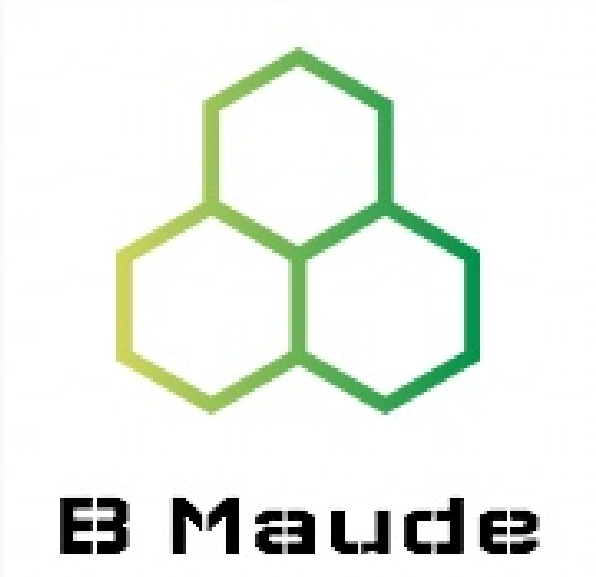
BMaude

Christiano Braga

May 12, 2018

Universidade Federal Fluminense



- Presentation automatically generated with BMaude and Pandoc:

```
beamer: ./bmaude demo-md | sed '1d' | pandoc --highlight-style=kate -H
    doc/make-code-scriptsize.tex --wrap=preserve -V theme:metropolis -V
    logo:bmaude.jpg -V institute:"Universidade Federal Fluminense" -V
    address:"http://www.ic.uff.br/~cbraga" -fmarkdown-implicit_figures -t
    beamer -o doc/bmaude-demo.pdf
epub: ./bmaude demo-md | sed '1d' | pandoc --highlight-style=tango -H
    doc/make-code-scriptsize.tex -V theme:metropolis -V logo:bmaude.jpg -V
    institute:"Universidade Federal Fluminense" -V
    address:"http://www.ic.uff.br/~cbraga" -fmarkdown-implicit_figures -o
    doc/bmaude-demo.epub
```

Welcome!

Hi, welcome to the BMaude Prototype presentation!

- This demo is for version BMaude Prototype (Uruçu-Amarela February 2018)
  - Uruçu-amarela is a kind of bee common in the Rio de Janeiro area.
- In this presentation, we illustrate the subset of the Abstract Machine Notation currently supported by the tool and the available verification techniques.
- A machine declares variables, constants,their values and operations. An operation can be any composition of generalized substitutions.
- The verification techniques currently supported by BMaude are execution by rewriting, symbolic execution by narrowing, state space search, and Linear Temporal Logic model checking.

Iterative factorial example

- So let us begin by loading a machine enconding the factorial function with a loop.

```
BMaude: Machine FACT loaded.
MACHINE FACT
 VARIABLES y
 VALUES y = 1
 OPERATIONS
  fact(x)=
   WHILE ~(0 == x) DO y := y * x ; x := x - 1 ; print(y)
END
```

Iterative factorial example (cont.)

- Now we can execute a call to factorial of 100, for instance, ...

```
exec fact(100)
rewrites: 2727 in 57ms cpu (62ms real) (47575 rewrites/second)
BMaude: Execution result
933262154439441526816992388562667004907159682643816214685929638952175999932299
```

Iterative factorial example (cont.)

- . . . some steps, say 3, of the same operation call. . .

```
execn 3 fact(100)
rewrites: 201 in 2ms cpu (3ms real) (68835 rewrites/second)
BMaude: Execution state
WHILE(~ 0 == x)...[x = 98 y = 9900]
```

Iterative factorial example (cont.)

- . . . or symbolically execute some steps of a call to fact with some rational number.

```
sexec 3 fact(98 + R:Rat)
rewrites: 124 in 16ms cpu (24ms real) (7472 rewrites/second)
BMaude: symbolic execution state after 3 steps of fact(98 + R:Rat)
WHILE(~ 0 == x)...[x =(#1:Rat + 98)- 1 - 1 - 1 y = 1 *(#1:Rat + 98)*((#1:Rat +
    98)- 1)*((#1:Rat + 98)- 1 - 1)]
```

Recursive factorial example

- We continue with a recursive implementation of the factorial function to further illustrate the efficiency of the tool.

```
BMaude: Machine FACT2 loaded.
MACHINE FACT2
 VARIABLES y
 VALUES y = 1
 OPERATIONS
  fact(x)=
   IF ~(0 == x)
   THEN y := y * x ; fact(x - 1)
   ELSE print(y)
   END
END
```

Recursive factorial example (cont.)

- Let's call fact of 1000 and check its execution time.

```
exec fact(1000)
rewrites: 1051125 in 3295ms cpu (3315ms real) (318990 rewrites/second)
BMaude: Execution result
4023872600770937735437024339230039857193748642107146325437999104299385123986290
```

Model checking i

- We have exercised execution by rewriting and symbolic execution by narrowing.
- Let's play now with state search and LTL model checking. We will use a simple mutual exclusion protocol in this example.

Model checking ii

```
BMaude: Machine MUTEX loaded.
MACHINE MUTEX
 VARIABLES p1, p2
 CONSTANTS idle, wait, crit
 VALUES crit = 2 ; idle = 0 ; p1 = 0 ; p2 = 0 ; wait = 1
 OPERATIONS
  mutex =
   WHILE true DO
    IF p2 == idle /\ p1 == idle
    THEN p2 := wait [] p1 := wait
    ELSE
     IF wait == p2 /\ p1 == idle
     THEN p2 := crit [] p1 := wait
     ELSE
      IF p1 == idle /\ p2 == crit
      THEN p2 := idle [] p1 := wait
      ELSE
       IF wait == p1 /\ p2 == idle
       THEN p2 := wait [] p1 := crit
       ELSE
        IF wait == p2 /\ wait == p1
        THEN p2 := crit [] p1 := crit
```

Model checking (cont.) i

- Let's first search for a state where process p1 is in the critical section.
- Note that the initial state is given by the VALUES declaration.
- In this example,the state space is induced by a call to operation mutex,but in general,it could be any substitution.

Model checking (cont.) ii

```
search 1 mutex() where p1 = 2
rewrites: 1346 in 40ms cpu (43ms real) (32929 rewrites/second)
BMaude: Search trace
State 1 :
WHILE(true)...[p1 = 0 p2 = 0]
State 2 :
WHILE(true)...[p1 = 0 p2 = 0]
State 3 :
p2 := wait OR p1 := wait[p1 = 0 p2 = 0]
State 4 :
WHILE(true)...[p1 = 1 p2 = 0]
State 5 :
p2 := wait OR p1 := crit[p1 = 1 p2 = 0]
State 6 :
WHILE(true)...[p1 = 2 p2 = 0]
```

Model checking (cont.) i

- We can also check for the safety property using search,that is,a state where neither p1 nor p2 are in the critical section.
- No such state exists therefore our specification is safe.

```
search 1 mutex()where p1 = 2 and p2 = 2
rewrites: 882 in 34ms cpu (35ms real) (25770 rewrites/second)
BMaude: No solution while searching mutex()
```

Model checking (cont.) i

- Let's check for liveness now.
- For that we will use Linear Temporal Logic model checking.
- We say that a process is live if it tries to enter the critical section then it will eventually will.
- The model checker then returns a counter-example showing that there exists an infinite path where only process p2 enters the critical section.

Model checking (cont.) ii

```
mc mutex()|=[](p1(1)-> <> p1(2))
rewrites: 1009 in 46ms cpu (48ms real) (21656 rewrites/second)
BMaude: Model check counter example
Path from the initial state:
WHILE(true)...[p1 = 0 p2 = 0]-> WHILE(true)...[p1 = 0 p2 = 0]-> p2 := wait OR
    p1 := wait[p1 = 0 p2 = 0]
Loop:
WHILE(true)...[p1 = 1 p2 = 0]-> p2 := wait OR p1 := crit[p1 = 1 p2 = 0]->
    WHILE(true)...[p1 = 1 p2 = 1]-> p2 := crit OR p1 := crit[p1 = 1 p2 = 1]->
    WHILE(true)...[p1 = 1 p2 = 2]
```

Model checking (cont.) i

- Let's take a look now at another implementation of the mutex protocol that has the liveness property.

Model checking (cont.) ii

```
BMaude: Machine MUTEX2 loaded.
MACHINE MUTEX2
 VARIABLES p1, p2
 CONSTANTS idle, wait, crit
 VALUES crit = 2 ; idle = 0 ; p1 = 0 ; p2 = 0 ; wait = 1
 OPERATIONS
  mutex =
   WHILE true DO
    IF p2 == idle /\ p1 == idle
    THEN p2 := wait [] p1 := wait
    ELSE
     IF wait == p2 /\ p1 == idle
     THEN p1 := wait ; p2 := crit [] p2 := crit
     ELSE
      IF p1 == idle /\ p2 == crit
      THEN p2 := idle [] p1 := wait
      ELSE
       IF wait == p1 /\ p2 == idle
       THEN p1 := crit ; p2 := wait [] p1 := crit
       ELSE
        IF wait == p1 /\ p2 == crit
        THEN p2 := idle
```

Model checking (cont.)

```
mc mutex()|=[](p1(1)-> <> p1(2))
rewrites: 793 in 28ms cpu (33ms real) (28069 rewrites/second)
BMaude: Model check result
true
```

Model checking (cont.) i

- It also has safety...

```
mc mutex()|=[]~(p1(2)/\ p2(2))
rewrites: 783 in 5ms cpu (7ms real) (154012 rewrites/second)
BMaude: Model check result
true
```

Model checking (cont.) i

- . . . and strong liveness.

```
mc mutex()|=([]<> p1(1))->([]<> p1(2))
rewrites: 813 in 2ms cpu (7ms real) (339598 rewrites/second)
BMaude: Model check result
true
```

Conclusion

- Well, that is it for now.
- Let me conclude by saying that future work includes giving support to the complete Abstract Machine Notation,adding new verification techniques and refinement.
- Thanks for watching! Bye.