A semantic framework for symbolic execution in Maude

WRLA 2026

Rafael Morales-Palacios, Rubén Rubio, and Ignacio Fábregas

Universidad Complutense de Madrid (UCM)

Program verification

Semantics specification

Maude

Symbolic reasoning

Maude SMT and narrowing

Symbolic execution
Concolic testing

Semantics transformation

Introduction

- Executable specification language based on rewriting logic
- Organized in modules
  - Functional modules $\Longrightarrow$ Membership equational specifications
  - System modules $\Longrightarrow$ Rewriting logic specifications
- `META-LEVEL` $\Longrightarrow$ Terms and modules as terms of universal logic

```
mod WHILE-MAUDE is
  protecting CONCRETE-FW .
  sorts Inst Prog .
  subsort Inst < Prog .

  op skip : -> Inst [ctor] .
  op _;_  : Prog Prog -> Prog [ctor assoc id: skip] .
  op if_then{_}else{_} : BExp Prog Prog -> Inst [ctor] .
  op if_then{_}        : BExp Prog -> Inst .
  *** [...]
  op <_|_> : Prog Stores -> State [ctor] .
```

```
vars P P1 P2 : Prog . var STR : Stores .
var  Q       : Qid  . var B   : BExp   .
 rl [assign-B] : < bv(Q) := B ; P | STR >
              => < P | insert(Q, eval(B, STR), STR) > .

crl [if-then] : < if B then {P1} else {P2} ; P | STR >
             => < P1 ; P | STR > if eval(B, STR) = true .

crl [if-else] : < if B then {P1} else {P2} ; P | STR >
              => < P2 ; P | STR >
              if not eval(B, STR) = true .
```

Related work

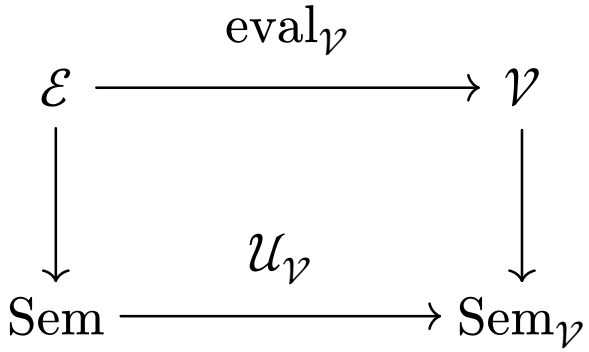
Related work

- Symbolic execution
  - Clang compiler, KLEE
  - K Framework
    - Based on semantics specification
- Maude supports narrowing and SMT
  - Maude-NPA
  - MaudeSE
    - Rewriting modulo SMT $\Longrightarrow$ Symbolic execution

Infrastructure

Overview

- Transformation description
  - From concrete setting to symbolic and concolic settings



Concrete setting

- `Int`, `Rat`, and `Bool` values and variables $\Longrightarrow$ `IExp`, `RExp`, and `BExp`
- `Map` $\Longrightarrow$ Variable stores
- `eval` function

```
op eval : BExp Stores -> Bool .
op eval : IExp IStore -> Int .
op eval : RExp RStore -> Rat .
```

Symbolic execution

Symbolic execution

- Given a program with symbolic variables
  - Branch on conditionals ⇒ Explores reachable execution paths

Absolute value

```
if (x > 0) then {
  y = x; // Constraint: x > 0
} else {
  y = -x; // Constraint: not x > 0
}
```

Rewriting modulo SMT

- Let:
  - $\mathcal{T}$ be the SMT theory
  - φ the accumulated constraints at some point in execution

Given a constrained term (t, φ), we can apply a rule $l \to r \text{ if } \psi$ when there exists a matching substitution θ, such that $\varphi \wedge \theta(\psi)$ is satisfiable in $\mathcal{T}$.

And the result is $(\theta(r), \varphi \wedge \theta(\psi))$.

Translation for symbolic execution

- Translation for MaudeSE
  - Translate `eval` and variable stores in specification

```
crl S => S'
  if evalevalS(e, STRSTRS) = (true).BoolBoolean .
```

- Direct translation:

```
crl S {C} => S {C'}
  if C' := C and evalS(e, STR')
  /\ metaCheck(['REAL-INTEGER], upTerm(C')) .
```

Symbolic execution example

- Symbolic execution as a Maude `search`:
  - Equivalent to `smt-search` in MaudeSE

Absolute value (P)

```
1 if x > 0 then {
2   y = x;
3 } else {
4   y = - x;
5 }
```

Maude

Pattern

```
Symbolic State: < nil | y |-> Y:Integer, IS:IStoreS | ... >
                s.t. Y:Integer < 0
```

Maude

Initial state

```
Symbolic State: < P | 'x |-> X:Integer > {true}
```

Maude

Step 1 [if-thenS]

```
Symbolic State: < y = x; | 'x |-> X:Integer >
                {true and X:Integer > 0}
```

Maude

Step 1′ [if-elseS]

```
Symbolic State: < y := - x; | 'x |-> X:Integer >
                {true and not X:Integer > 0}
```

Maude

Step 2 [assign-IS]

```
Symbolic State: < nil | 'y |-> X:Integer, ... >
                {true and X:Integer > 0}
```

Maude

Step 2′ [assign-IS]

```
Symbolic State: < nil | 'y |-> - X:Integer, ... >
                {true and not X:Integer > 0}
```

Maude

Result

```
Maude> No solution
```

Symbolic execution via narrowing

- Narrowing ad hoc transformation
- `IntFVP` and `BoolFVP` $\Longrightarrow$ `IExp` and `BExp`
  - Presburger arithmetic
- Theory with unconditional equations and finite variant property
- Transform conditional rules with `eval` conditions to:

```
rl S => < S | evalS(e, STR') > [narrowing] .
rl < S | B:BoolFVP > => S' [narrowing] .
```

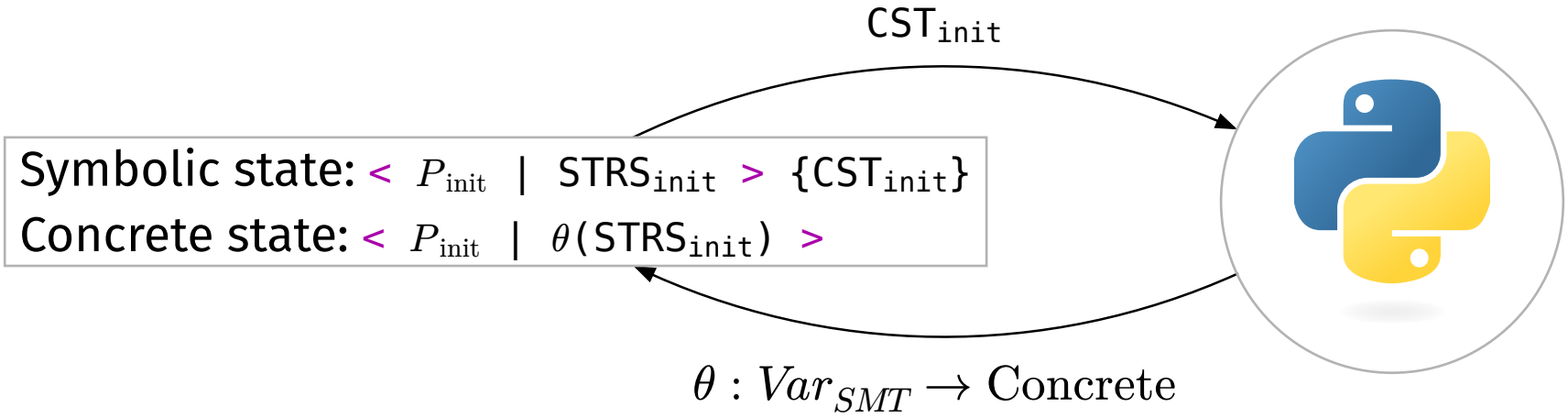
Concolic testing

Concolic testing

- Loops $\Longrightarrow$ symbolic execution may not end
  - Concolic testing
- Dual state concrete/symbolic
  - Concrete: Execute concretely
  - Symbolic: Explore, accumulate path constraints
- Restart method $\Longrightarrow$ Initialize concrete state with values satisfying constraints
  - Python hook to retrieve model

Translation for concolic testing

Init(P_{init}, $\text{STRS}_{\text{init}}$, CST_{init}):



- P_{init} and $\text{STRS}_{\text{init}}$ are stored in a third part of the state

Unconditional rules:

Concrete

```
< bv(Q) := B ; P | STR >
```

↓

```
< P | Q |-> eval(B, STR), STR >
```

Symbolic

```
< bv(Q) := B ; P | STRS > {CST}
```

↓

```
< P | Q |-> evalS(B, STRS), STRS > {CST}
```

Conditional rules `crl` $t \to t'$ `if eval(B, STR) = true`

- Example with rule `[if-then]`
  - Concrete path exploration

Concrete

```
< if B then {P1} else {P2} ; P | STR >
```

↓

```
< P1 ; P | STR >
```

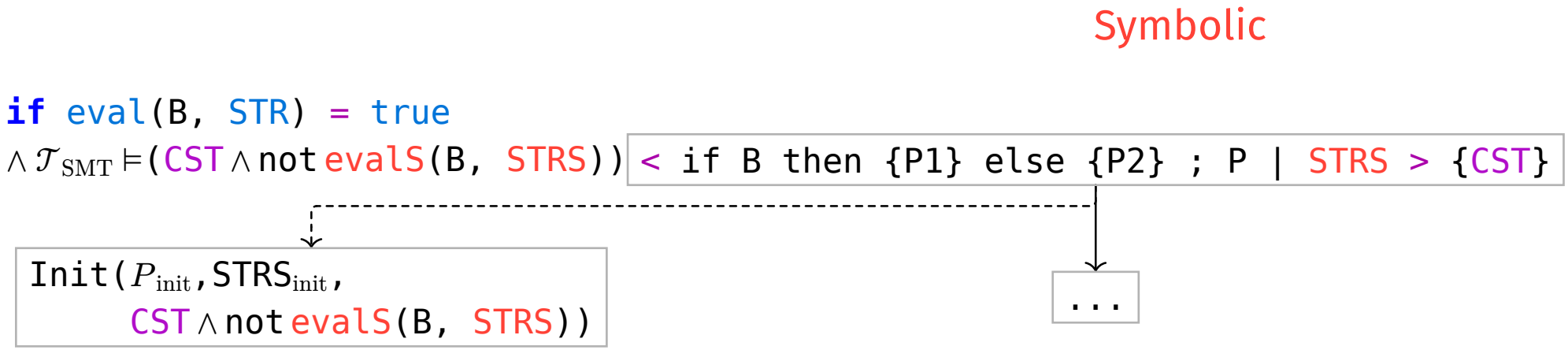
Symbolic

```
< if B then {P1} else {P2} ; P | STRS > {CST}
```

↓

```
< P1 ; P | STRS > {CST ∧ evalS(B, STRS)}
```

- Spawn a new execution with another rule
  - Explore the remaining concrete paths



Concolic testing example

- Concolic testing with loops as a Maude `search`:

Division by 0 bug (P_1)

```
1 int i = 5;
2 while (k < y) do {
3   if (k > 10) then {
4       x = k * y
5       a = x - i
6       z = y / a;} // Error!!
7   k += 1;
8   i *= 2;}
```

Maude

Pattern

```
Concrete State: < Var := IExp / iv('a) ; P' | 'a |-> 0, IS:IStore | ... >
Symbolic State: S':State {CST:Boolean}
```

Maude

Initial state

```
Concrete State: < P1 | 'k |-> 0, 'y |-> 0 >
Symbolic State: < P1 | 'k |-> K:Integer, 'y |-> Y:Integer > {true}
```

=> [assign-I] => [while-exit-bis] =>

Maude

Step 2 (First restart)

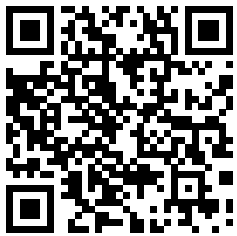
```
Concrete State: < P1 | 'k |-> 0, 'y |-> 1 >
Symbolic State: < P1 | 'k |-> K:Integer, 'y |-> Y:Integer >
                {true and K:Integer < Y:Integer}
```

- It restarts 11 times and finds a solution when $\mathtt{k} \mapsto 9$, $\mathtt{y} \mapsto 20$ after 7 iterations

Conclusions and future work

Conclusions

- Generic framework for symbolic execution and concolic testing
- Based on the semantics specification in Maude
- Tested with illustrative examples
- Code and examples publicly available



https://github.com/rafamor-exe/Symbolic_Execution_in_Maude

Future work

- Implement the narrowing transformation in Maude ✓
  - Extend with narrowing modulo SMT
- More complex datatypes and structures with narrowing
- Realistic programming languages
  - WebAssembly, VHDL

Questions?