Bandera

Software Model Construction for Finite-state Verification

Dave Schmidt, Matthew Dwyer, John Hatcliff	(Kansas State)
James Corbett	(Hawaii)
Hongjun Zheng, Corina Pasareanu	(Kansas State)
Shawn Laubach, Robby	(Kansas State)

Project Overview

Bandera is designed to *extract* finite-state models of Java code

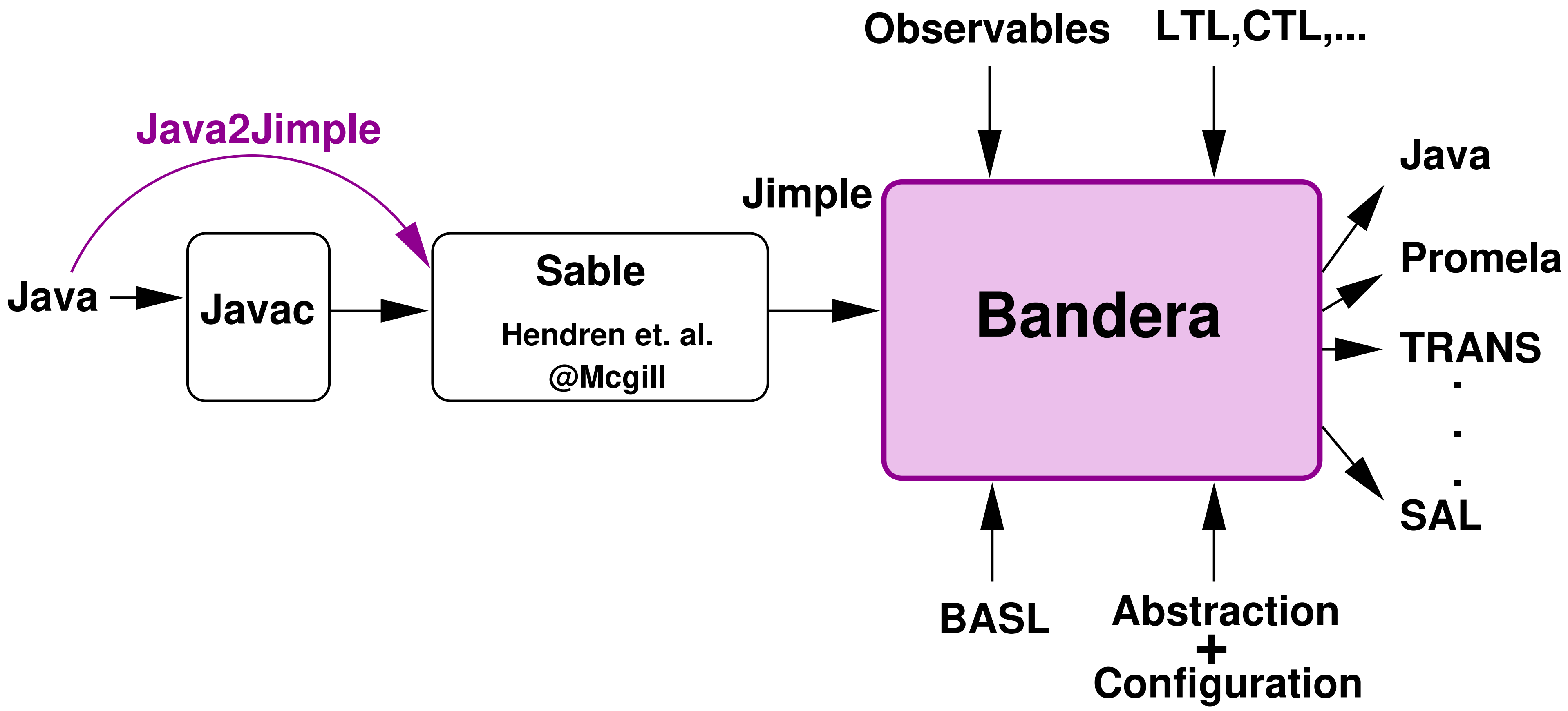
Extraction as opposed to translation

- property driven extraction
- model per property

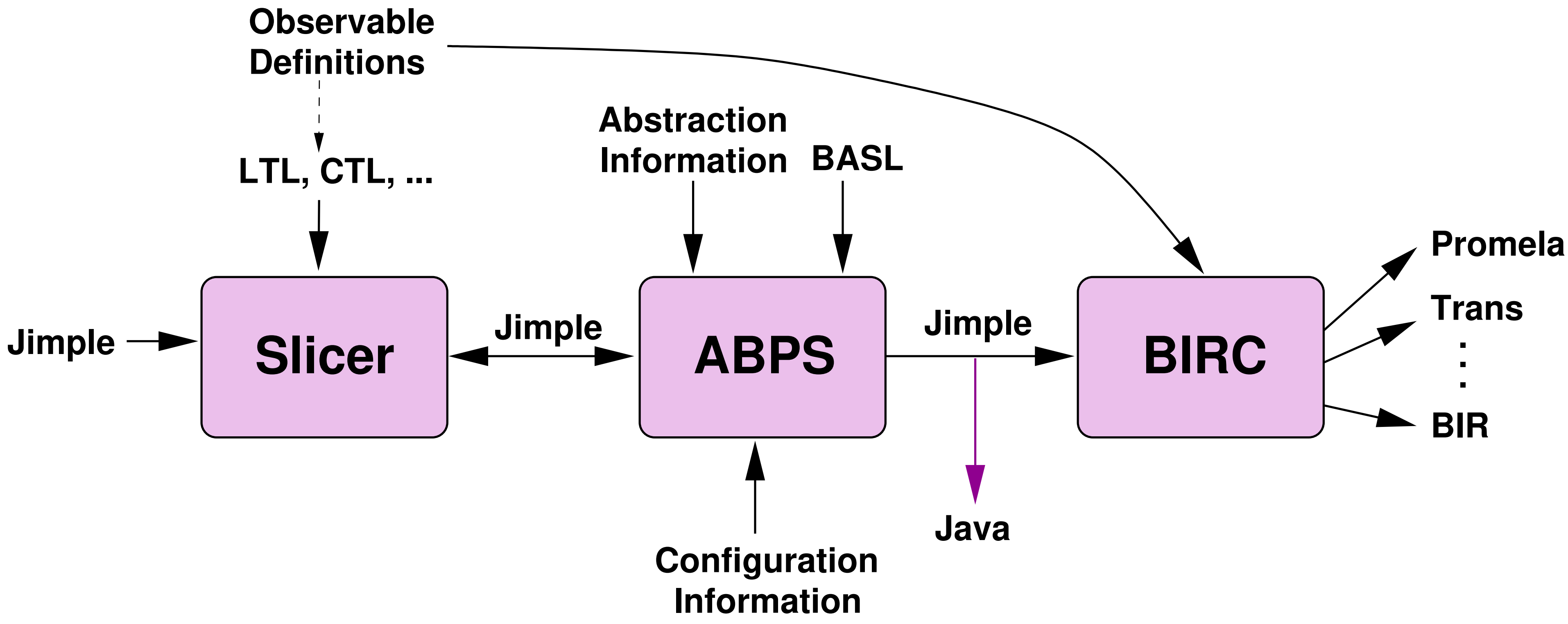
Open architecture of tools and components

- incorporate new analysis techniques
- new combinations of existing techniques

Bandera Toolset



Major Bandera Components



Bandera Interfaces

Jimple 3-address code with JVM primitives

Observables define states or actions of interest

LTL,CTL,... define correctness properties over observables

BASL defines abstract interpretation (A.I.)

Abstraction Info bind and control A.I. for program variables

Configuration Info restrict extent of object types

Promela,SMV,... target verifier input languages

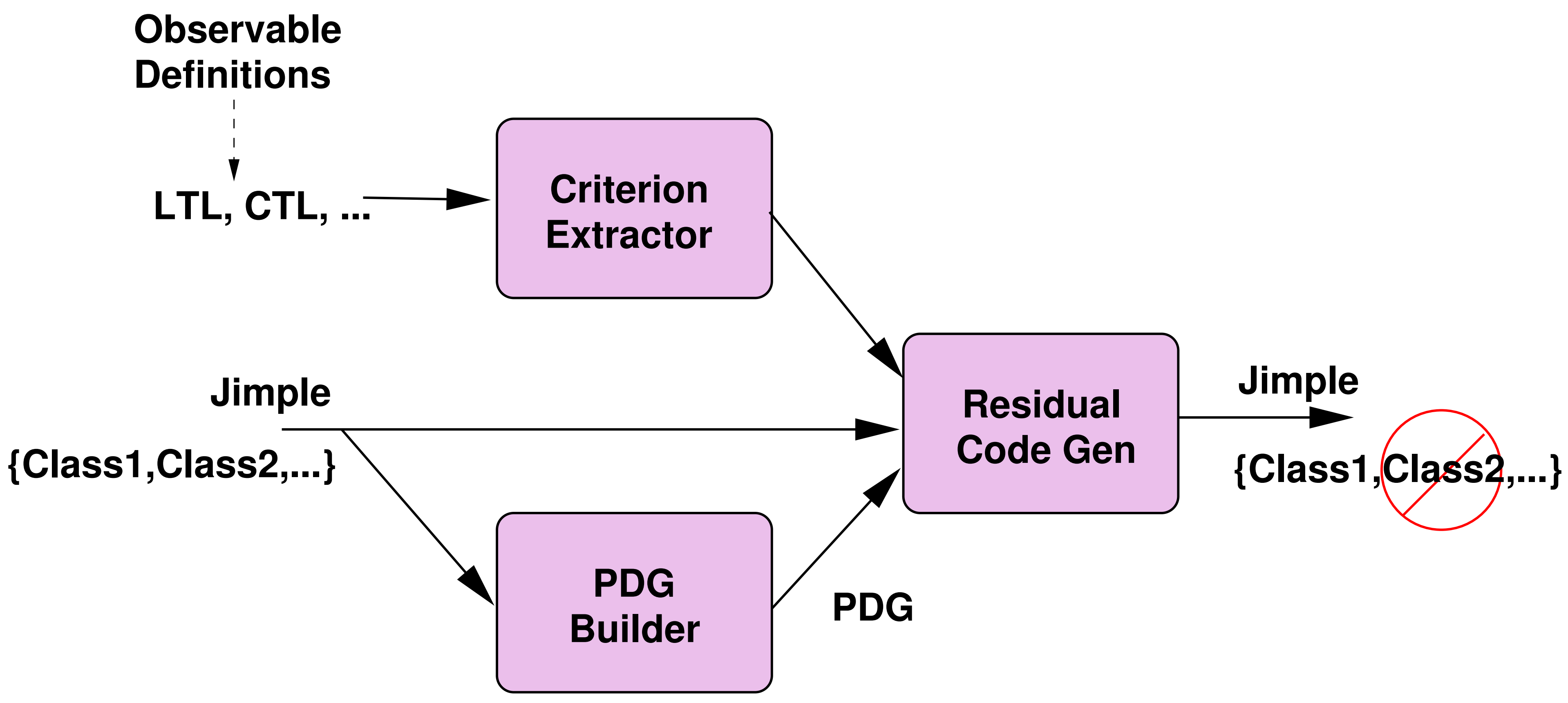
Slicing

Has attracted some attention recently for model extraction.

An *obvious* way to safely eliminate irrelevant parts of system, but

- **want to slice *multi-threaded* Java**
- **i.e., eliminate threads if possible**
- **want slicing criterion to insure preservation of TL property**
- **i.e., form an upper-bound on abstraction**

Bandera Slicer



Specialization

A mechanism for incorporating abstract interpretations into models

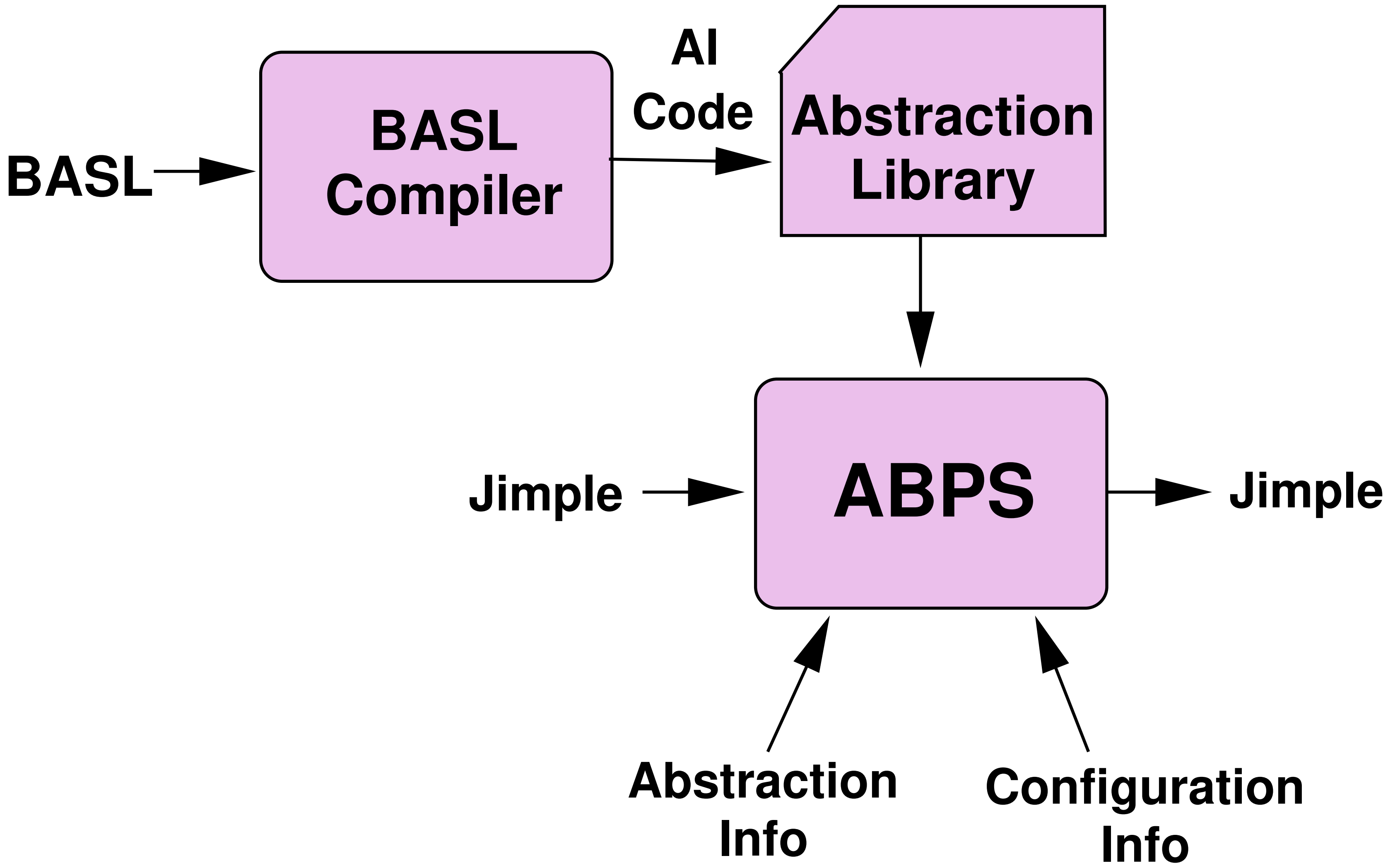
More than just substitution of abstractions

- simplify model based on abstract constant propagation, ...
- shift dynamically created objects to compile-time

Need a flexible description of the abstractions

- interpretation and code generation
- Bandera Abstraction Specification Language (BASL)

Bandera Specializer



Example : BASL

```
abstraction Signs extend Int{
  TOKENS = {NEG, ZERO, POS};
  LIFT = {NEG, ZERO, POS};

  abstract {
       (*n < 0*) -> NEG;
       (*n == 0*) -> ZERO;
       (*n > 0*) -> POS;
  }
```

```
operator add + {
       (POS, POS) -> POS;
       (NEG, NEG) -> NEG;
       (ZERO, _) -> op2;
       (_, ZERO) -> op1;
       (_, _) -> T;
}
```

Model Constructor

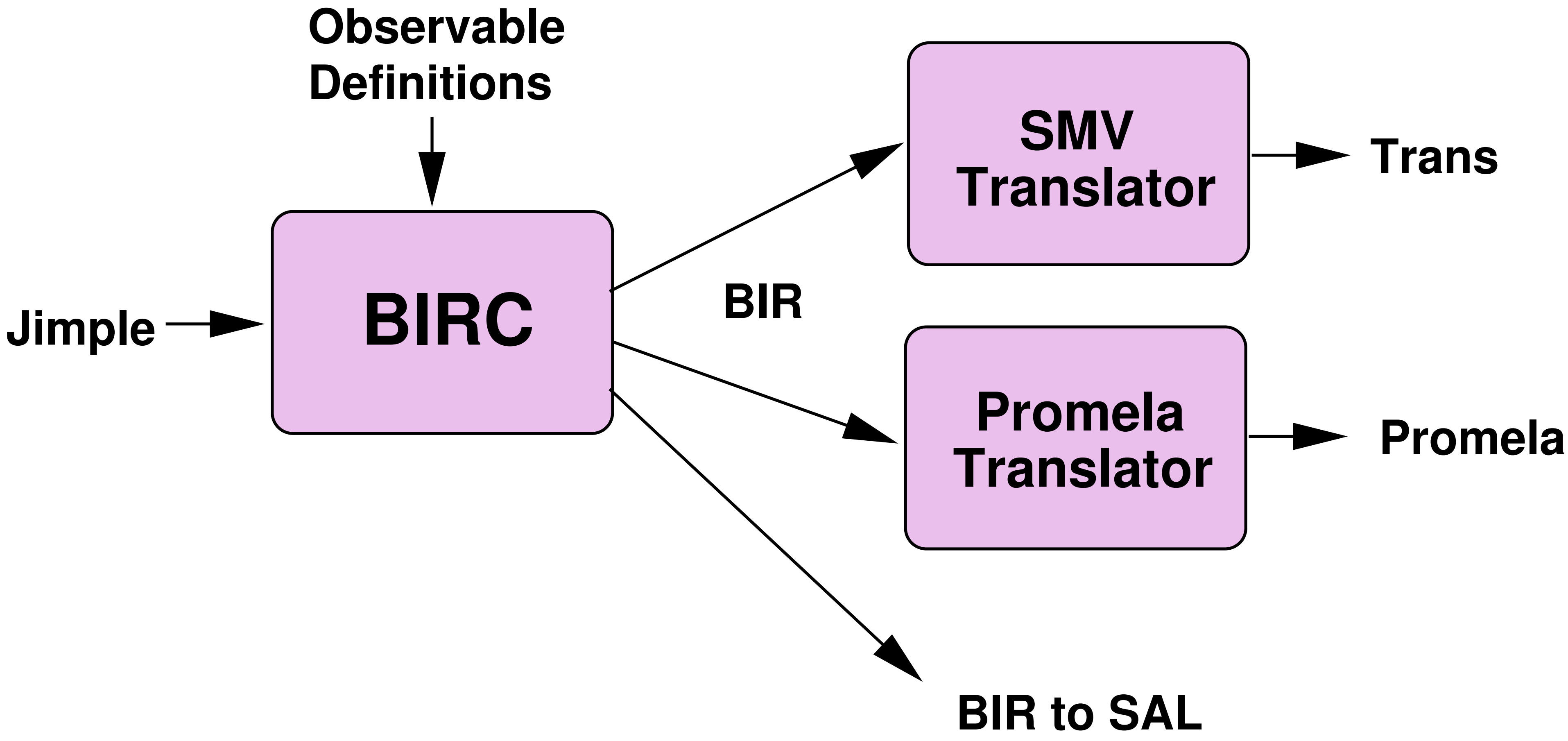
Targets multiple verification backends

- uses JVM-specific guarded-transition system (BIR)
- reductions applied to intermediate form
- compact model generators (550, 800 SLOC)

Embeds observables to make them visible to model checkers

- supports both state and action observables
- observables are included in counter-examples

Bandera Model Constructor



Current Status

Initial working prototypes for

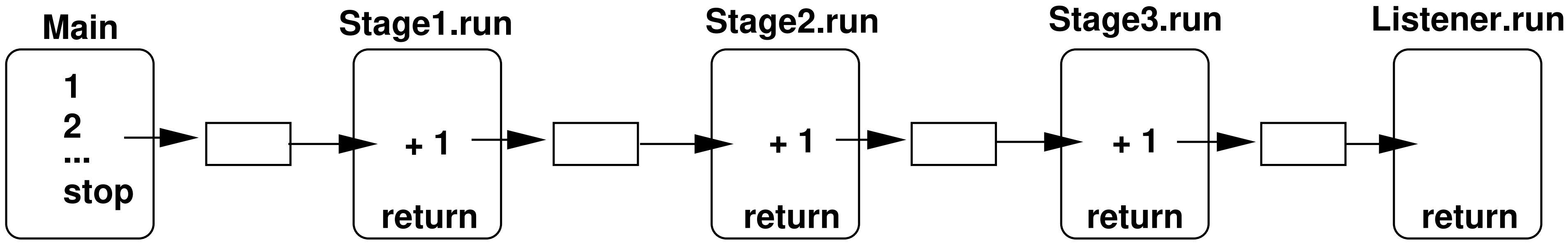
- Slicer (multiple threads, no method calls)
- Specializer (compiles A.I.s, inlines, 6 A.I.s)
- BIRC (SPIN, SMV)

There are papers or documentation for each of these.

Non-trivial Java programs have been processed

- Readers/Writers Problem
- Multi-threaded Pipeline

Pipeline Example



Pipeline Properties

Response Property

- “When stop is issued Stage$_i$ returns”
- $\Box(\textbf{stop} \rightarrow \Diamond\ \textbf{return}_i)$

Precedence Property

- “Stage$_i$ returns only after a stop is issued”
- $\Diamond\ \textbf{stop} \rightarrow (\neg\ \textbf{return}_i\ \mathcal{U}(\ \textbf{stop} \wedge \neg\ \textbf{return}_i))$

Effects of Slicing and Abstraction

Problem	Threads	Vars	Time (s)	Space (states)
Pipeline/Response				
Original	5	24	120	702229
Slice(4)	5	20	116	702229
Slice(3)	4	16	9	74380
Slice(2)	3	11	.5	5791
Slice(1)	2	6	.04	195
Slice+Signs(4)	5	20	65	399949
Slice+Signs(3)	4	16	4.4	39873
Slice+Signs(2)	3	11	.2	3013
Slice+Signs(1)	2	6	.03	190
Pipeline/Precedence				
Original	5	24	14	169044
Slice+Signs	2	6	.03	123
ReadersWriters/Absence				
Original	5	20	3.1	71104
Slice+Signs	5	11	.34	5887

Ongoing Work

- Java->Jimple->Java (bypass bytecodes, residualize Java)
- Inter-thread value propagation in ABPS
- Inter-method slicing
- Porting of environment generation tools from Ada
- Enhancing abstraction library (e.g., A.I.s for `java.util.Vector`)