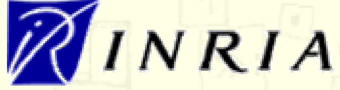
Meeting GECCOO Project
6-7 may 2004

Symbolic animation of JML specifications

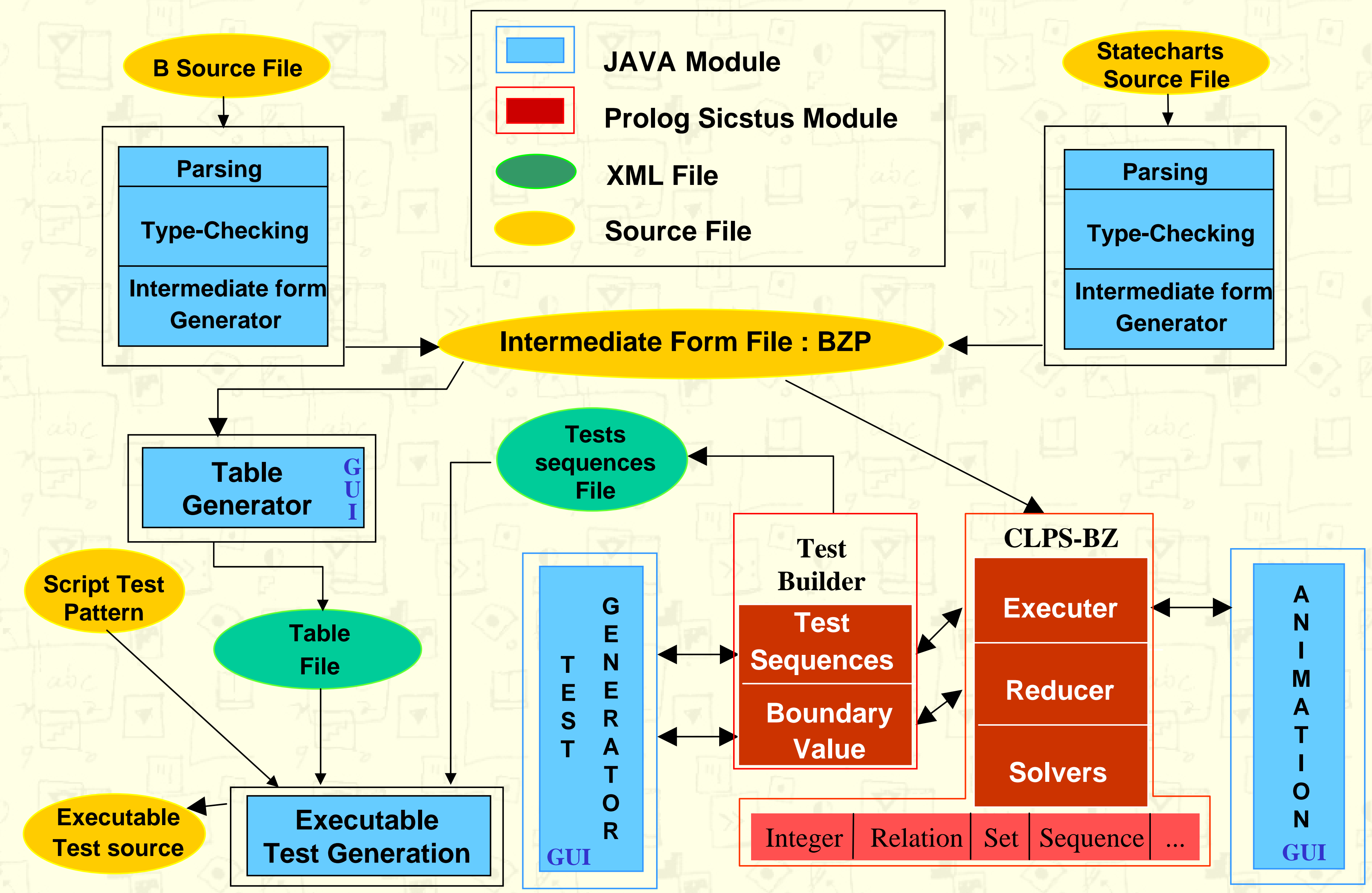
Frédéric Dadeau



Introduction

- BZ-Testing-Tools: symbolic animator and model-based test generator based on constraint solving technologies
- Work: symbolic animation of JML specification, based on the model described in the JML annotations
- Motivation: use constrained animation to simulate the execution of the formal model and ensure its validity

Architecture



Outline

- The BZP(E) format
- Symbolic animation
- Translation of JML specifications to BZPE
- Checking properties
- The animator
- Future work

Outline

- The BZP(E) format
- Symbolic animation
- Translation of JML specifications to BZPE
- Checking properties
- The animator
- Future work

The BZP(E) format

- Intermediate format of the BZ-Testing-Tools environment
- Pre / post-condition format
- Animation realized through the PROLOG module « executer » based on the CLPS-BZ solver
- Uses finite data structures

The BZP(E) format

- `PROLOG` syntax:

  specification(*spec-name*).

  declaration(*spec-name*, data-kind, *data-name*, data-type).

  operation(*spec-name*, *operation-name*).

  predicat(*spec-name*, pred-kind, *pred-id*, *predicate*)

data-kind : static | variable | input(*op-name*) | output(*op-name*) | local(*op-name*)

data-type : atom | int | set(data-type) | pair(data-type,data-type)

pred-kind : static | invariant | initialisation | pre(*op-name*) | post(*op-name*)

The BZP(E) format

Semantics

- Use of a constraints store containing the set of variables values and their constraints

- Initialization
  - Add the « initialisation » predicate to the store

- Execution of an operation
  - Decomposition of the operation in behaviors (rewriting of disjunctions) ;
  - Parallel adding of different kind of predicates (precondition, postconditon, invariant) to the store.

The BZP(E) format

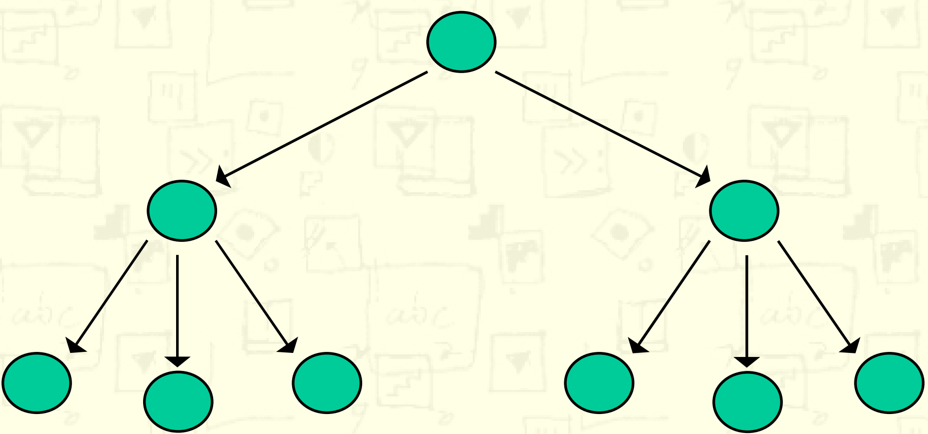
- From BZP … to BZPE
  - BZPE is an extension of BZP
  - Allows multiple specifications declaration
  - References to data across specifications
  - Allows translation of object concepts (ex. UML/OCL)

Outline

- The BZP(E) format
- Symbolic animation
- Translation of JML specifications to BZPE
- Checking properties
- The animator
- Future work

Animation

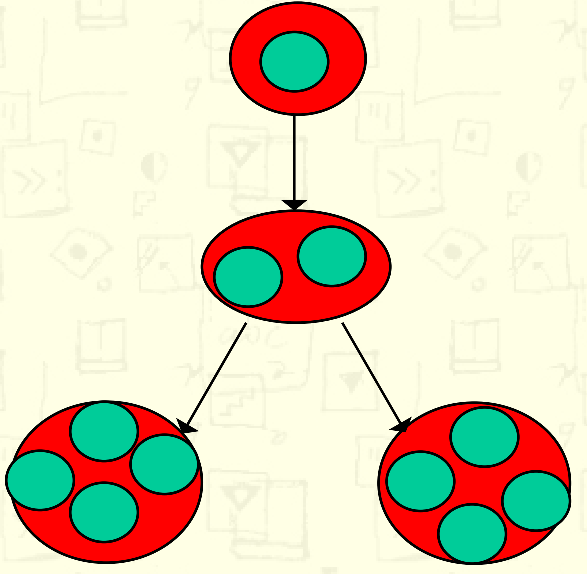
- « Classical » animation:
  - Run through of a reachability graph
  - Each node represents a valued state
  - Each arc represents an activable behavior



Animation

Symbolic animation:

- Run through a constraint reachability graph
- Node are constraints stores
- A constraints store represents a finite set of valued states



Animation

What is a constraint store?

- Set of predicates giving informations on a variable's domain
- Each variable has a domain:

  $x \in [-2^{15}, 2^{15}]$
- Each predicate added to the store reduces the variable domain:

  $x \in [-2^{15}, 2^{15}] \wedge x > 0 \rightarrow x \in [1, 2^{15}]$
- If a predicates reduces a domain to the empty set, it becomes inconsistent:

  $x \in [1, 2^{15}] \wedge x = 0 \rightarrow x \in [\,] \rightarrow$ inconsistency!

Animation

- How to create constrained states?
  - With an unvalued initial state
  - By leaving operation’s input parameters unvalued

Animation

- Advantages of symbolic animation:
  - Dramatical decrease of the reachability graph size
  - Propagation of the non-determinism of data
  - The user is free to valuate input parameters or not

Outline

- The BZP(E) format
- Symbolic animation
- Translation of JML specifications to BZPE
- Checking properties
- The animator
- Future work

Translation of JML specifications into BZPE

Translation of class structure

- Class ➔ BZPE specification
  - With a set of possible instances (constant)
  - With a set of created instances (variable)
- Class attribute ➔ BZPE variable
  - A total function mapping the set of created instances to the value
- Class method ➔ BZPE operation
  - The class instance is an input parameter
  - Each method parameter ➔ A BZPE operation parameter

Translation of JML specifications into BZPE

Translation of JML clauses

- Creation of new BZPE kind of predicates
  - jml_invariant ;
  - jml_constraint ;
  - jml_when (*operation*)

  interpreted by a specific « executer » ;

- Requires clause ➔ BZPE operation pre-condition

- Ensures clause ➔ BZPE operation post-condition

Translation of JML specifications into BZPE

```
public class date

  int day ;
  int month ;
  int year ;

  //@ invariant day > 0 &&
  //@           day < 32;

  /*@ public behavior
    @ ensures
    @  \result == \old(day);
    @*/

  public int getDay()  {

     return day;

  }
```

specification(date).
declaration(date, static, all_instances, set(atom)).
predicat(date, static, 1, all_instances = {d1,...,dN}).
declaration(date, variable, instances, set(atom)).
predicat(date, invariant, 2, instances ⊆ all_instances).
predicat(data, initialisation, 3, instances = { }).

declaration(date, variable, day, set(pair(atom,int))).
predicat(date, invariant, 4, day ∈ instances → IntRan)

predicat(date, jml_invariant, 5, ∀x. x ∈ instances ⇒
day(x) > 0 & day(x) < 32).

operation(date,getDay).
declaration(date, input(getDay), this, atom).
declaration(date, output(getDay), result, int).
predicat(date, pre(getDay), 3, this ∈ instances).
predicat(date, post(getDay), 4, result = day(this)).

Outline

- The BZP(E) format
- Symbolic animation
- Translation of JML specifications to BZPE
- Checking properties
- The animator
- Future work

Properties checking

- How to check a **satisfiable** predicate:
  - A constraint store represents a constrained state
  - Addition of the predicate to the store:
    - If the store is still consistent: the predicate is satisfiable
    - If the store becomes inconsistent: the predicate is unsatisfiable
      A valuation of the store presents a counter-example

- Properties we want to be **satisfiables**:
  - Pre-conditions (requires clause)
  - Non divergence (diverges clause)
  - Method activation after delaying (when clause)

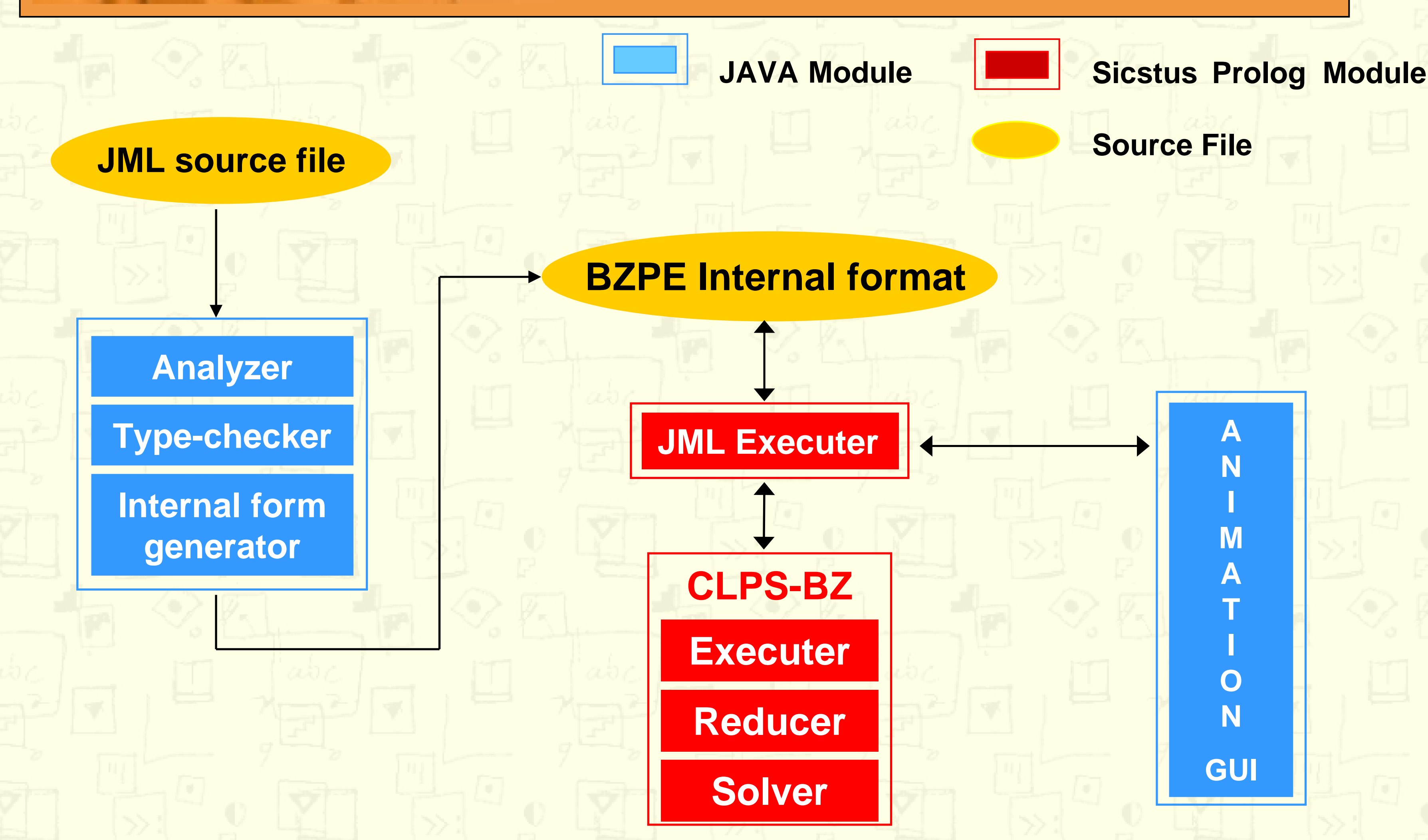
Properties checking

- How to check a **valid** predicate:
  - A constraint store represents a constrained state
  - Addition of the predicates negation to the store:
    - If the store is still consistent: the predicate is not valid
      A valuation of the store presents a counter-example
    - If the store becomes inconsistent: the predicate is valid
- Properties we want to be **valid**:
  - Invariant
  - History constraints

Outline

- The BZP(E) format
- Symbolic animation
- Translation of JML specifications to BZPE
- Checking properties
- The animator
- Future work

The JML animator



The JML animator

- Considers the model described by the JML annotations

- Execution capabilities:
  - Valued or constrained animation ;
  - Activation of behaviors ;
  - Behaviors consistency ;
  - Method divergence avoiding ;
  - Method delaying ;

- On-the-fly properties checking:
  - Class invariant;
  - History constraints;

The JML animator

- Supported JML data types:
  - integers (byte, short, int, char)
  - arrays (multi-dimensionals)
  - objects
- Supported JML clauses:
  - requires
  - ensures
  - diverges
  - when
  - invariant
  - constraints
  - initially

Outline

- The BZP(E) format
- Symbolic animation
- Translation of JML specifications to BZPE
- Checking properties
- The animator
- Future work

Future work

- Improving animator:
  - Increasing JML support
    - Inheritance
    - Multiple specification files
    - Floats (waiting for solver)
  - Translation
    - Method calls
  - Type checking
  - Improving GUI

Future work

- From the constrained animation on:
  - Manual / automatic simulation of the execution of a JML specification
  - On-the-fly properties checking with reachables counter-examples production
  - (Boundary) test cases generation
  - Modelling errors detection
    - Too weak preconditions
    - Too strong / too weak invariant
    - …