Bogor

An extensible and highly-modular model checking framework

SAnToS Laboratory, Kansas State University, USA

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

Principal Investigators

Matt Dwyer
John Hatcliff

Students

Robby

Support

US Army Research Office (ARO)
NSF CCR-SEL

Model Checking-based Analyses

Pros

- Rich-class of correctness properties
- Precise semantic reasoning
  - Relative to typical static analyses

Cons

- Scalability
- Mapping software artifact onto model checking framework

Checking Behavioral Software Models

- Requirements
  - Chan, Atlee, Heitmeyer, Chechik, Heimdahl …
- Architectural
  - Garlan, Magee & Kramer, …
- Design
  - State-machines, HERMES, Cadena, …
- Implementation
  - JPF, Bandera, SLAM, MAGIC, BLAST, …

Experience with Existing Tools

- Significant experience using existing tools
  - hand-crafted models
  - as target of translation
- Clever mappings required
  - to express artifact features, e.g., heap data
  - for efficiency, e.g., symmetry reductions
- Often times additional state variables were needed
  - to express events in state-based formalisms
  - to state properties over extent of a reference type
  - to implement scheduling policies

Custom Model Checkers

- Modifying a model checker is daunting
  - dSpin (Iosif)
  - SMV (Chan)
- Building a custom model checker is a point-solution
  - JPF, SLAM, …
- Tool-kits don't yield optimized checkers
  - NuSMV, Concurrency Factory, "The Kit", …

Bogor incorporates the advantages of each of these approaches

We would like ….

- Rich core input language for modeling dynamic and concurrent system
- Extensible input language
  - Minimizes syntactic modification when extending
  - Minimizes effort for customized semantics
- Highly-capable core checker
  - State-of-the-art reduction algorithms
- Customizable checker components
  - Eases specialization or adaptation to a particular family of software artifacts
  - Domain-experts with some knowledge of model checking can customize the checker on their own

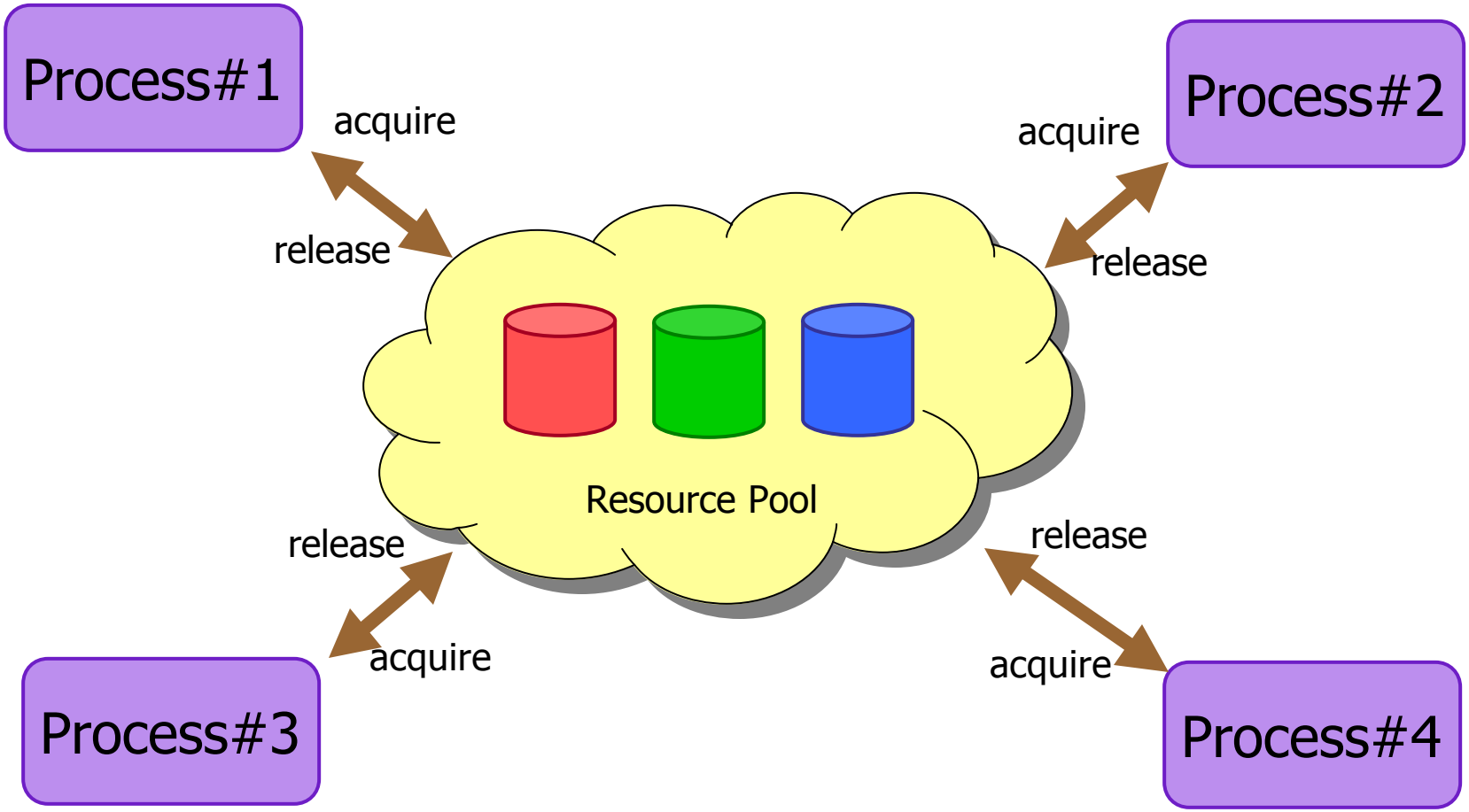
Rich Input Language

- Built on guarded-assignment language
  - Natural to model concurrent system
- Features to support dynamic, concurrent software artifacts
  - Dynamic creation of threads and objects
  - Automatic memory management (ala Java)
  - Inheritance, exceptions, (recursive) functions
  - Type-safe function pointers

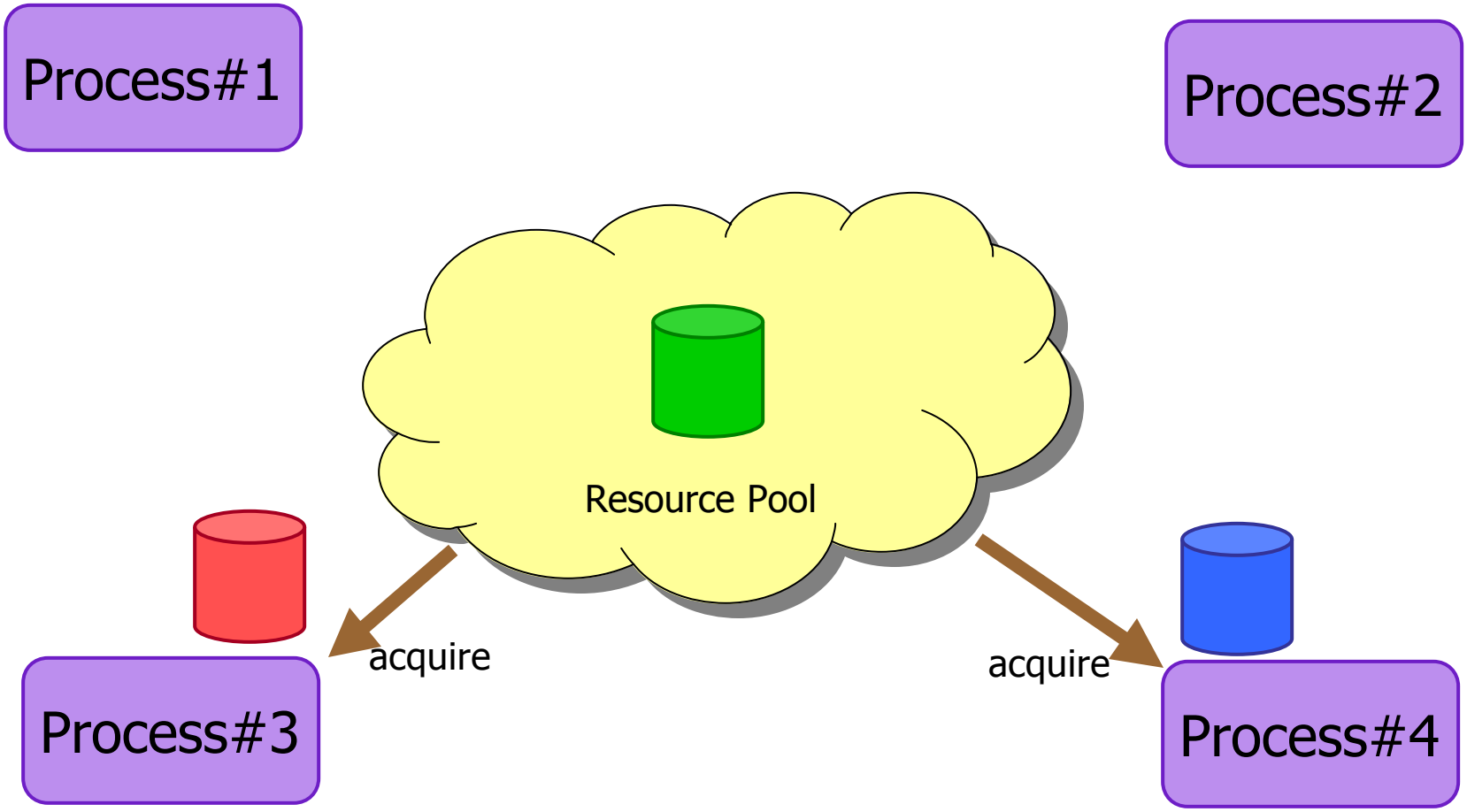
Extensible Input Language

- Allow model checker input language to directly support natural domain abstractions
- For example, `chan` in Promela
  - Hides intricate details of how a communication channel is implemented
  - Only focuses on the behavior of the channel that is relevant, e.g., synchronous, rendezvous, etc.
  - Fewer details of states that need to be stored, and possibly less states that needs to be explored
- Syntax does not need to change, and new abstractions can be added on demand

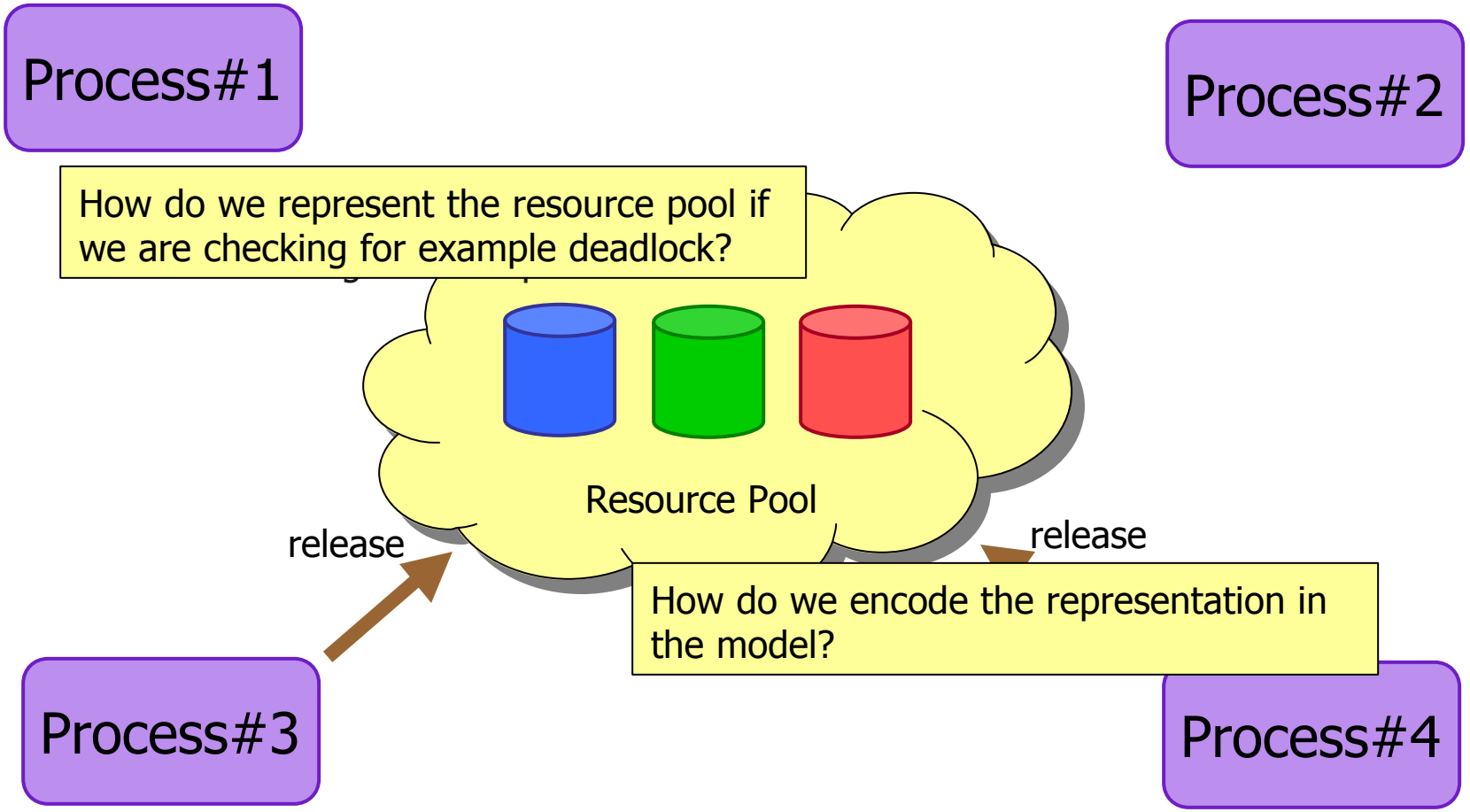
Extensible Input Language – Resource Contention Example



Extensible Input Language – Resource Contention Example



Extensible Input Language – Resource Contention Example



Extensible Input Language – Set Extension Example (Syntax)

- Bogor allows new abstract types and abstract operations as first-class construct

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

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

    expdef 'a choose<'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>);
}
```

Extensible Input Language – Set Extension Example (Semantics)

- Implementing the set value for the set type

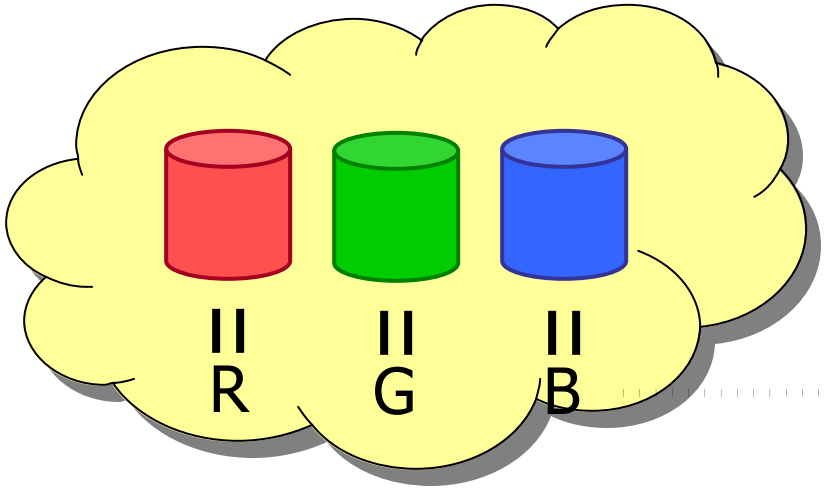
```
public class MySet implements INonPrimitiveExtValue {
 protected HashSet set = new HashSet();
 protected boole

 public void add(

 public byte[][] linearize(..., int bitsPerNPV,
   ObjectIntTable npvIdMap) {
    Object[] elements = set.toArray();
    BitBuffer bb = new BitBuffer();
    if (isNonPrimitiveElement) {
      int[] elementIds = new int[elements.length];
      for (int i = 0; i < elements.length; i++)
         elementIds[i] = npvIdMap.get(elements[i]);
      Arrays.sort(elementIds);
      for (int i = 0; i < elements.length; i++)
         bb.append(elementIds[i], bitsPerNPV);
    } else ...
    return new byte[][] { bb.toByteArray() };
 } ...
```

Constructs a bit vector that represents the set instead of encoding the HashSet instance

Reuse Java collection to implement set

Wrapper for add operation

Extensible Input Language – Set Extension Example (Semantics)



The state of the set consists of encodings of the references to resources

Suppose the references to resources are represented as integers R, G, B

<R,G,B>

And ordered!

<B,G,R>

Extensible Input Language – Observable Extension Example

- Expression/action extensions can:
  - be non-deterministic
  - have access to all the information in the current state and all previous states in the DFS stack
- Easy to express predicates/functions over dynamic data
  - e.g., non-deterministically choose a reachable heap instance from a given reference

Extensible Input Language – Observable Extension Example (Syntax)

- An invariant example using Bogor function expressions

```
fun isResourceFree(Resource resource) returns boolean =
    resource.state == ResourceState.FREE;

fun AllResourcesFreeInv() returns boolean =
    Set.forAll<Resource>(isResourceFree, resources);

...
resource := Set.choose<Resource>(resources);
...
```

- Function expressions can be recursive

Extensible Input Language – Observable Extension Example (Semantics)

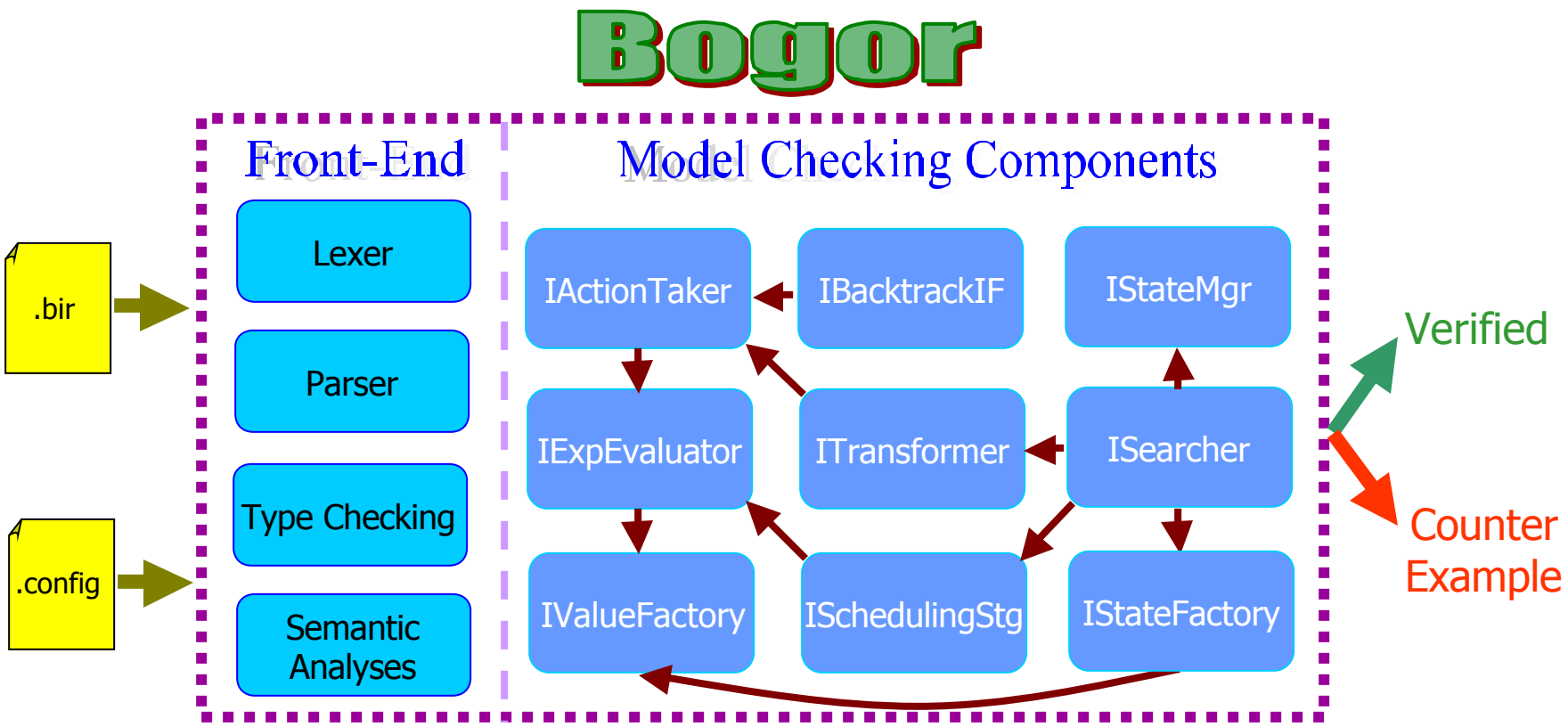
```
public IValue forAll(IExtArguments arg) {
  String predFunId = ..
  MySet set = (MySet) arg.getArgument(1);
  IValue[] els = set.elements();
  for (int i = 0; i < els.length; i++) {
    IValue val = ee.evaluateApply(predFunId,
                      new IValue[] {els[i] });
    if (isFalse(val))
      return getBooleanValue(false);
  }
  return getBooleanValue(true);
}
```

Highly Capable Core Checker

Bogor implements state-of-the-art reduction techniques

- Collapse compression [Holzmann ’97]
- Heap symmetry [Iosif ‘02]
- Thread symmetry [Bosnacki ‘02]
- Partial-order reduction [Dwyer-al ‘03]

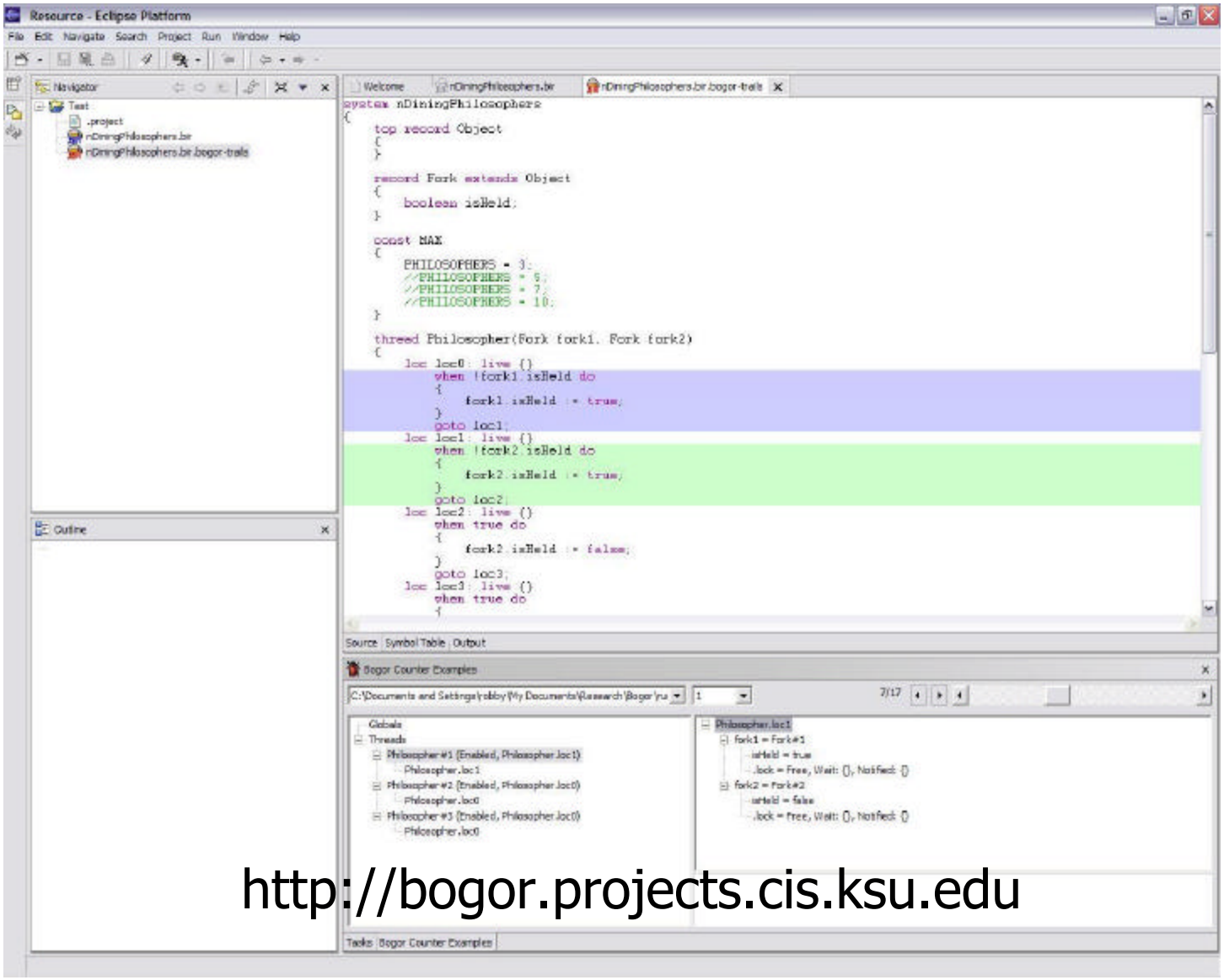
Customizable Architecture



Assessment

- Dynamic Escape Analysis for POR (150 LOC, 20x)
- Rich-forms of heap quantification
- Dynamic atomicity (5 LOC, 5x)
- Middle-ware model
- Priority Scheduling (200 LOC, 10x)
- Relative-time environment (10x)
- Lazy-time environment (240 LOC, 100x)
- Frame-bounded safety properties
- Quasi-cyclic search (200 LOC, 100x)

Tool Availability (Summer ’03)



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