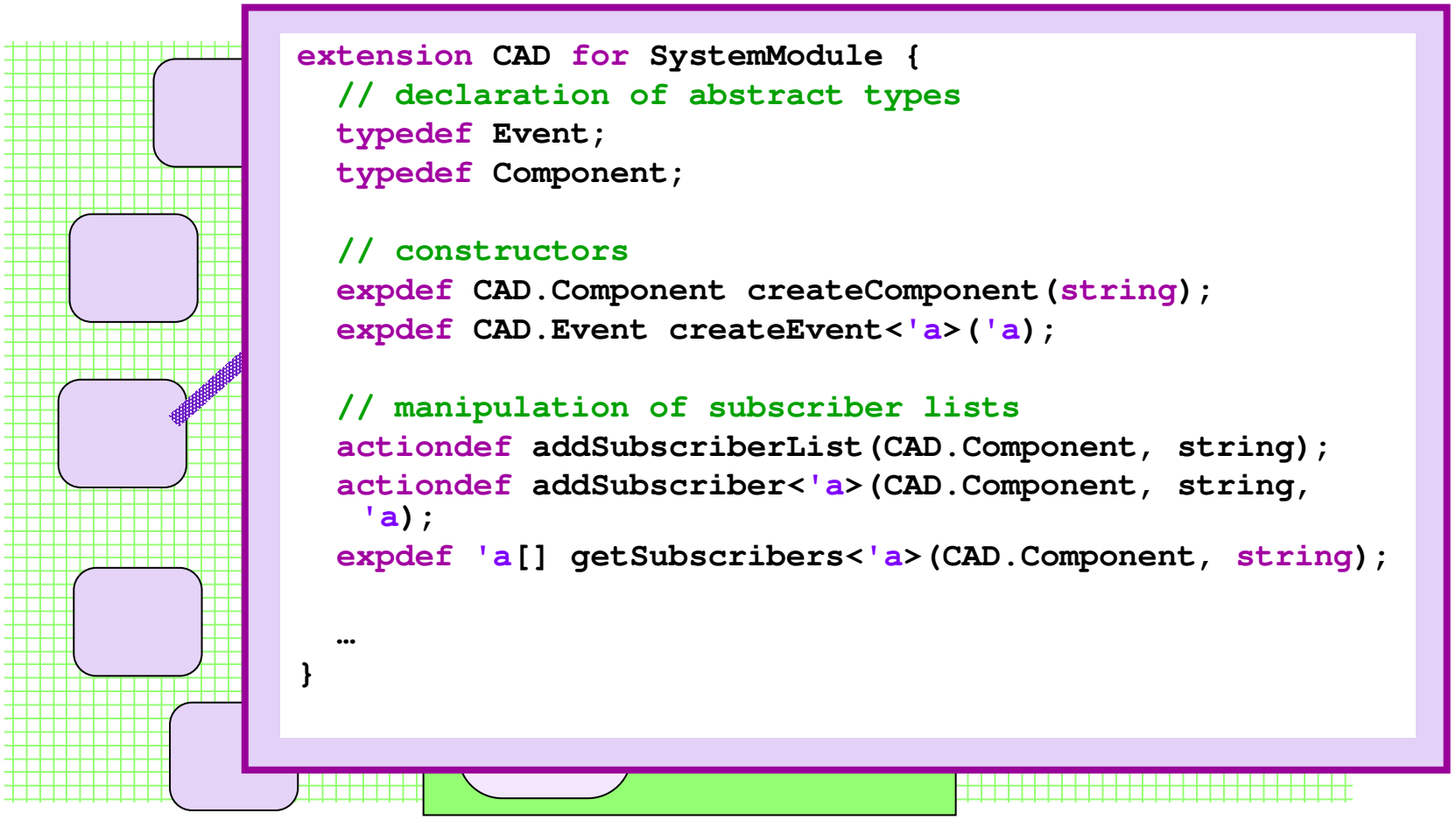
Bogor Modeling Extensions




```
extension Queue for cadena.Queue {
  typedef type<'a>;

  expdef Queue.type<'a> create<'a>(int size);

  actiondef enqueue<'a>(Queue.type<'a>, `a);
  actiondef dequeue<'a>(Queue.type<'a>);

  expdef int size<'a>(Queue.type<'a>);
  expdef 'a getFront<'a>(Queue.type<'a>);
  expdef boolean isFull<'a>(Queue.type<'a>);
  expdef boolean isEmpty<'a>(Queue.type<'a>);
}
```

Bogor extensions for representing event-channel queue data structures

Bogor Modeling Extensions

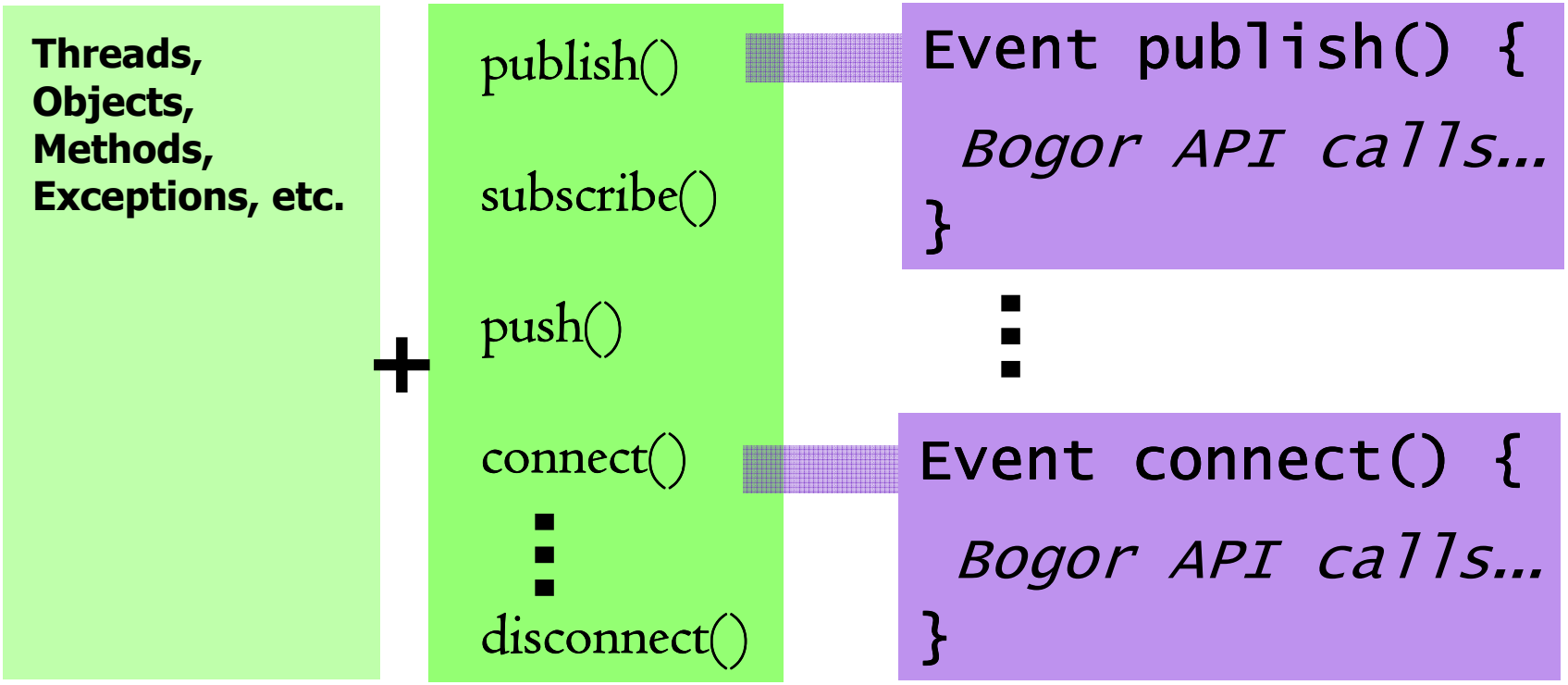


```
extension CAD for SystemModule {
  // declaration of abstract types
  typedef Event;
  typedef Component;

  // constructors
  expdef CAD.Component createComponent(string);
  expdef CAD.Event createEvent<'a>('a);

  // manipulation of subscriber lists
  actiondef addSubscriberList(CAD.Component, string);
  actiondef addSubscriber<'a>(CAD.Component, string,
    'a);
  expdef 'a[] getSubscribers<'a>(CAD.Component, string);

  …
}
```

Bogor extensions for representing CCM component API

Domain-Specific Modeling

API of RT-Event Channel incorporated as BIR primitives



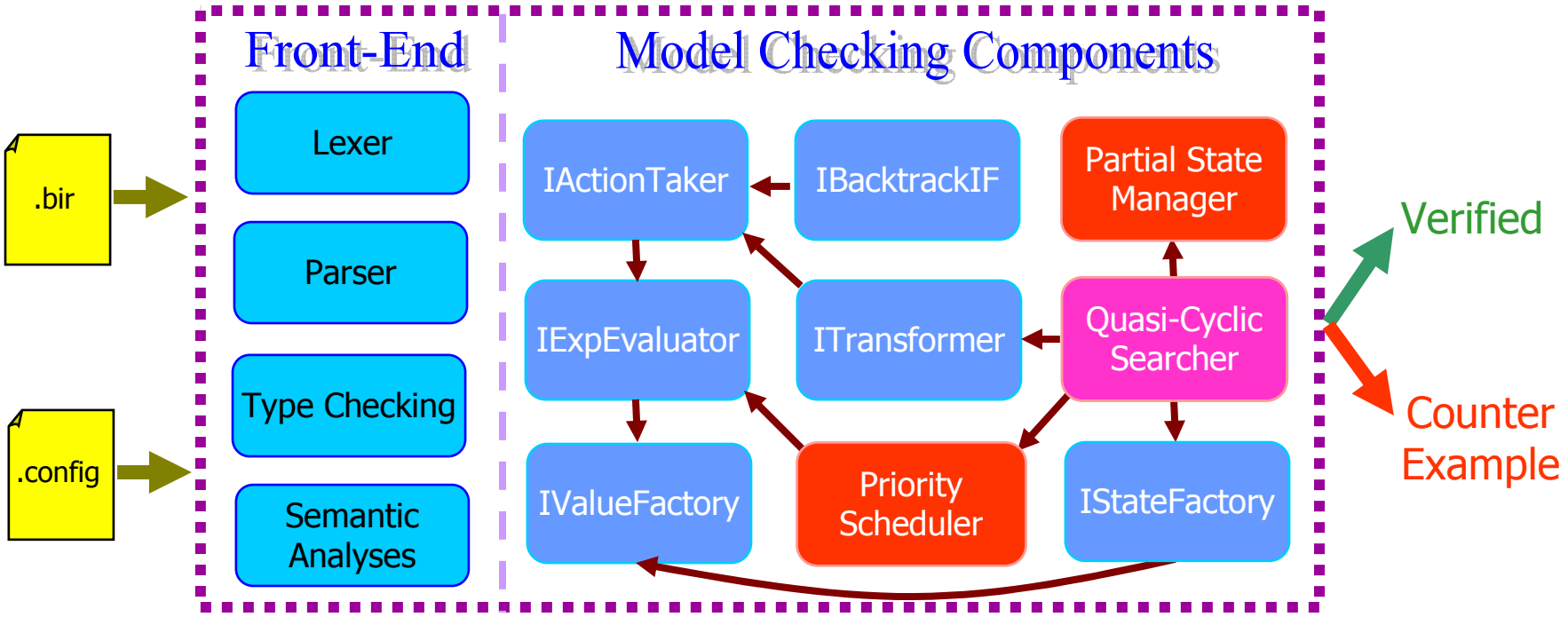
Core Modeling Language

New Bogor primitives corresponding to Event Channel API

Java implementation of new primitives inside model-checker

Domain-Specific Algorithms

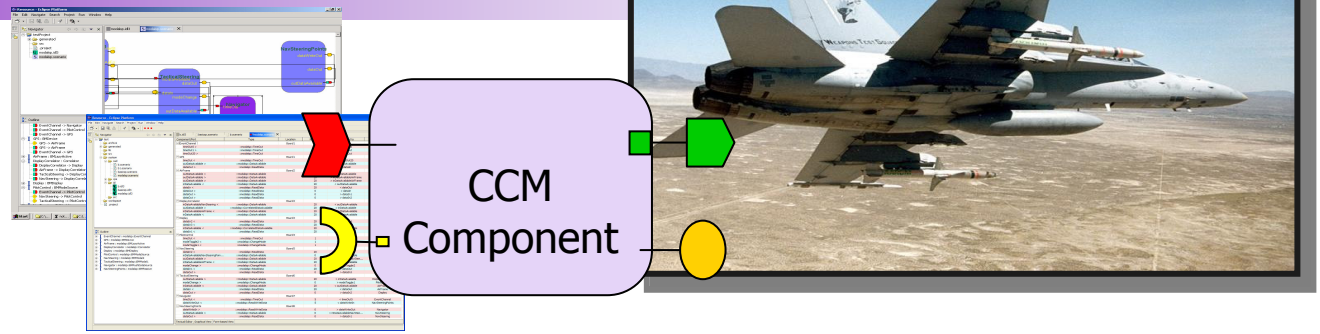
Bogor -- Customizable Checking Engine Modules



Bogor default modules are *unplugged* and *replaced* with state representation, scheduling and search strategies *customized to the Bold Stroke domain*

Assessment

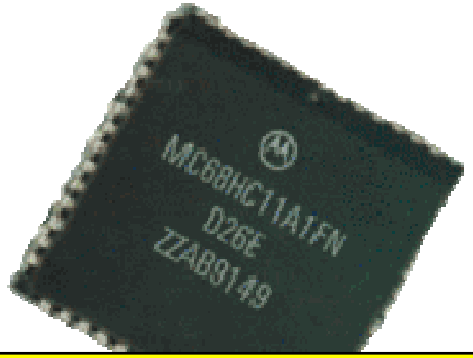
Lessons learned…



- By customizing the modeling language
  - built-in primitives for RT-CORBA communication services
  - built-in primitives for component infrastructure
- …models are must less tedious to create
- By tailoring scheduling and state-storage strategies to domain
  - we were able to gain orders of magnitude improvement in checking performance over previous SPIN-based implementation
  - we developed a parallel version of the state-space exploration strategy that gave additional improvements

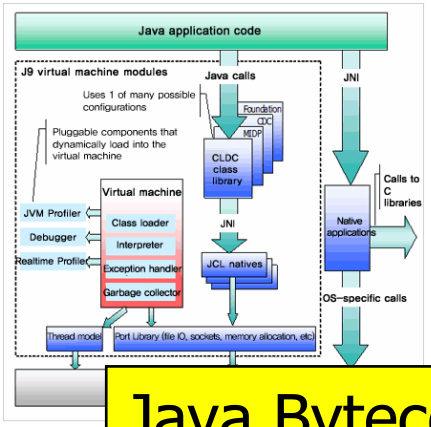
…see papers from EMSOFT 03, ISOLA 04

Model Checkers for Machine Instruction Sets

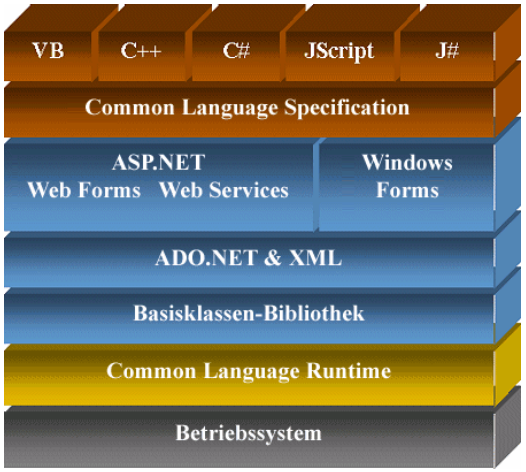


Motorola M68HC11

- Add new Bogor commands corresponding to instruction sets of virtual machines or hardware processors



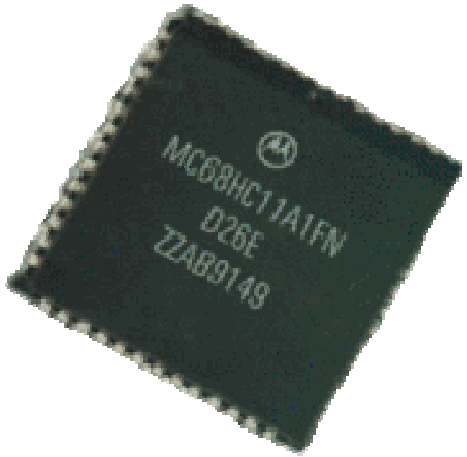
Java Bytecode



Microsoft MSIL .NET Bytecode

Simple Microprocessor

- Motorola M68HC11
- Flat 64K memory model
- Accumulators, memory bus, program counter, call stack
- Undergraduate project at Brigham Young University



Extension Declarations

```
typealias byte int wrap(0, 0xff);
typealias word int wrap(0, 0xffff);

extension m68hc11 for edu.byu.cs.bogor.m68hc11.M68hc11Module {
  // declaration of processor type
  typedef type;

  // constructor
  expdef m68hc11.type create();

  // accumulators
  expdef word getA(m68hc11.type);
  actiondef   setA(m68hc11.type, word value);
  …
  // control
  expdef word getSP(m68hc11.type);
  actiondef   setSP(m68hc11.type, word newSP);
  expdef byte getPC(m68hc11.type);
  actiondef   setPC(m68hc11.type, word newPC);
  …
  // memory bus
  expdef byte getByteMem(m68hc11.type, word address);
  actiondef   setByteMem(m68hc11.type, word address, byte value);
  expdef word getWordMem(m68hc11.type, word address);
}
```

Extension Declarations

```
typealias byte int wrap(0, 0xff);
typealias word int wrap(0, 0xffff);

extension m68hc11 for edu.byu.cs.bogor.m68hc11.M68hc11Module {
  // declaration of processor type
  typedef type;

  // constructor
  expdef m68hc11.type create();

  // accumulators
  expdef word getA(m68hc11.type);
  actiondef   setA(m68hc11.type, word value);
  …
  // control
  expdef word getSP(m68hc11.type);
  actiondef   setSP(m68hc11.type, word newSP);
  expdef byte getPC(m68hc11.type);
  actiondef   setPC(m68hc11.type, word newPC);
  …
  // memory bus
  expdef byte getByteMem(m68hc11.type, word address);
  actiondef   setByteMem(m68hc11.type, word address, byte value);
  expdef word getWordMem(m68hc11.type, word address);
}
```

Raw Integral Types

CPU Datatype

Instantiate CPU

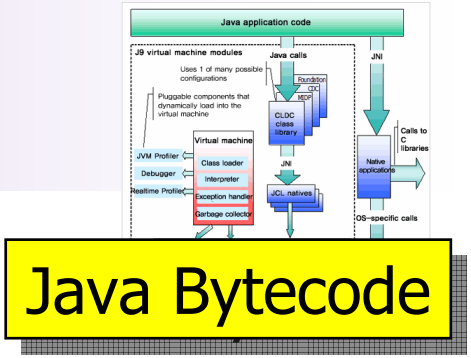
Register operators

Call stack, Program counter

Memory controller

Customization Mechanisms

Bogor -- Extensible Modeling Language



Java Bytecode

Threads, Objects, Methods, Exceptions, etc.

+

Java Virtual Machine (JVM) Abstractions

Core Modeling Language

Bogor -- Customizable Checking Engine Modules

	Scheduling Strategy
State-space Exploration	State Representation

Core Checker Modules

...existing modules...	Optimized JVM Scheduler
JVM-Specific Search	JVM State Rep.

Customized Checker Modules

VM Instruction Examples

Modeling language extensions

```
function {|Process1.run()V|}(transient (|Process1|) this, VM.F f) {
  …
  loc l0$27: do {
               VM.max(f, 3, 2); VM.set<(|Process1|)>(f, "this", 0);
               VM.fld(f, Op.GETSTATIC, "/|Deadlock.state|\\");
             } goto l1;

  loc l1:    do independent { VM.zro(f, Op.ICONST_1); }
             goto l2;

  loc l2:    do independent { VM.zro(f, Op.IADD); }
             goto l3$27;

  loc l3$27: do { VM.fld(f, Op.PUTSTATIC, "/|Deadlock.state|\\"); }
             goto l4$28;

  loc l4$28: do { VM.fld(f, Op.GETSTATIC, "/|Deadlock.lock1|\\"); }
             goto l5;

  loc l5:    do independent { VM.zro(f, Op.DUP); }
             goto l6;

  loc l6:    do independent { VM.lcl(f, Op.ASTORE, 1); }
             goto l7;
  …
}
```

Bogor Extension Examples

See paper!

Core algorithm customizations

- Partial Order Reductions
  - Sleep sets (298)
  - Conditional stubborn sets (618), and
  - Ample sets (306)
- Independence Relations for OO Software
  - read-only data (515)
  - patterns of locking (73)
  - patterns of object ownership (69)
  - escape information (216)

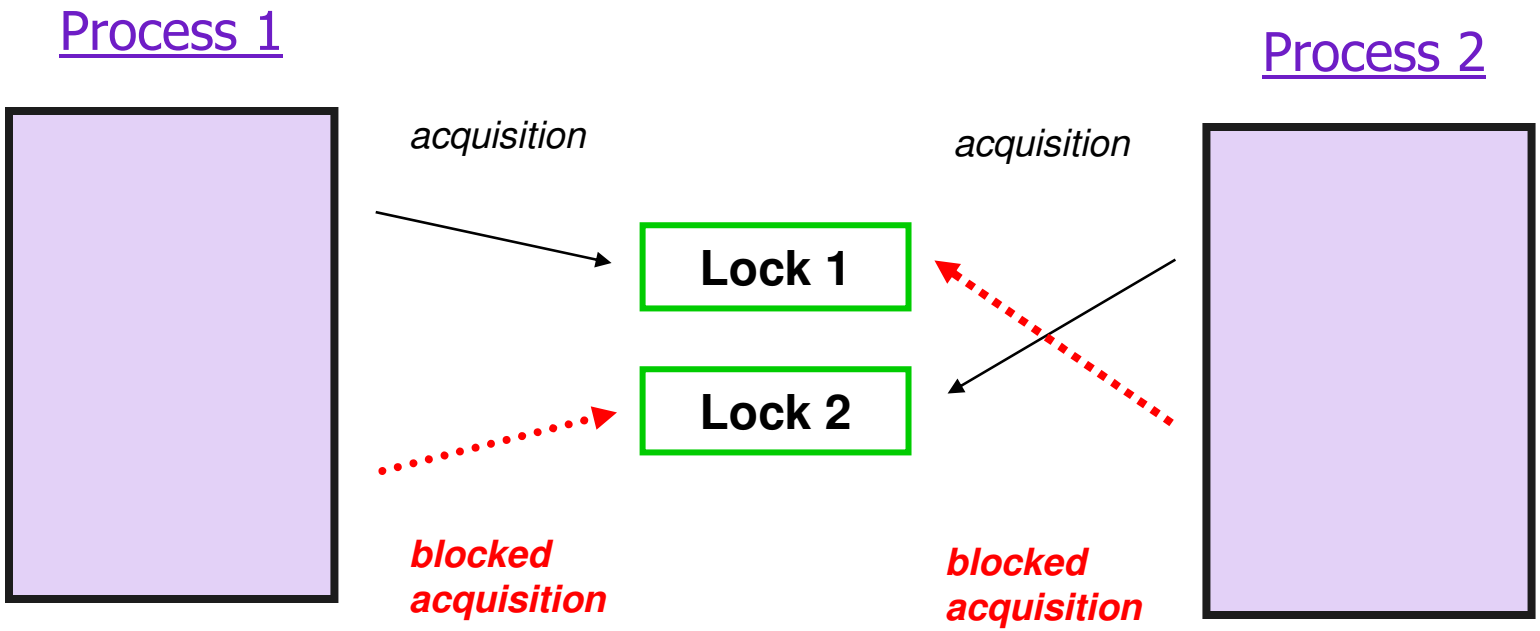
…some info supplied by static analysis – illustrating synergy between different techniques

Note: independence relation calculations, while modest in size and complexity to implement, have resulted in more than *four orders of magnitude reduction* in model checking concurrent Java program

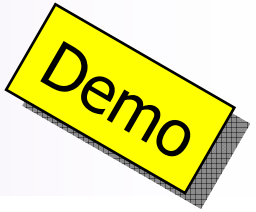
…this suite of optimizations has been incorporated in the NASA JPF model checker

Simple Deadlock Example

Demo



Simple Deadlock Example




```
public class Deadlock {
  static Lock lock1;
  static Lock lock2;
  static int  state;

public static
   void main(String[] args) {
     lock1 = new Lock();
     lock2 = new Lock();
     Process1 p1
         = new Process1();
     Process2 p2
         = new Process2();
     p1.start();
     p2.start();
}
```

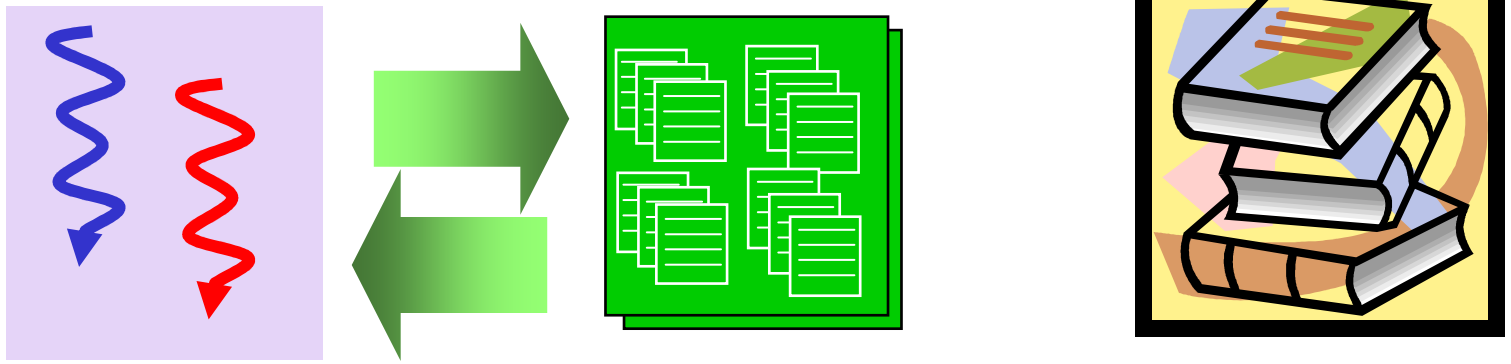
```
class Lock {}
```

```
class Process1 extends Thread {
  public void run() {
    Deadlock.state++;
    synchronized (Deadlock.lock1) {
      synchronized (Deadlock.lock2) {
        Deadlock.state++;
      }}}}
```

```
class Process2 extends Thread {
  public void run() {
    Deadlock.state++;
    synchronized (Deadlock.lock2) {
      synchronized (Deadlock.lock1) {
        Deadlock.state++;
      }}}}
```

Position in Development Flow

Apply in conjunction with unit testing, for highly concurrent units…

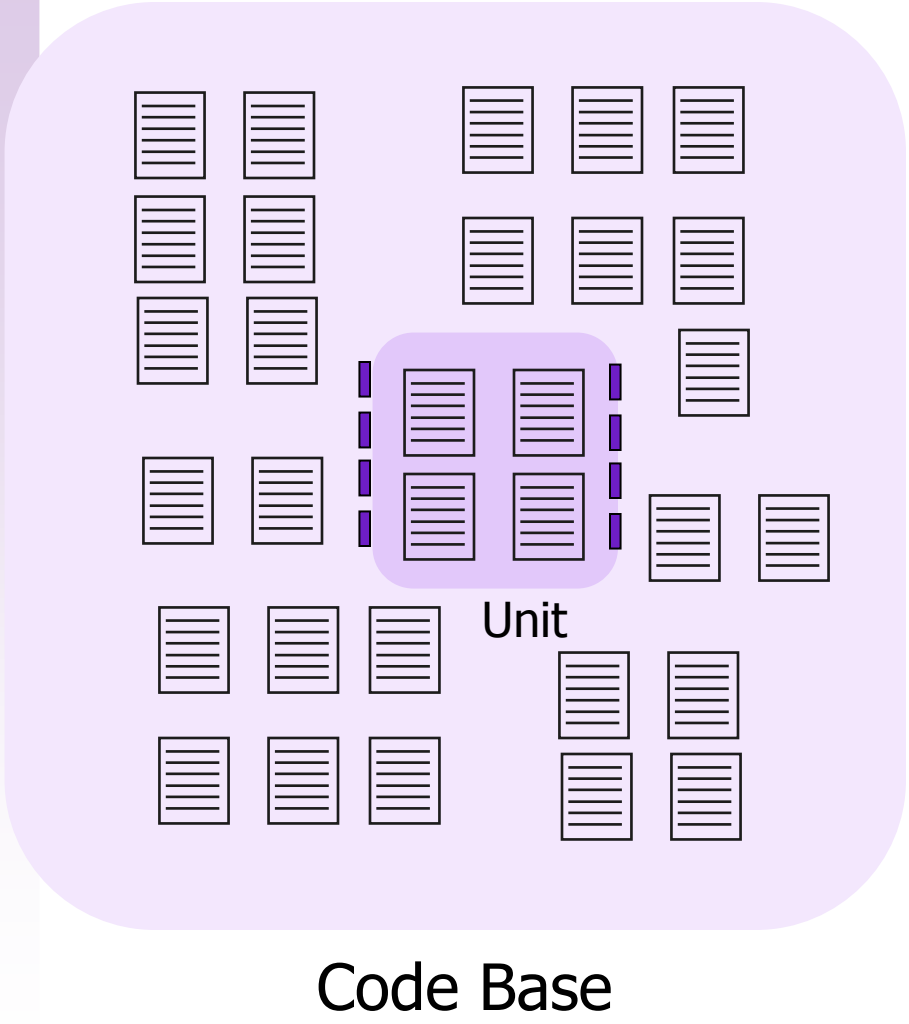


Test Harness + Unit under Test + Libraries

Code implementing a *driver* to *close the system*

- Need coverage metrics for *concurrency* to indicate…
  - when model checking techniques should be applied
  - the quality of the test harness

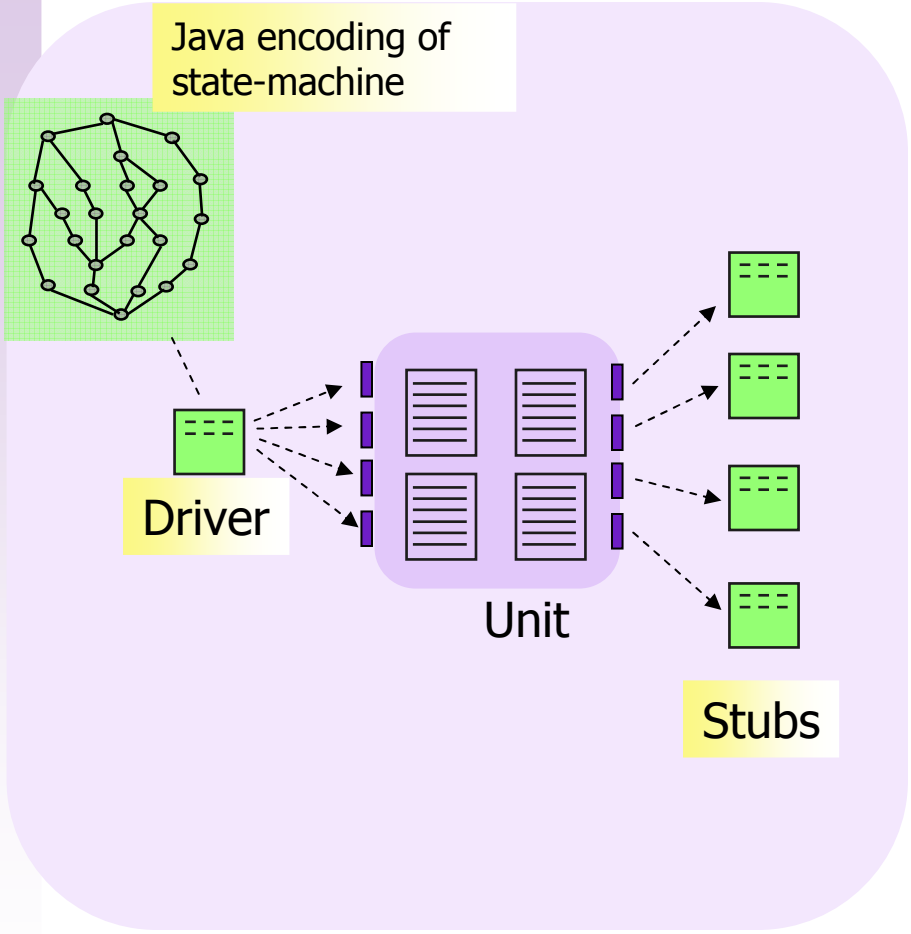
Aiding Unit Checking



Bandera Environment Generation Tools

- Identify classes in unit
- Automatically finds points of interaction (where unit calls outside classes or is called itself)

Aiding Unit Checking



Closed Unit

...connections to Daniel Brenner's talk

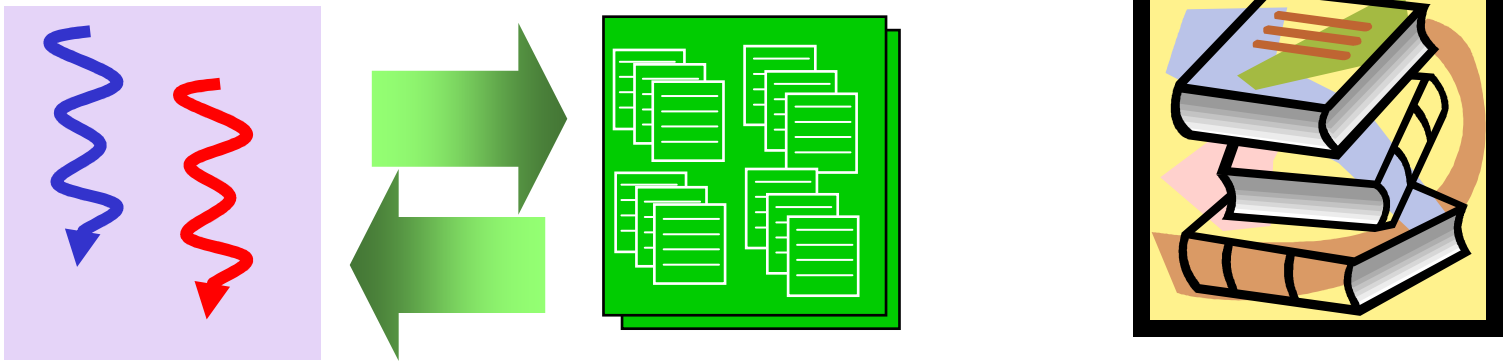
Bandera Environment Generation Tools

- Identify classes in unit
- Automatically finds points of interaction (where unit calls outside classes or is called itself)
- Cuts away non-unit classes
- Automatically generates driver (generates calls to unit based on regular expression or LTL formula)
- Automatically generates stubs -- partially filled based on side effect analysis

...by Tkachuk, Dwyer

Using Slicing to Prune Libraries

Apply in conjunction with unit testing, for highly concurrent units…



Test Harness + Unit under Test + Libraries

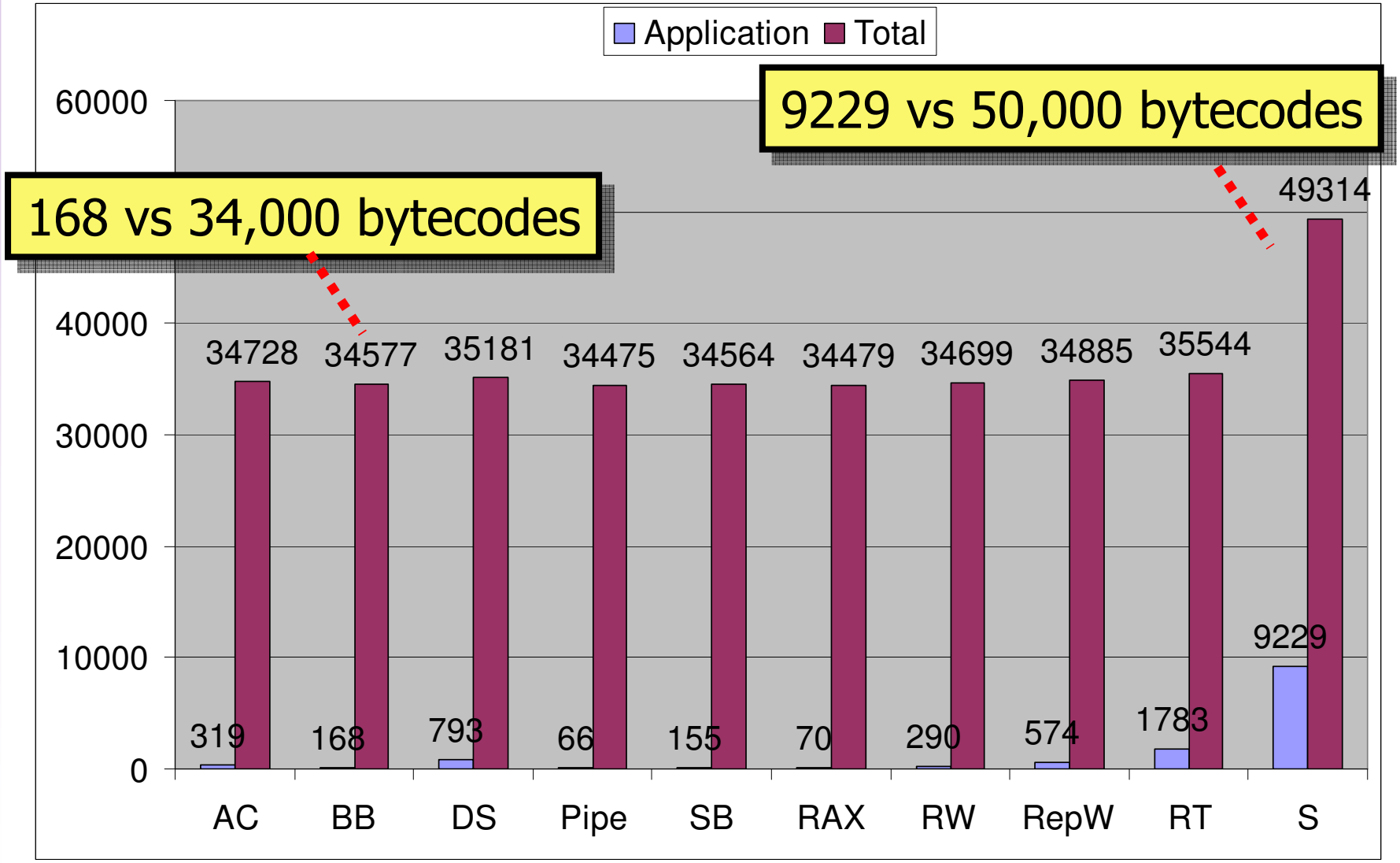
Code implementing a *driver* to *close the system*

- Idea: program slicing (static analysis) can be used to prune away unit and library code that's irrelevant for property being checked
- …*see our TACAS 2006 paper*

Classes of Examples

- Algorithm Sketches (used elsewhere in the literature)
  - Bounded Buffer
  - Pipeline
  - Sleeping Barbers
  - Readers/Writers
  - RAX (distillation of NASA Remote Agent Bug)
- Small Applications (significantly more heap intensive)
  - Disk Scheduler
  - Alarm Clock
- Larger Applications
  - Replicated Workers (client/server data distribution framework)
  - Ray Tracer (Java Grande)
  - Siena (internet-scale publish/subscribe middleware)

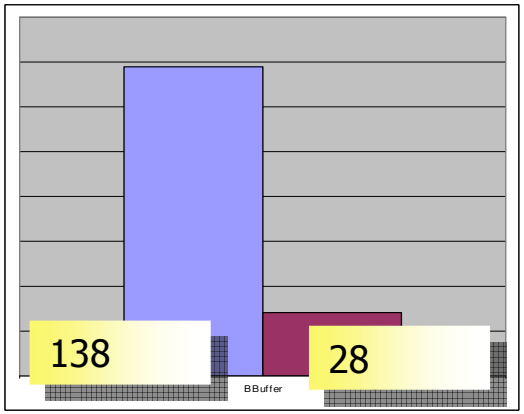
Size in Bytecodes



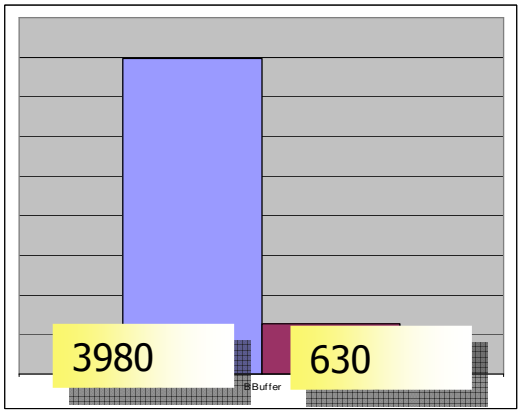
Dynamic Metrics - BBuffer w/POR

Results for checking deadlock (blue) – slicing optimizations (red)

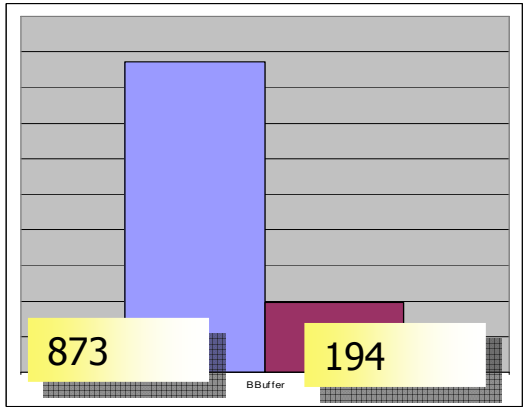
States



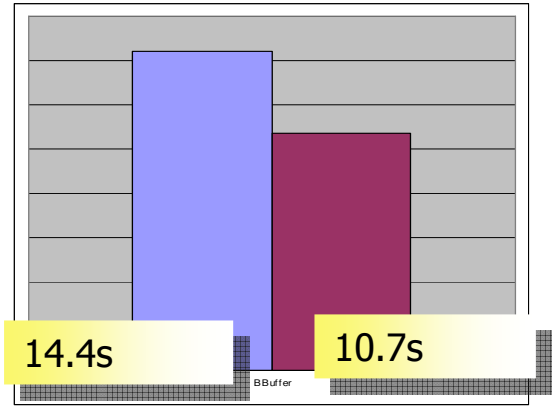
Transitions



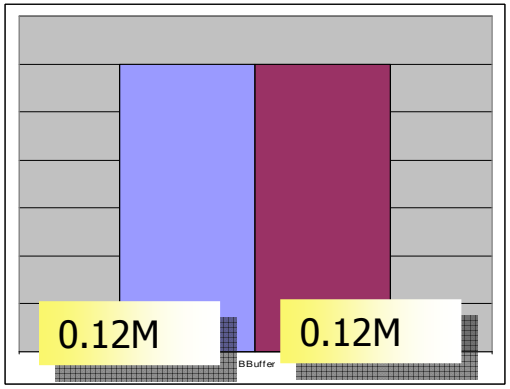
Byte-codes



Time



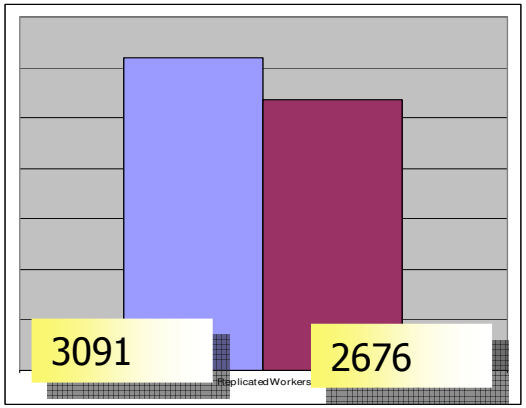
Memory



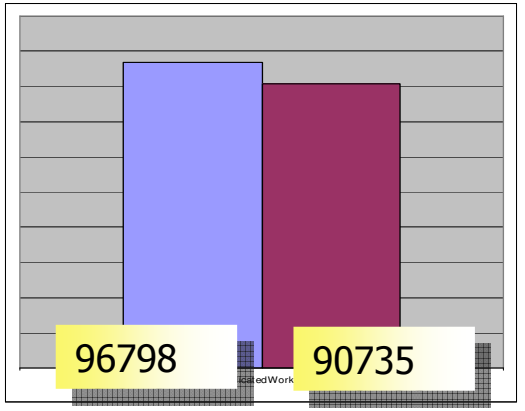
Dynamic Metrics - RepW (w/POR)

Results for checking deadlock (blue) – slicing optimizations (red)

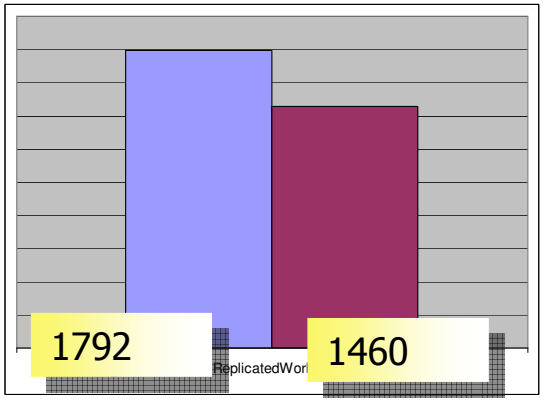
States



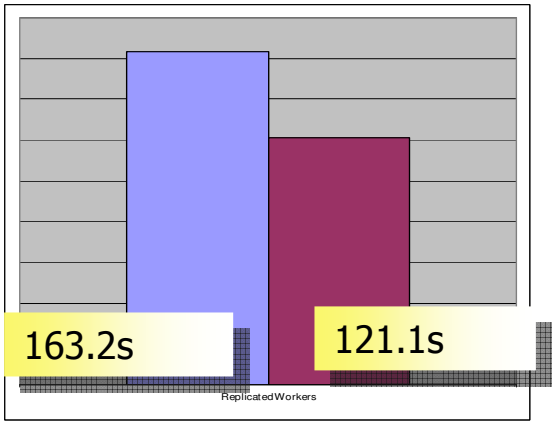
Transitions



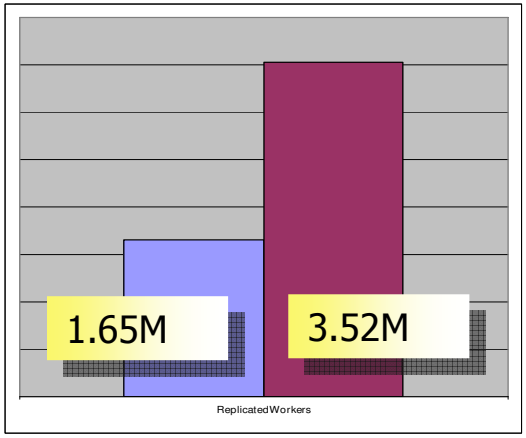
Byte-codes



Time



Memory

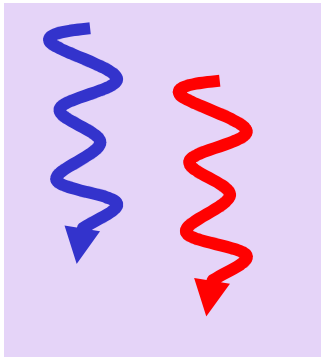


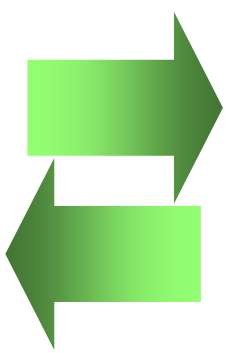
Lockheed Martin Tech Transition

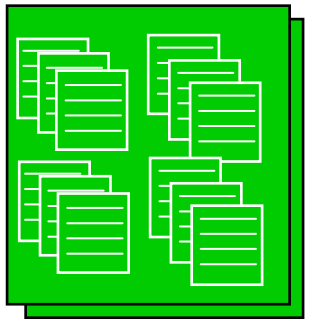
Bogor development currently being funding by Lockheed Martin IRAD program focusing on challenges of large-scale system integration

- Two LM engineers developing Bogor extensions to capture events of interest in C# executions
  - control structure, object & thread allocations, interthread communication
  - state space querying and visualization for understanding of complex system
- KSU developer building .NET bytecode interpreter
  - with no previous knowledge of Bogor or model-checking, was able to adapt Bogor Java VM to .NET VM in three months
- Target domain: satellite mission control systems

Recall Challenges with Methodology







Test Harness + Application + Libraries

Code implementing a *driver* to *close the system*

Application code

- *How can we avoid having to write a test harness?*
- *How can we break the applications and libraries into manageable chunks by taking a more compositional approach?*

idea: take a symbolic approach

Bogor / Kiasan

Kiasan – A Bogor Extension for Symbolic Execution

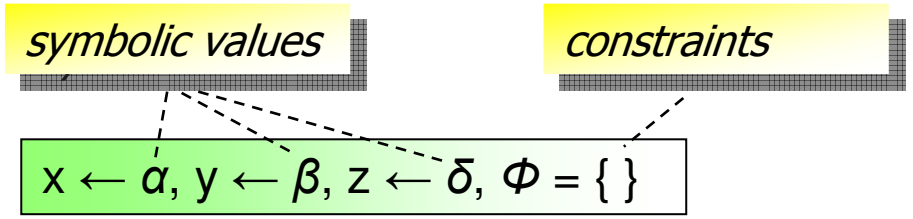
"kiasan"
=
"symbolic"

- combines…
  - model checking
  - theorem proving & decision procedures
  - constraint solving
- builds on…
  - classic symbolic execution (e.g., King)
  - recent techniques for dealing with objects (e.g., Kurshid, et.al. – JPF)
- formalization: soundness & relative completeness

- compositional checking of
  - light-weight specifications
  - strong statements about heap properties
  - e.g., exceeding capabilities of ESC/Java
- compact symbolic representation of collections of execution traces
- create concrete instances of symbolic traces
- automatic JUnit test-case generation
  - from specifications
  - from code
- bounding strategy gives controller and quantifiable notion of coverage

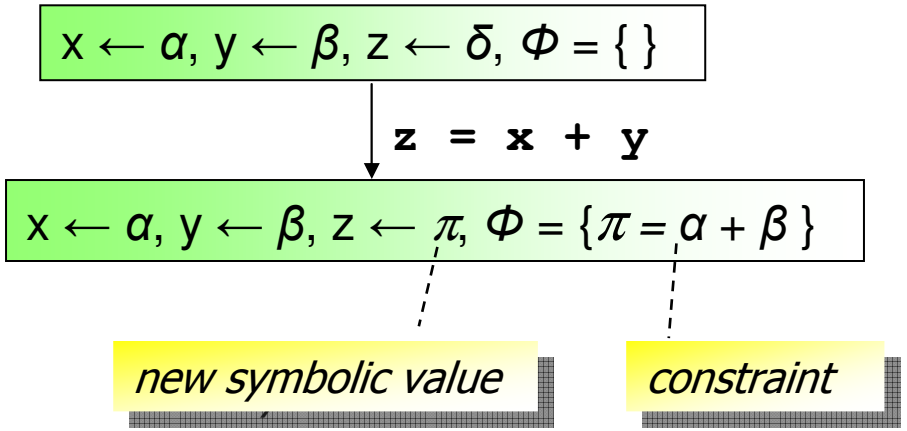
Background: Symbolic Execution

```
void foo(int x,
     int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



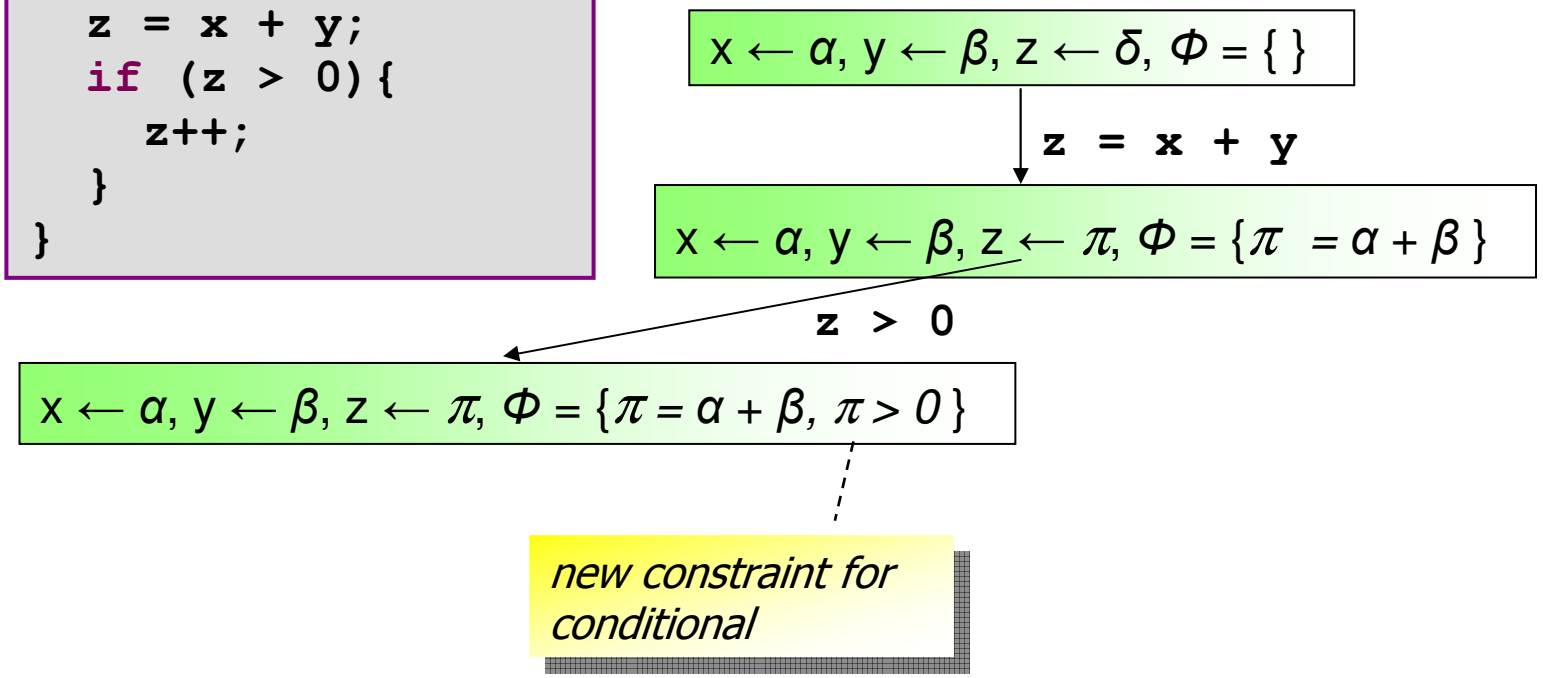
Background: Symbolic Execution

```
void foo(int x,
     int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



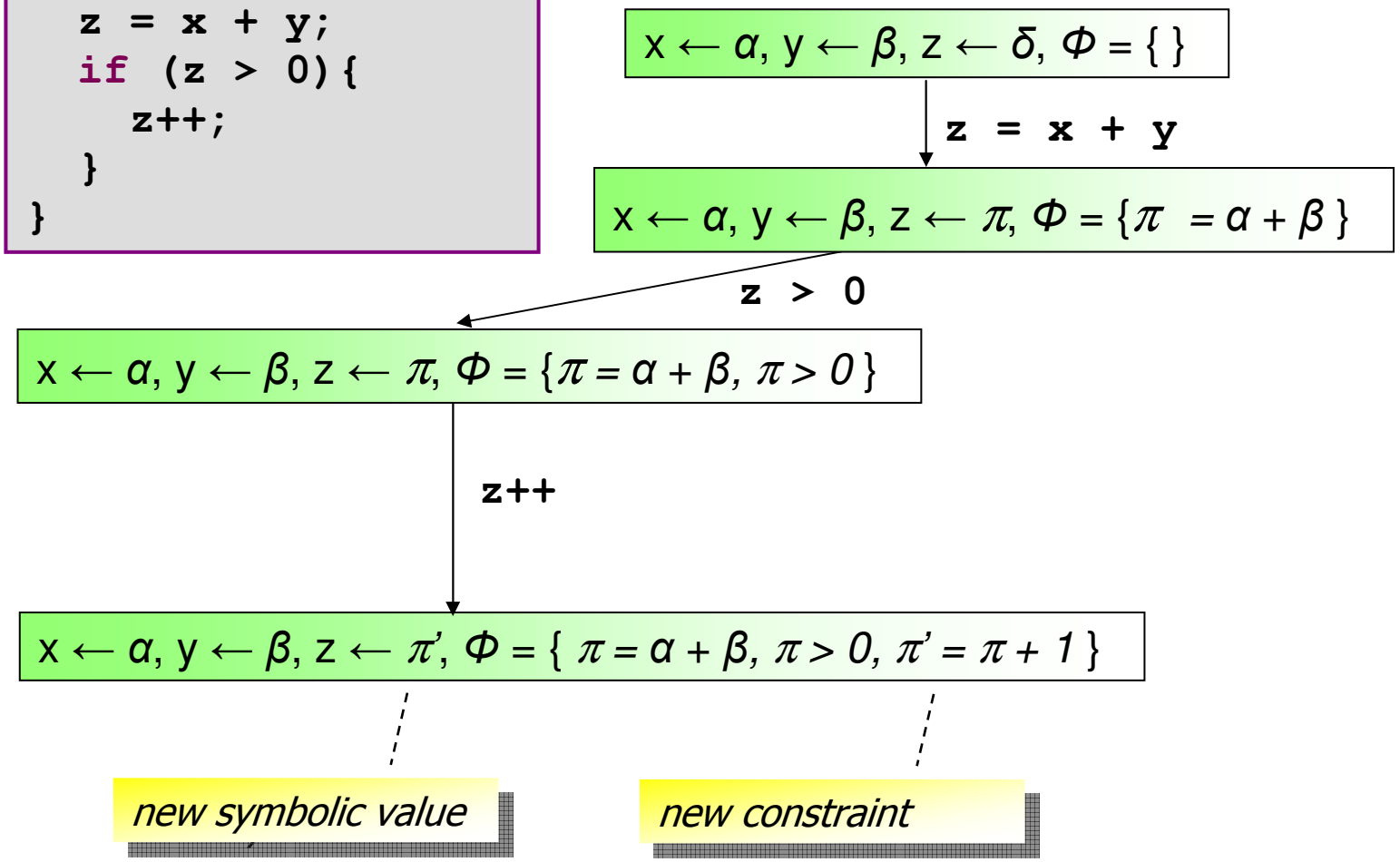
Background: Symbolic Execution

```
void foo(int x,
         int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```

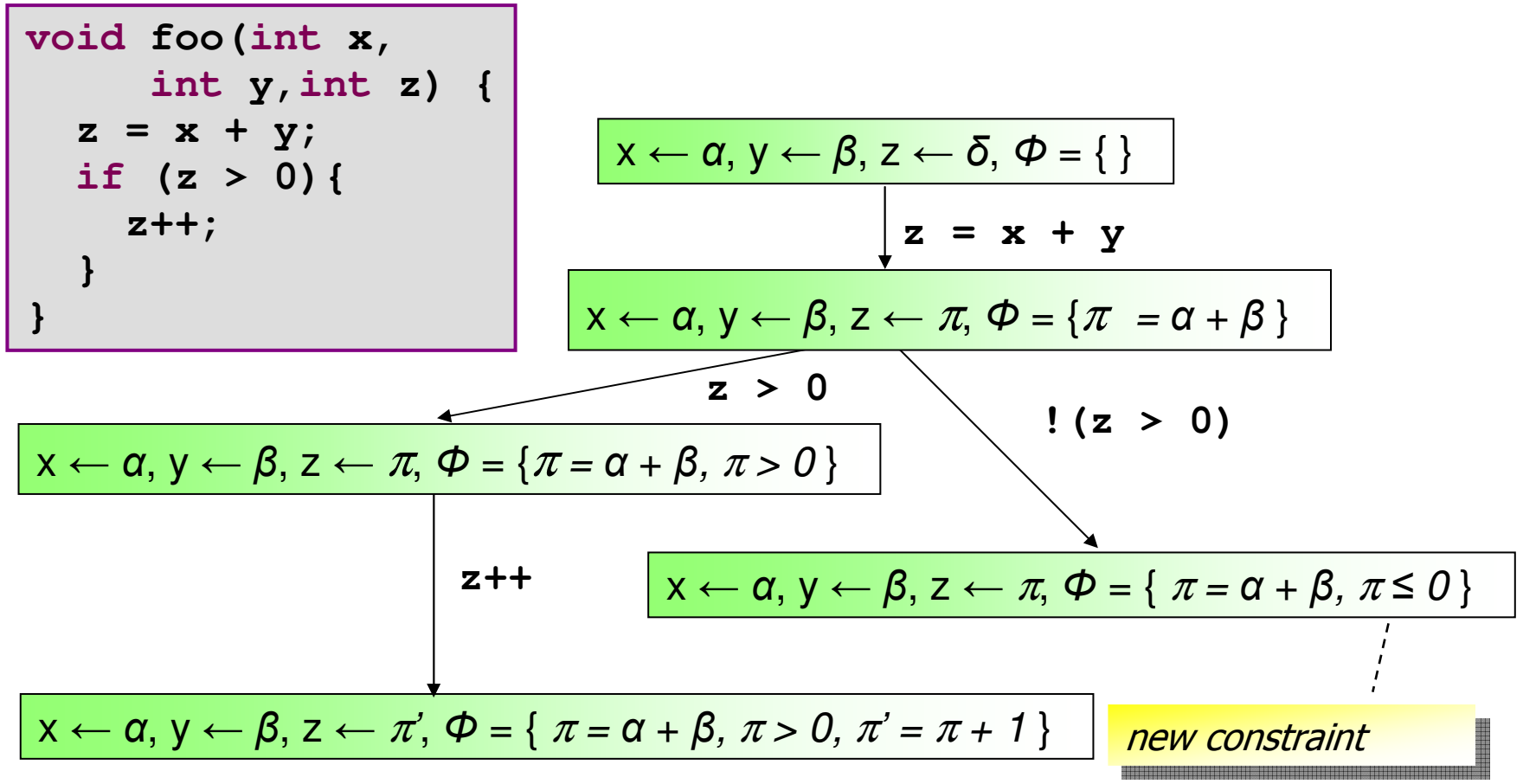


Background: Symbolic Execution

```
void foo(int x,
     int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



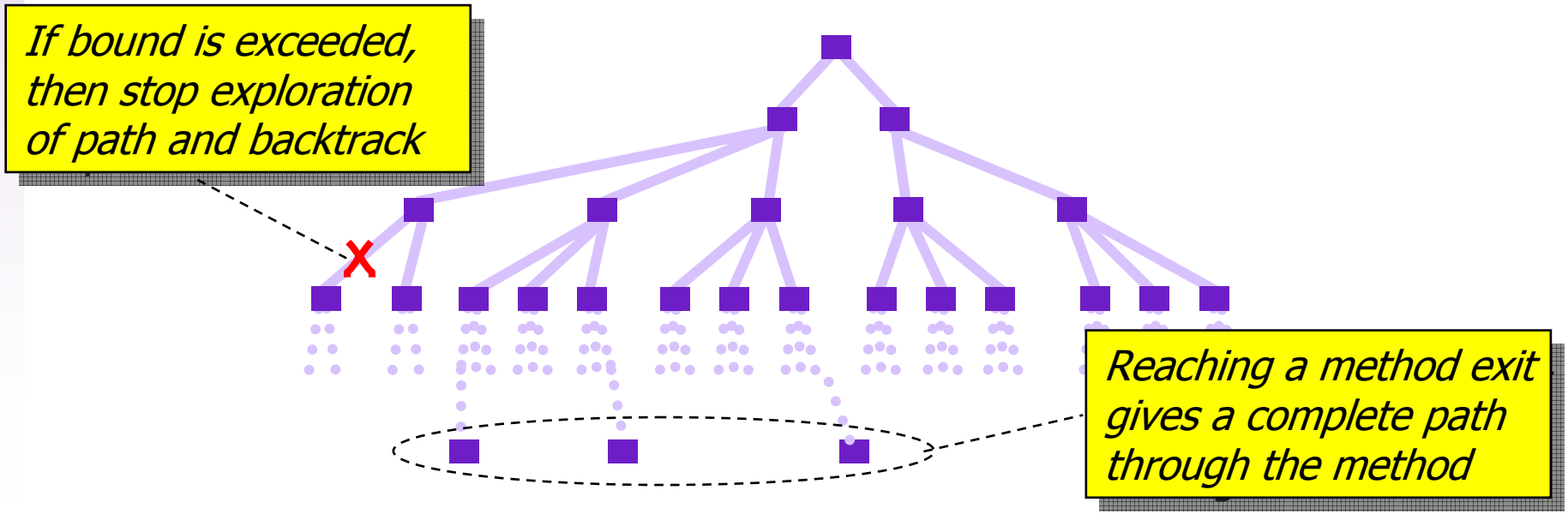
Background: Symbolic Execution



…symbolic execution characterizes (theoretically) infinite number of real executions!

Issue: Handling Loops

- How do we know when to quit going around a loop?
  - Could leverage loop invariants, but that is difficult for several reasons
  - Common strategy is to use different forms of bounds
    - bound total number of steps, or
    - bound number of loop iterations



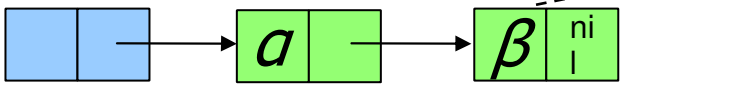
Representing Heap Data

How should dynamically allocated heap data be represented in symbolic execution?

…model checker maintains a representation of the heap *…take advantage of that*

Combined Concrete & Symbolic Representation

…abstract heap location; represents an arbitrary heap structure



…use conventional symbolic constraints on scalars in heap

$\alpha > \beta$

…lazily expand symbolic representation as program interacts with the heap

Representing Heap Data

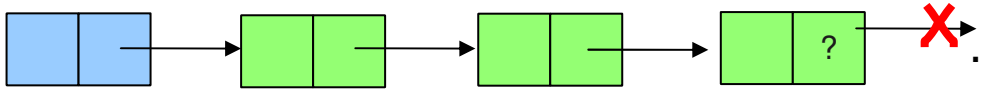
How should dynamically allocated heap data be represented in symbolic execution?

...model checker maintains a representation of the heap *...take advantage of that*

Bound search by bounding length of reference chains

length limit k = 3

...# references is 2; keep expanding

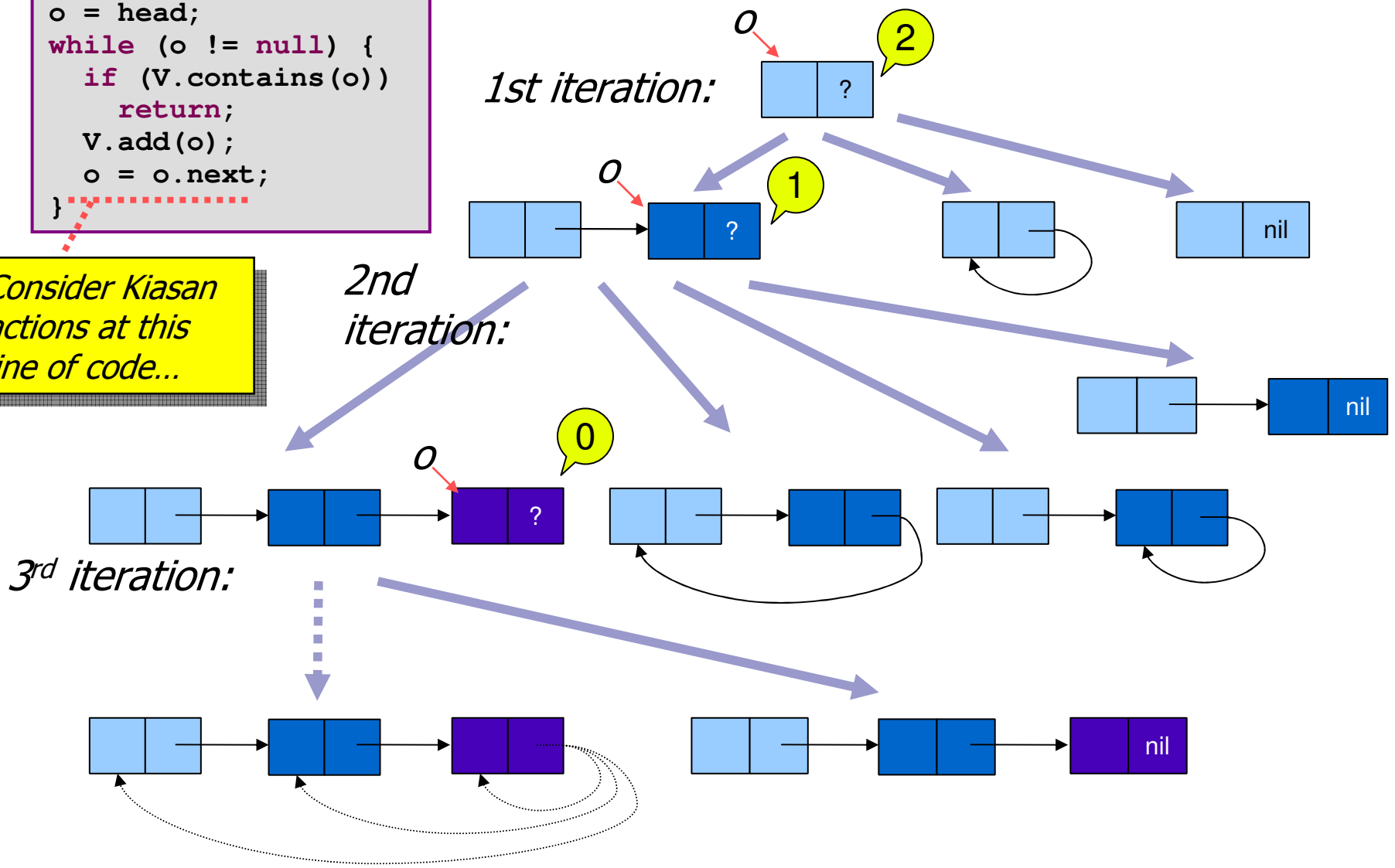


...backtrack when execution generates a chain longer than *k*

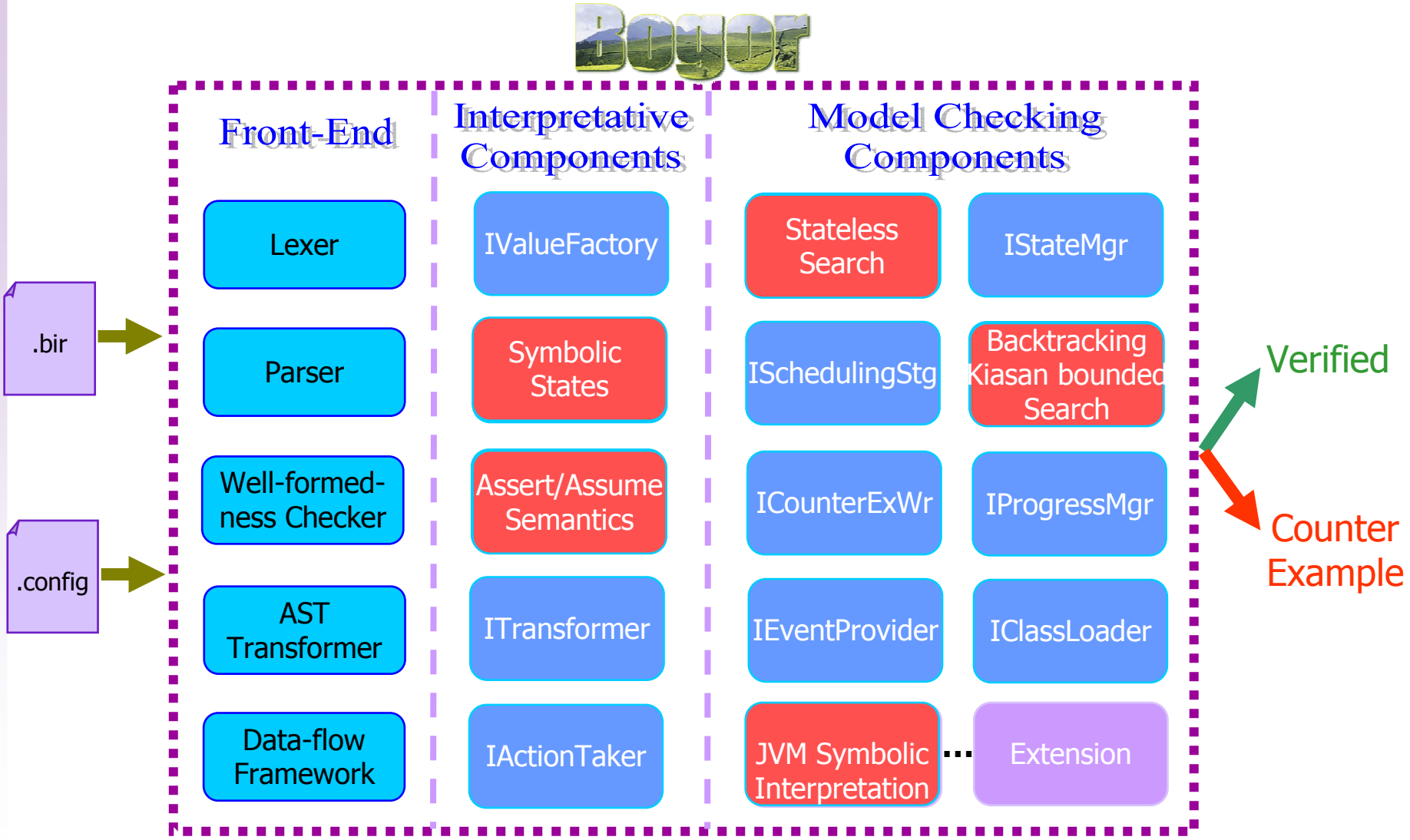
Handling Objects using Lazy Initialization ($k = 2$): LinkedList

```
o = head;
while (o != null) {
  if (V.contains(o))
    return;
  V.add(o);
  o = o.next;
}
```

Consider Kiasan actions at this line of code…



Modules Customized for Kiasan



...modular components with clean and well-designed API using design patterns

Themes – Code Analyses

Sales pitch for those who believe that…
"The ***code*** is the ***truth***, and the only ***truth***!"

- Static Detection of Possible Runtime Exceptions
  - Null dereference
  - Array index out of bounds
  - Division by zero
  - Type casting error
  - …. etc.
- Code Understanding
  - auto-generation of input/output scenarios
- Test Case Generation from code

Static Detection of Possible Runtime Exceptions

- Similar to ESC/Java2, Kiasan can catch possible runtime exceptions in Java

```
class Node {
  Node next;
  int x;
}

void foo2(Node n)   {
  if (n.x > 0) {
    n.x++;
    return;
  }
  n.x = 0;
}
```

possible null dereference

Detecting Assertion Violations

- Consider the following example:

```
void foo3(Node n1, Node n2) {
  if (n1 != null && n2 != null) {
    n1.x = 2;
    n2.x = 3;
    assert (n1.x == 2));
  }
}
```

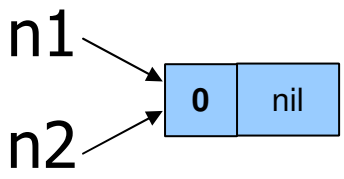
possible assertion violation!

Detecting Assertion Violations

- Not only that Kiasan can detect the assertion violation, it can even give the scenario where the assertion is violated

```
void foo3(Node n1, Node n2) {
  if (n1 != null && n2 != null) {
    n1.x = 2;
    n2.x = 3;
    assert (n1.x == 2);
  }
}
```

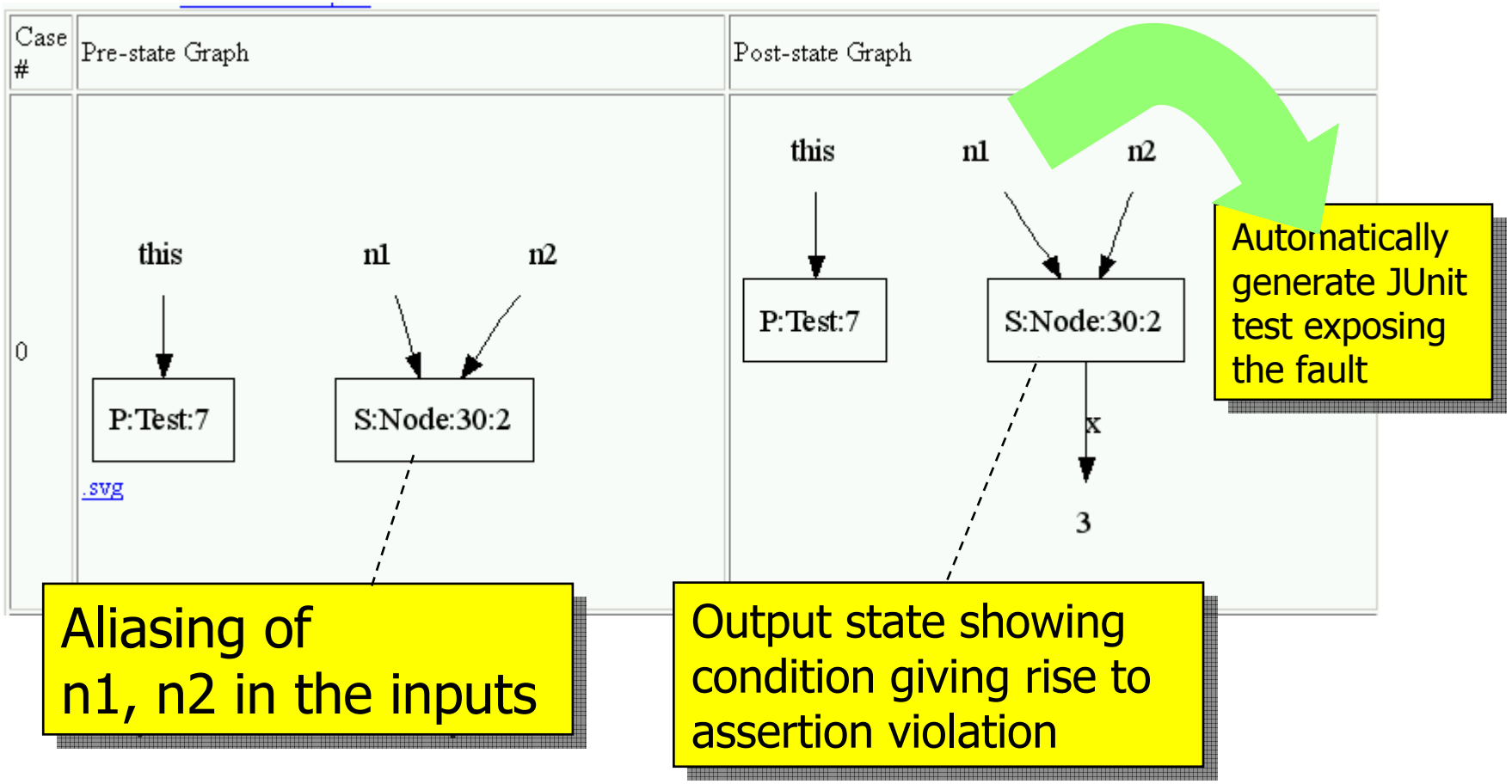
Error Case



Kiasan Output

Pre-state Graph

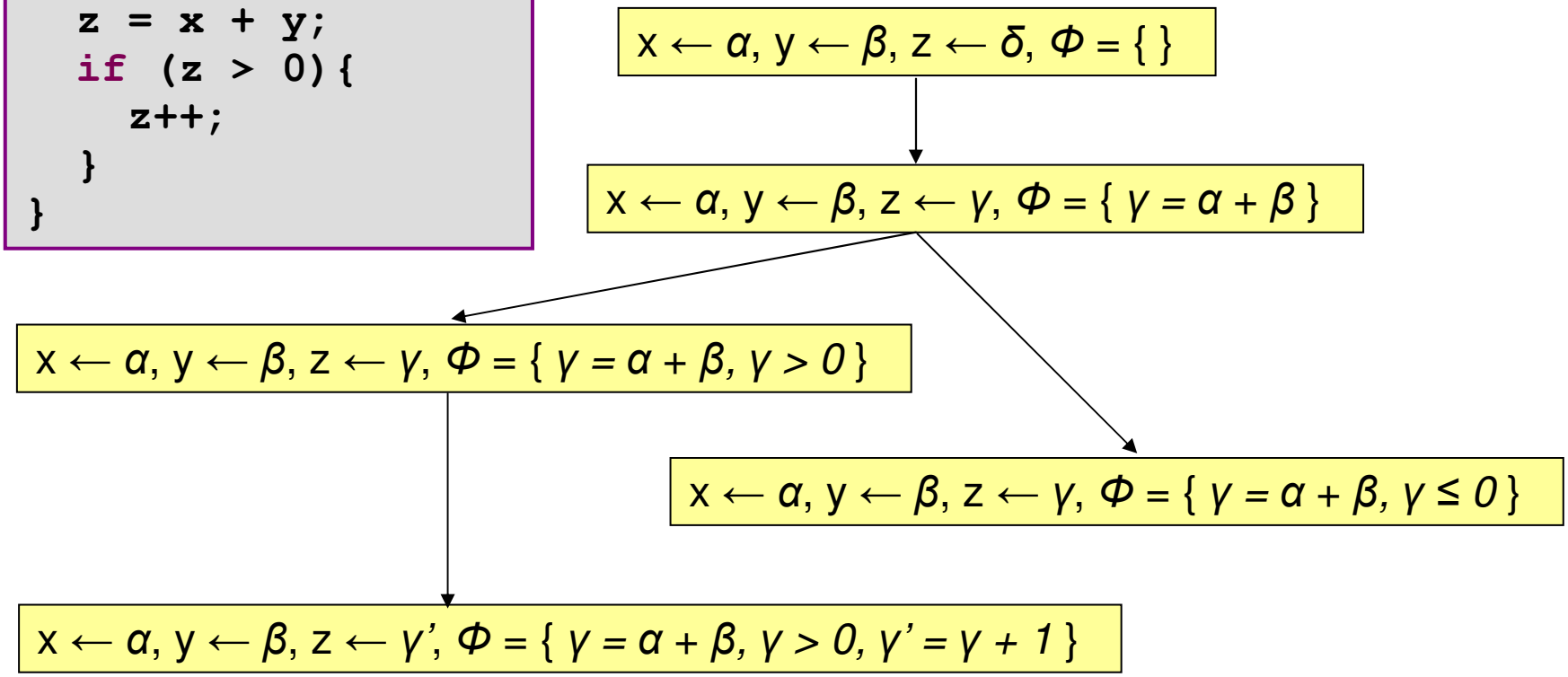
Post-state Graph



Collecting All Paths

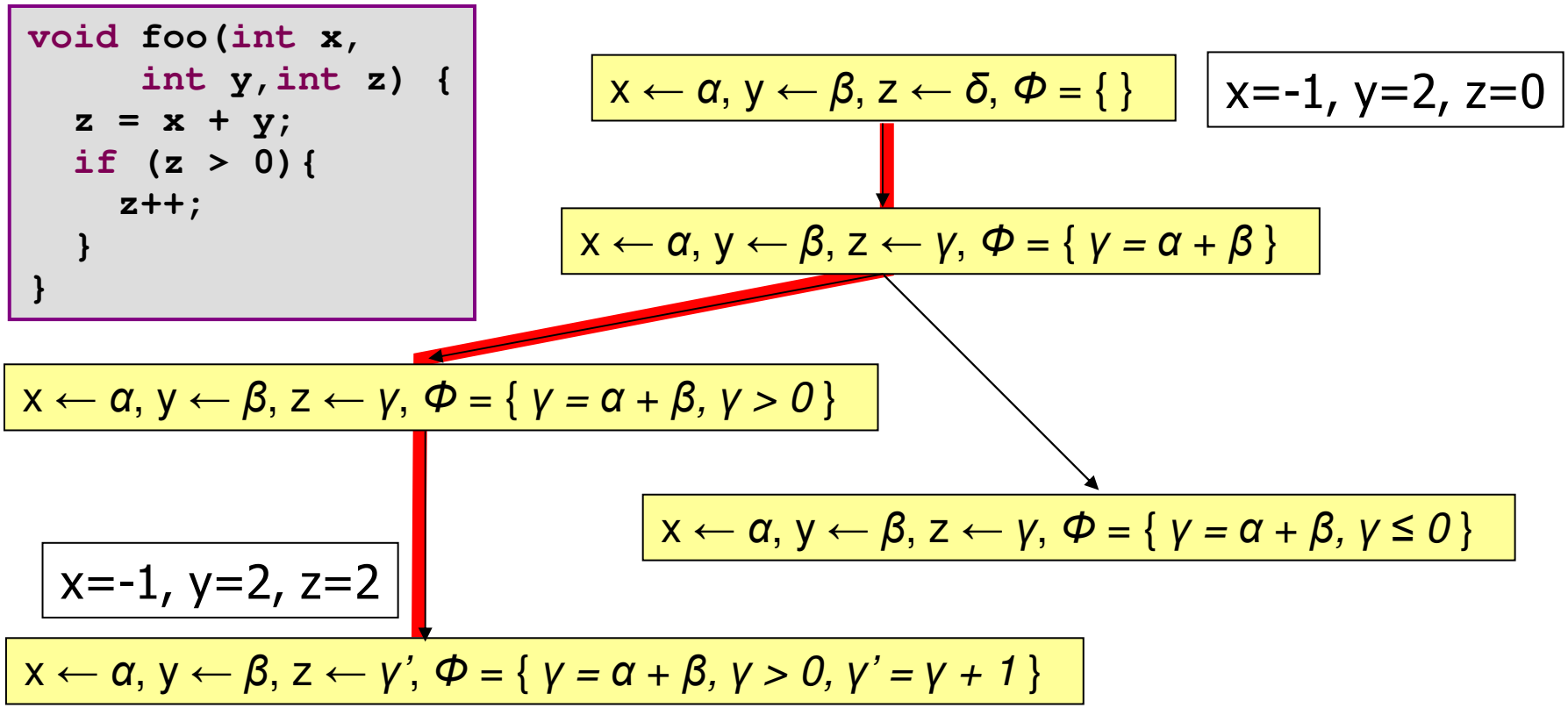
- Recall our example earlier:

```
void foo(int x,
     int y,int z) {
  z = x + y;
  if (z > 0){
    z++;
  }
}
```



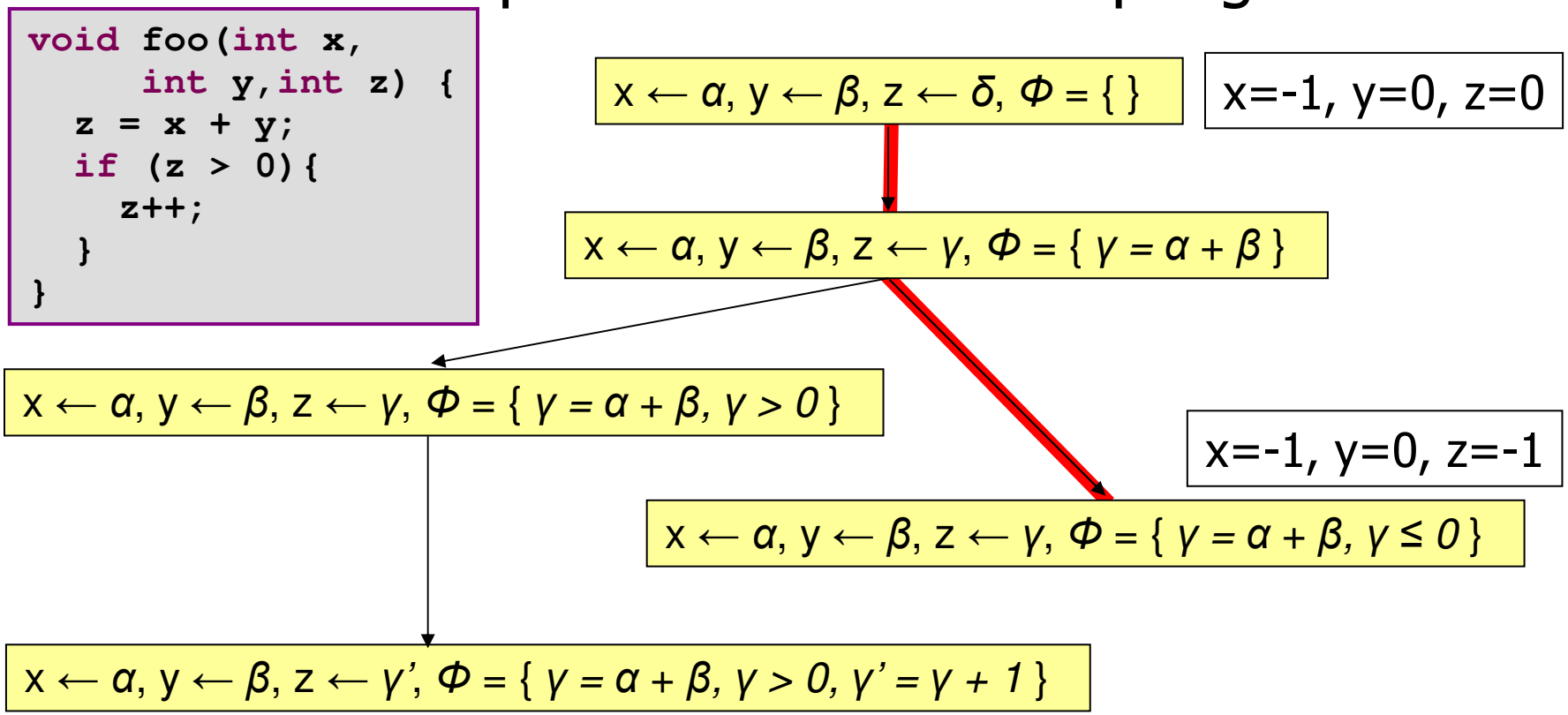
Collecting All Paths

- Kiasan finds each path through method
  - collects constraints
  - solves constraints to generate symbolic/concrete



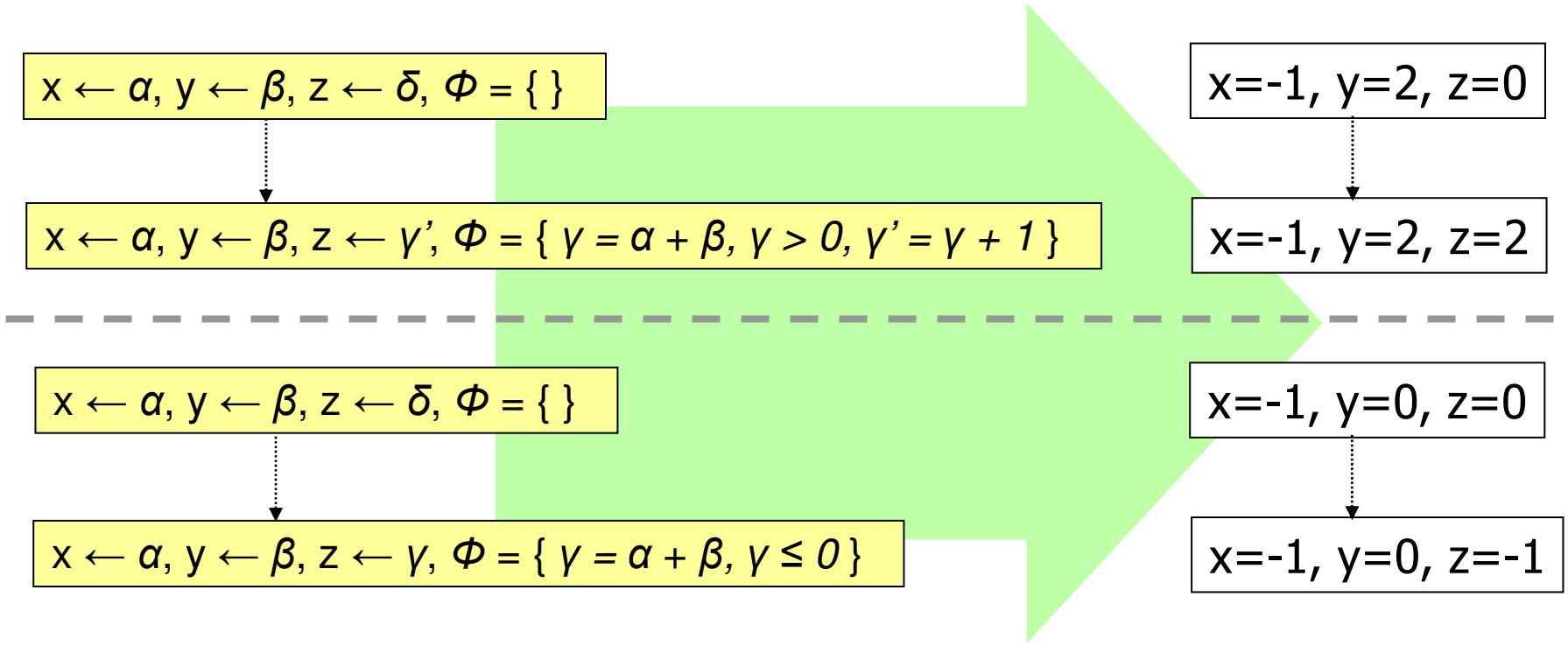
Code Understanding

- Kiasan can be used to understand the computation along the traces (concrete) using similar techniques to counter example gen.

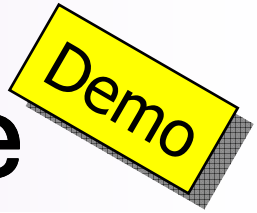


Code Understanding

- While Kiasan does not give all scenarios for *all possible input values (there could be an infinite number)*, it does give concrete traces representing *all distinct paths* in the program computation tree (i.e., it gives exhaustive path coverage up to our bounding strategy)



Example: Binary Search Tree



Debugging/Testing/Understanding Insert Method

```
 private BinaryNode myins( int x, BinaryNode t )
         {
/* 1*/         if( t == null )
/* 2*/             t = new BinaryNode( x, null, null );
/* 3*/         else if( x < t.element)
/* 4*/             t.left = myins( x, t.left );
/* 5*/         else if( x> t.element )
/* 6*/             t.right = myins( x, t.right );
/* 7*/         else
/* 8*/             ;  // Duplicate; do nothing
/* 9*/         return t;
         }
```

...generate scenarios/test cases giving complete path coverage for heap data with reference chains up to length K=2

Automatic Test Case Generators and Aiding Code Inspection

Demo

- Extends the generation of error scenarios to generate test cases
  - generate cases for "good" behaviors
  - While test generations should not based on code alone, this is valuable for regression testing
- This can be used for code inspection
  - the generated input/output (side-effects) of a method give some clue about the method's behavior
  - generalize to any statement block

Kiasan Methodology

- Checking in IDE
  - start with small bounds
  - incrementally check
  - scenario and test case generation for violations
- More exhaustive checking
  - higher bounds with overnight/parallel checking

- Code understanding
  - select any block of code, Kiasan generates flow scenarios giving path coverage
- Test case generation for regression testing
  - automatically generate tests from code
  - incrementally add tests as changes are made

Themes - Spec 'n Code Analyses

Sales pitch for those who believe that…

"***Without specification***,
the ***code*** is trivially ***correct*** !"

- Contract-based Reasoning
- Test Case Generation – constrained by specifications

Strong Property Checking

Kiasan can check specifications in fashion similar to ESC/Java but does much better with heap properties

```
public class LinkedList<E> {
  //@ inv: isAcyclic();

  /*@ pre:  isSorted(c) && other.isSorted(c);
    @ post: isSorted(c)
    @       && size() = \old(size()) + other.size()
    @       && (\forall E e;
    @           elements.contains(e);
    @           \old(this).contains(e)
    @           || other.contains(e))
    @*/
  void merge(@NonNull LinkedList<E> other,
             @NonNull Comparator<E> c) {
    …
  }
}
```

Strong Property Checking

Kiasan can check specifications in fashion similar to ESC/Java but does much better with heap properties

every linked-list is acyclic

this list is sorted and the other list is sorted based on the Comparator

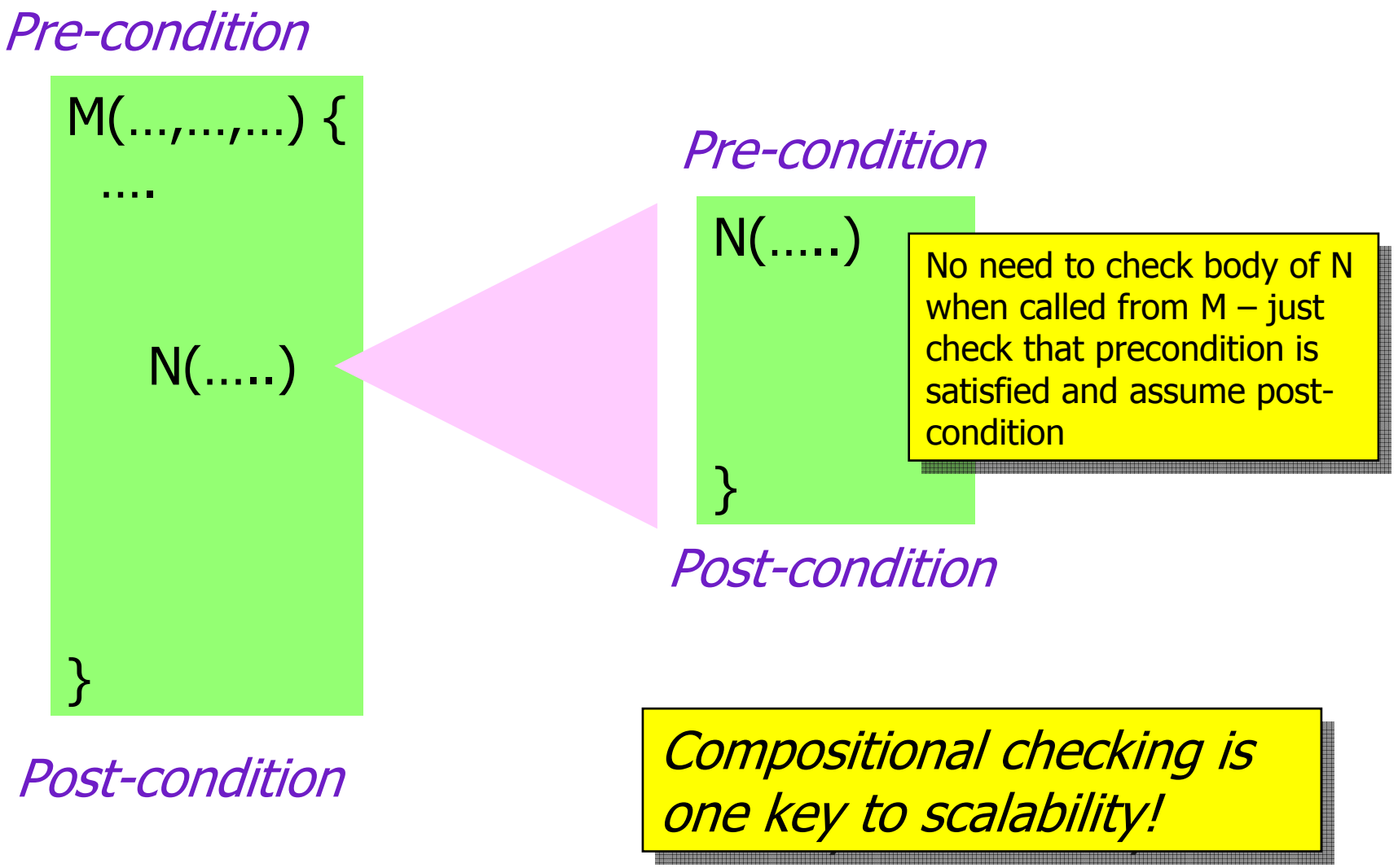
this list is sorted

the size is equal To the other size plus this list's old size

all the elements are from the previous two list

```
public class LinkedList<E> {
  //@ inv: isAcyclic();

  //@ ensures: isSorted(c) && other.isSorted(c);
  //@ ensures: isSorted(c)
  //@         && size() = \old(size()) + other.size()
  //@         && (\forall E e;
  //@             elements.contains(e);
  //@             \old(this).contains(e)
  //@             || other.contains(e))

  NonNull LinkedList<E> other
  NonNull Comparator<E> c) {
```

Checking is Composition



Experimental Results

Example	k	Pre	B-Post	A-Post	States	Time
Array	1	1	1	1	181	0:00.6/0:00.2
Binary Heap	2	2	2	2	332	0:01.0/0:00.6
`deleteMin()`	3	3	4	4	628	0:02.3/0:01.7
`(int)`	4	4	7	7	1.1k	0:05.0/0:04.3
Array	1	1	1	1	86	0:00.4/0:00.0
Insertion Sort	2	1	3	3	214	0:00.7/0:00.3
`sort()`	3	1	9	9	760	0:02.1/0:01.6
`(int)`	4	1	33	33	3.4k	0:10.8/0:10.0
Linked-list	1	1	1	1	667	0:00.6/0:00.0
`merge()`	2	4	5	5	3.3k	0:01.0/0:00.0
`(int)`	3	9	19	19	16.1k	0:02.6/0:00.6
	4	16	69	69	78.5k	0:11.2/0:04.9
Binary	1	2	4	4	1.6k	0:00.8/0:00.1
Search Tree	2	5	21	21	12.6k	0:02.7/0:01.1
`insert()`	3	26	236	236	233k	0:55.8/0:39.5
`(int)`	4	-	-	-	-	> 10*min*
Red-black Tree	1	2	5	5	1.4k	0:00.9/0:00.1
`remove()`	2	6	43	43	34.7k	0:07.3/0:04.2
`(int)`	3	31	579	579	1M	5:25.9/4:17.5
	4	-	-	-	-	> 10*min*

Example	k	Pre	B-Post	A-Post	States	Time
Array	1	1	1	1	218	0:00.7/0:00.3
Binary Heap	2	2	2	2	418	0:01.4/0:00.9
`deleteMin()`	3	3	4	4	819	0:04.3/0:03.6
`(Comparable)`	4	4	7	7	1.5k	0:14.5/0:13.6
Array	1	2	2	2	179	0:00.6/0:00.1
Insertion Sort	2	3	4	4	376	0:01.3/0:00.8
`sort()`	3	4	10	10	1.1K	0:06.4/0:05.6
`(Comparable)`	4	5	34	34	4.4k	1:15.1/1:13.9
Linked-list	1	1	1	1	904	0:01.1/0:00.0
`merge()`	2	4	5	5	4.0k	0:01.1/0:00.0
`(Comparator)`	3	9	19	19	18.7k	0:06.2/0:03.9
	4	16	69	69	89.5k	1:06.6/0:58.1
Binary	1	2	4	4	2k	0:00.9/0:00.1
Search Tree	2	5	21	21	15.3k	0:03.1/0:01.1
`insert()`	3	26	236	236	285k	1:09.7/0:49.0
`(Comparator)`	4	-	-	-	-	> 10*min*
Red-black Tree	1	2	5	5	1.9k	0:00.9/0:00.1
`remove()`	2	6	43	43	40k	0:08.3/0:04.7
`(Comparator)`	3	31	579	579	1.3M	6:01.5/4:40.5
	4	-	-	-	-	> 10*min*

Moving forward

- Use traditional coverage metrics to tell when to stop
- Parallelize search

Unit Testing Today

```
Object expected =
  new Object[] { ... };
Object result =
  sort(new Object[] { ... },
       new Comparator()
           { ... }));
// OK if a === result


Object[] sort(Object[] a,
          Comparator c)
{ ...

  c.compare(a[i],
            a[i+1]);
  ...
}
```

- When testing a method, developers need to write
  - the method's input
  - the expected output
  - some *mock* objects to close the unit testing
    - e.g., necessary to simulate network communications or GUI operations
- It is expensive to (manually) achieve good path and data coverage

Unit Testing Using Kiasan

```
TestRequirement: sort
- input: non-null a, c,
         elements of a
- output:
    asExpected(result, c)
// asExpected is a method


Object[] sort(Object[] a,
          Comparator c)
{ ...

  c.compare(a[i],
            a[i+1]);
  ...

}
```

- When testing a method, developers need to write
  - *descriptions* of the method's input and output
    - comes from software requirements (design intentions)
  - some mock objects to close the unit testing
    - e.g., necessary to simulate network communications or testing GUI applications
    - in future, generate mock objects from test requirements
- Kiasan generates test cases with good path and data coverage (coverage/cost is adjustable)

Samples of Design Intentions

Specifying common patterns

- Null-ness

```
class LinkedList { @NonNull LinkedNode head; }

class LinkedList { @MaybeNull LinkedNode head; }
```

- Null-ness of a container's element

```
class TreeNode {
 @NonNull @NonNullElements Set<TreeNode> children;
}
```

Samples of Design Intentions

Specifying common patterns

- Cyclic/Acyclic

```
class LinkedList { @Acyclic LinkedNode head; }
```

OR

```
@Acyclic("head") class LinkedList { ... }
```

- Tree/Graph

```
@Tree("children") class TreeNode {
  Set<TreeNode> children;
}
```

Kiasan Leverages Design Intentions

- Design intentions are used
  - for documentation purposes
  - as properties to check (statically)
    - design-by-contract style reasoning ala Eiffel
    - (in general, Kiasan can check any assertion that developers are willing to write, even allowing properties of two states)
  - for driving test case generations
    - filtering out input configuration that does not satisfy the design intentions

Comparing To Commercial Tools

- Commercial Test Case Generation
  - Coverage Metric
  - Capabilities
  - Simple test results
- Kiasan Test Case Generation
  - Capabilities
  - Test results

Method 2

Example: More complex flow control….

```
int SimpleDualInBranchAdd(int a, int b) {
   int i = (a + b);
   if (i > 500) {
      return 0;
   } else {
      return 1;
   }
}
```

The branch condition uses integer variable ‘i’, which stores the result of the addition of both method arguments.

Method 2: CodePro Output

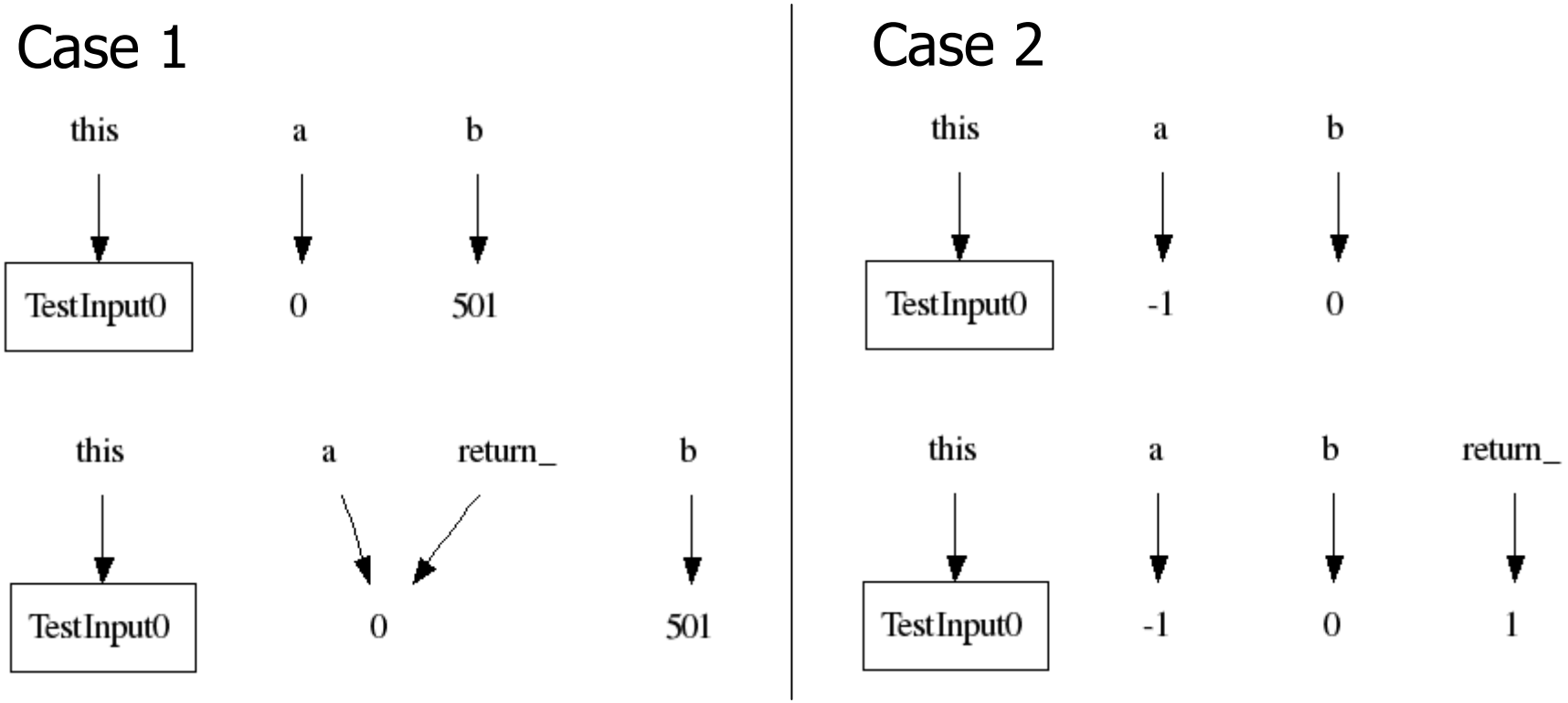
Test Case for (i <= 500) path

CodePro failed to generate test cases that would exercise the (i > 500) path.

```
void testSimpleDualInBranchAdd_fixture_1()
      throws Exception {
   TestyMcTest fixture = getFixture();
   int a = -1;
   int b = -1;
   int result =
         fixture.SimpleDualInBranchAdd(a, b);
   // add test code here
   assertEquals(1, result);
   fail("unverified");
}
```

Method 2: Kiasan Output

Kiasan generates JUnit test cases for both code paths



Kiasan Deals with Heap Data

Examples

- Array sorting (insertion sort)
- Binary heap
- Merging sorted lists
- Binary search tree
- Red-black tree (from java.util.TreeMap)

Dealing with Heap Data(cont.)

```
@Assertion(@Case(
  pre = "repOK(t)",
  post = "repOK($ret)"))
private BinaryNode myins( int x, BinaryNode t ) {
  if ( t == null )
    t = new BinaryNode( x, null, null );
  else if( x < t.element)
    t.left = myins( x, t.left );
  else if( x> t.element )
    t.right = myins( x, t.right );
  else
    ;  // Duplicate; do nothing
  return t;
}
```

High level assertions can apply design intentions that are codified in Java

Dealing with Heap Data(cont.)

```
@Assertion(@Case(
      pre = "repOK(root)",
      post = "repOK(root)"))

public void insert( int x )
{
  root = myins( x, root );
}
```

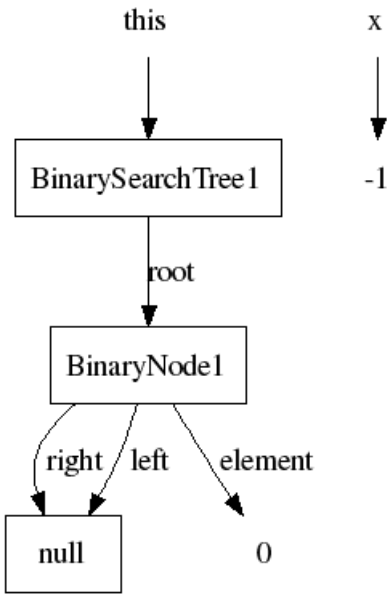
Dealing with Heap Data(cont.)

```
boolean repOK(BinaryNode t) {
  return repOK(t,new Range());
}

boolean repOK(BinaryNode t, Range range) {
  if (t == null) return true;

  if (!range.inRange(t.element)) return false;

  boolean ret = true;
  ret = ret &&
        repOK(t.left,range.setUpper(t.element));
  ret = ret &&
        repOK(t.right,range.setLower(t.element));
  return ret;
}
```

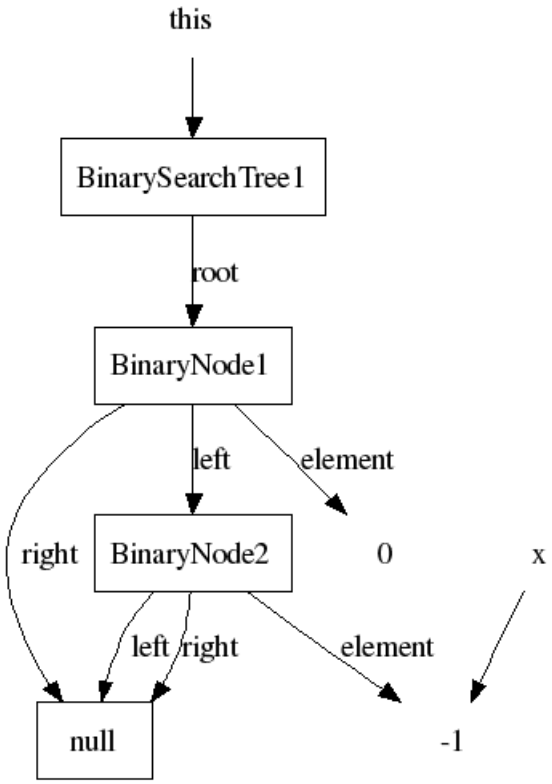
High level assertions can apply design intentions that are codified in Java

Dealing with Heap Data: Output

Pre-State: this.insert(-1)



Post: isOK(this.root)



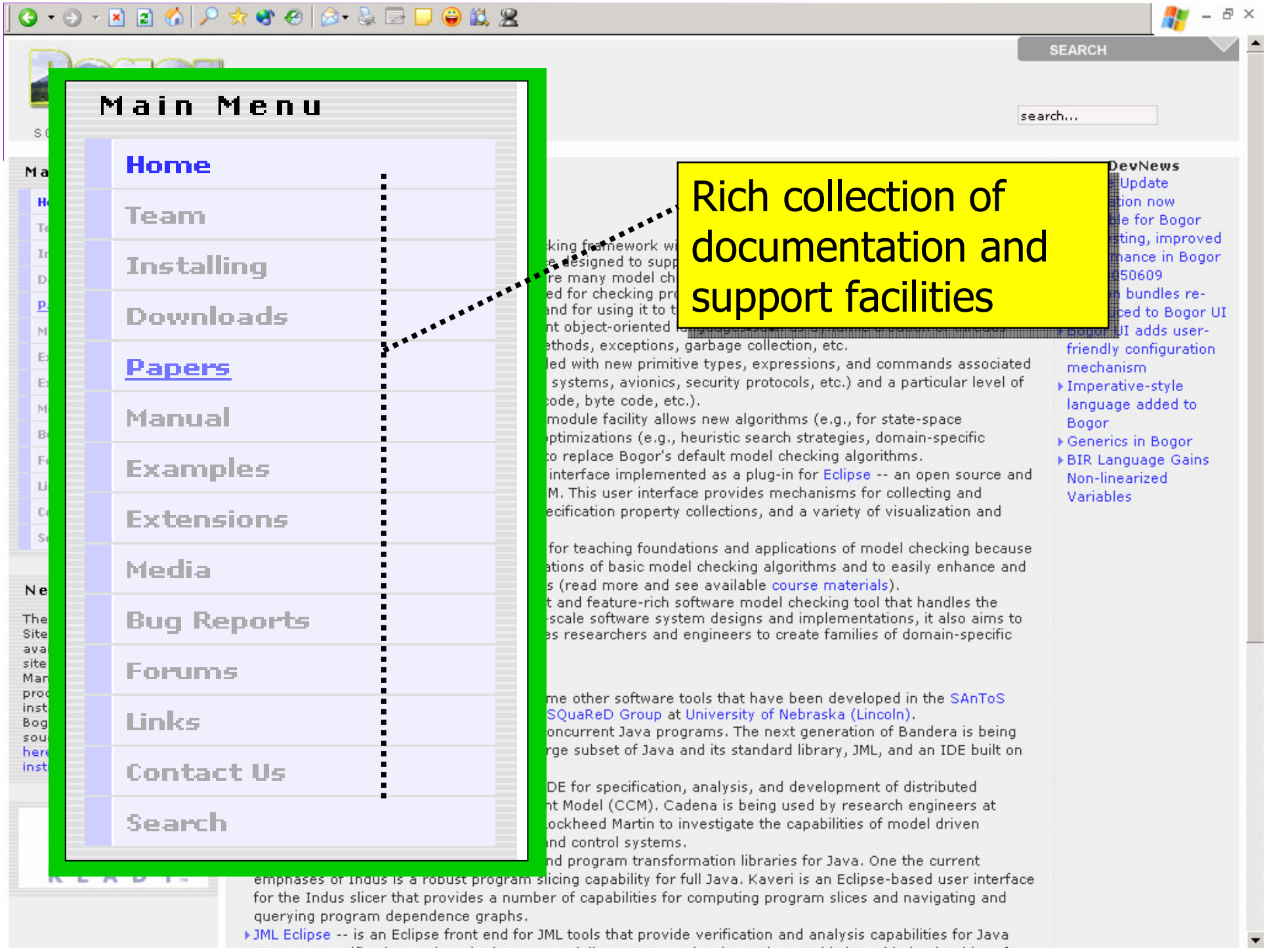
Concrete test cases are generated within the *k* and loop bounds

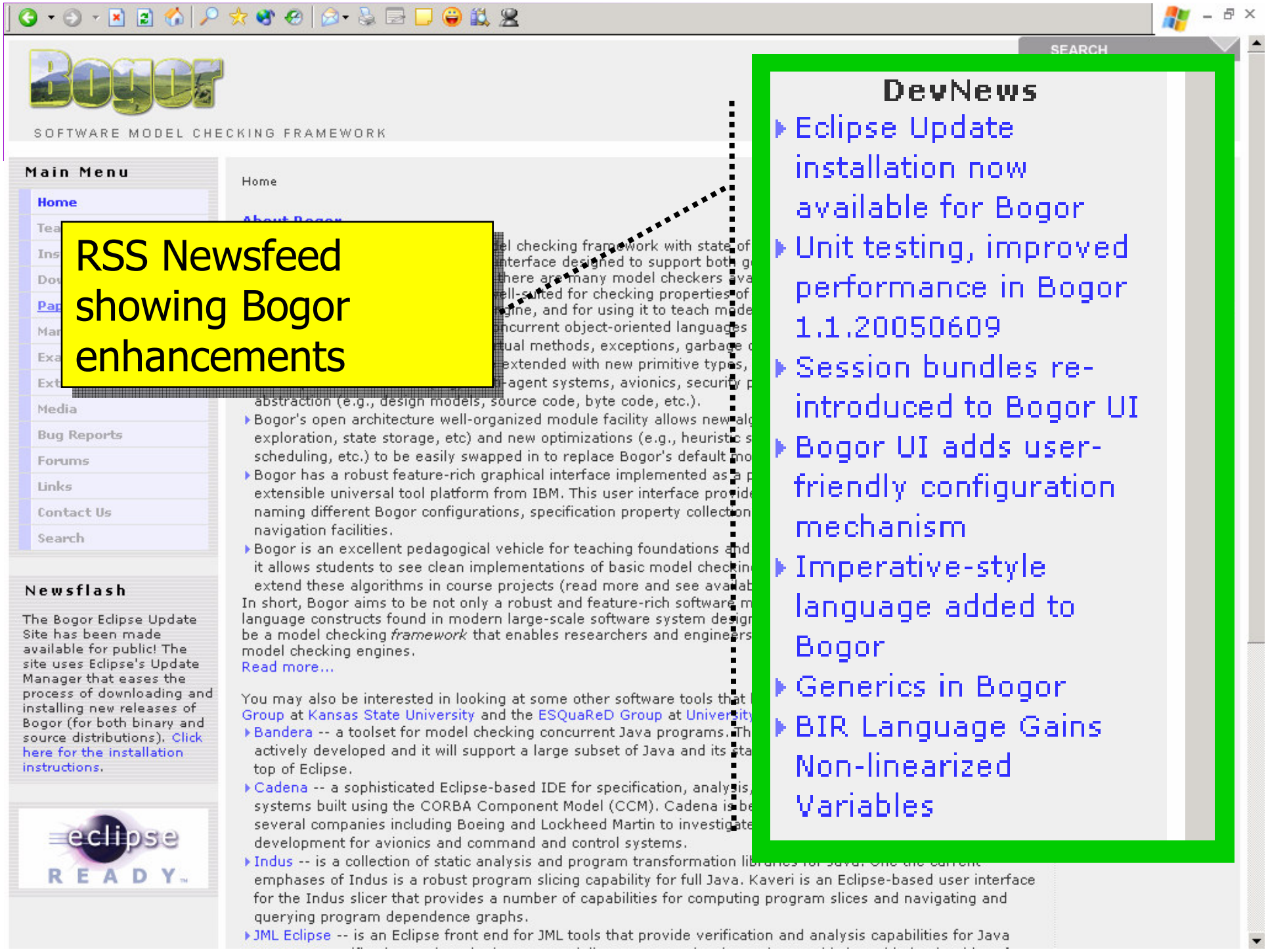
Brief Summary of Capabilities

- Static checker for common runtime errors
  - run in background for low bounds
  - run parallelizing checks at night with high bounds
- Test-case generation with complete path coverage up to bounds
  - Run in background in Eclipse, and update test suite with changes
- Gentle introduction to the inclusion of specifications (from light-weight to heavy-weight)
  - Support checking directly with controllable coverage
  - Generate tests as *evidence* for either bugs found or to illustrate coverage via a test suite
  - Argue that writing specs is easier than writing a high-coverage unit test suite – plus, specs can be leveraged in multiple ways

Bogor Online Resources

- Project website
  http://bogor.projects.cis.ksu.edu/
- Bogor User Manual
  http://bogor.projects.cis.ksu.edu/manual
- Distribution & Licensing
  - Freely downloadable from the Bogor project website
    - registration is required
  - Cannot redistribute the Bogor package, but you can redistribute Bogor extensions


Main Menu
Home
Team
Installing
Downloads
Papers
Manual
Examples
Extensions
Media
Bug Reports
Forums
Links
Contact Us
Search
Rich collection of documentation and support facilities
SEARCH
search...
DevNews
Update
ion now
e for Bogor
sting, improved
nance in Bogor
50609
bundles re-
ced to Bogor UI
UI adds user-friendly configuration mechanism
Imperative-style language added to Bogor
Generics in Bogor
BIR Language Gains Non-linearized Variables
SAnToS
SQuaReD Group at University of Nebraska (Lincoln).
JML Eclipse -- is an Eclipse front end for JML tools that provide verification and analysis capabilities for Java


SOFTWARE MODEL CHECKING FRAMEWORK
Main Menu
Home
Media
Bug Reports
Forums
Links
Contact Us
Search
Newsflash
The Bogor Eclipse Update Site has been made available for public! The site uses Eclipse's Update Manager that eases the process of downloading and installing new releases of Bogor (for both binary and source distributions). Click here for the installation instructions.
Home
RSS Newsfeed showing Bogor enhancements
DevNews
Eclipse Update installation now available for Bogor
Unit testing, improved performance in Bogor 1.1.20050609
Session bundles re-introduced to Bogor UI
Bogor UI adds user-friendly configuration mechanism
Imperative-style language added to Bogor
Generics in Bogor
BIR Language Gains Non-linearized Variables

Build Your Own Model Checker!

Bogor Web Site

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

- Significant "out of the box" capabilities for supporting modern software systems
- Novel extension and customization facilities
  - used by us
  - used by others
- Documentation and support
- Pedagogical material and courseware

Looking Toward the Future

- Synergistic integrations of
  - static analysis, dynamic analysis, type systems, theorem proving, model checking, testing
- Tools don't just hand off to each other, they accumulate coverage across techniques
- Continued use of multi-core and distributed architectures to aid scalability of model checking
- Innovative uses of state space information
  - querying, visualization, etc.