Making Maude Definitions more Interactive

Andrei Arusoaie[1] Traian Florin Şerbănuţă[1,2]
Chucky Ellison[2] Grigore Roşu[2]

[1]University Alexandru Ioan Cuza of Iaşi

[2]University of Illinois at Urbana-Champaign

March 28, 2012

Introduction
- Motivation

Adding I/O to Maude
- Scenario
- Our solution
- Maude I/O Interface Architecture

How it works in K

Conclusions and Future Work

Plan

Introduction
Motivation

Adding I/O to Maude
Scenario
Our solution
Maude I/O Interface Architecture

How it works in K

Conclusions and Future Work

Motivation

Defining programming languages in the K framework

In the K-Framework you can . . .

- . . . give operational semantics to a programming language.
- . . . run your semantics over programs defined in your language.

Motivation
Defining programming languages in the K framework

In the K-Framework you can . . .

- . . . give operational semantics to a programming language.
- . . . run your semantics over programs defined in your language.

However. . .

- . . . Most programming languages are interactive
- . . . Requiring support for I/O in the framework.

Current status of I/O in Maude

- *read-eval-print* loop from LOOP-MODE standard module

  > *. . . may not be maintained in future versions, because the support for communication with external objects makes it possible to develop more general and flexible solutions for dealing with input/output in future releases. [Maude Manual]*

- External objects?
  - Currently, only socket communication is supported

In this paper

- Achieve interactive I/O executions within Maude
- Using the socket external object to emulate an I/O external object.

Plan

Introduction
Motivation

Adding I/O to Maude
Scenario
Our solution
Maude I/O Interface Architecture

How it works in K

Conclusions and Future Work

Scenario - Simple Expression Language

```
mod EXP−SYNTAX is
  including INT .
  including STRING .
  sort Exp .
  subsort Int < Exp .
  op _+_ : Exp Exp →Exp [ditto] .
  op _*_ : Exp Exp →Exp [ditto] .
  op _ifnz_ : Exp Exp → Exp [strat(2 0)] .
  op nzloop : Exp → Exp [strat (0)] .
  op input : String → Exp .
  op print : String Exp → Exp .
endm
```

```
nzloop(print("3*x=",3 * input("x= (0 to stop)?  ")))
```

Scenario - Simple Expression Language

```
mod EXP−SYNTAX is
  including INT .
  including STRING .
  sort Exp .                            op _ifnz_ : Exp Exp → Exp [strat(2 0)] .
  subsort Int < Exp .                   op nzloop : Exp → Exp [strat (0)] .
  op _+_ : Exp Exp →Exp [ditto] .       op input : String → Exp .
  op _∗_ : Exp Exp →Exp [ditto] .       op print : String Exp → Exp .
endm
```

```
nzloop(print("3∗x=",3 ∗ input("x= (0 to stop)?  ")))
```

```
mod EXP−BASIC−SEMANTICS is including EXP−SYNTAX .
  eq nzloop(E:Exp) = nzloop(E:Exp) ifnz E:Exp .
  eq E:Exp ifnz 0 = 0 .
  eq E:Exp ifnz NzI:NzInt = E:Exp .
endm
```

Scenario - Simple Expression Language

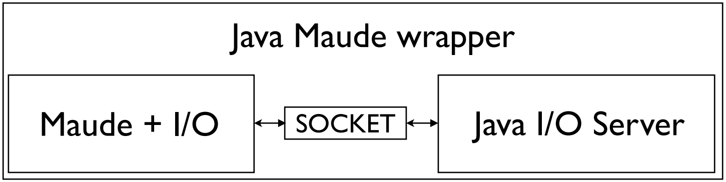
```
mod EXP−SYNTAX is
  including INT .
  including STRING .
  sort Exp .                           op _ifnz_ : Exp Exp → Exp [strat(2 0)] .
  subsort Int < Exp .                  op nzloop : Exp → Exp [strat (0)] .
  op _+_ : Exp Exp →Exp [ditto] .      op input : String → Exp .
  op _*_ : Exp Exp →Exp [ditto] .      op print : String Exp → Exp .
endm
```

```
nzloop(print("3*x=",3 * input("x= (0 to stop)?  ")))
```

```
mod EXP−BASIC−SEMANTICS is including EXP−SYNTAX .
  eq nzloop(E:Exp) = nzloop(E:Exp) ifnz E:Exp .
  eq E:Exp ifnz 0 = 0 .
  eq E:Exp ifnz NzI:NzInt = E:Exp .
endm
```

Problem: How to give semantics to `input`/`print`?

Our solution

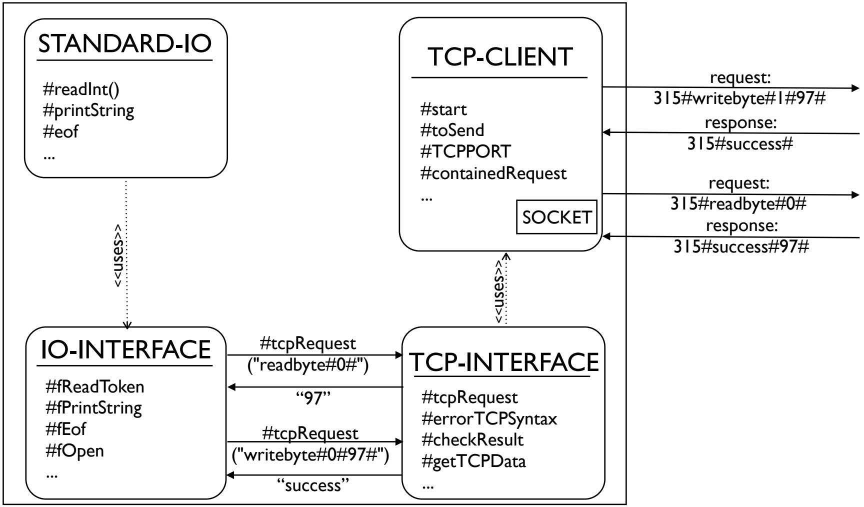


- Maude's SOCKET external objects allow interaction
- We use this interaction to provide:
  - a general client/server infrastructure to deal with I/O
  - a “friendly” Maude interface to access this infrastructure

Executing Scenario

```
$ cat io-test-cmd.maude

erew nzloop(print("3*x=",3 * input("x= (0 to stop)?  "))) .

$ java -jar MaudeIO.jar --maudeFile io-test.maude

--commandFile io-test-cmd.maude

x= (0 to stop)?  -12

3*x=-36

x= (0 to stop)?  0

3*x=0

Maude> ==========================================

erewrite in KRUNNER : nzloop(print("3*x=", 3 * input("x= (0 to stop)?  "))) .

rewrites:  6487 in 45ms cpu (53780ms real) (141396 rewrites/second)

result Zero:  0

Maude> Bye.
```

Maude Client



STANDARD-IO Interface

Basic console I/O operations

$\textbf{op}\ \#\text{printString} : \textsf{String} \ \rightarrow\ \textsf{IOResult}\ .$
$\textbf{op}\ \#\text{readInt}() : \ \rightarrow\ \textsf{IOResult}\ .$
$\textbf{op}\ \#\text{eof}() : \ \rightarrow\ \textsf{IOResult}\ .$

$\textbf{op}\ \#\text{printChar} : \textsf{Char} \rightarrow \textsf{IOResult}\ .$
$\textbf{op}\ \#\text{readChar}() : \rightarrow \textsf{IOResult}\ .$
$\textbf{op}\ \#\text{readToken}() : \rightarrow \textsf{IOResult}\ .$

IO-INTERFACE

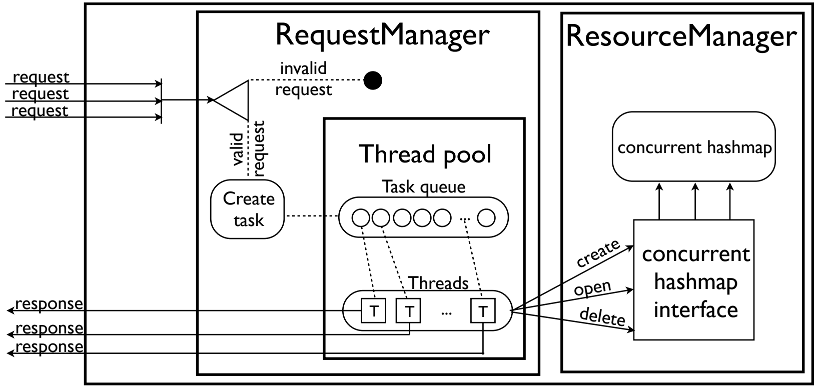
I/O operations

```
op #open : String → IOResult .
op #reopen : Nat String → IOResult .
op #close : Nat → IOResult .
op #fEof : Nat → IOResult .

op #fReadByte : Nat →IOResult .
op #fPutByte : Nat Nat →IOResult .
op #fPrintChar : Nat Char → IOResult .
op #fReadChar : Nat →IOResult .

op #flush : Nat → IOResult .
op #tell : Nat → IOResult .
op #seek : Nat Nat → IOResult .
op #fPeekByte : Nat →IOResult .

op #fReadToken : Nat →IOResult .
op #fReadInt : Nat → IOResult .
op #fPrintString : Nat String →
IOResult .
```

An I/O semantics for EXP

```
mod EXP−SEMANTICS is
  including EXP−BASIC−SEMANTICS . including STANDARD−IO .

  op _;_ : IOResult Exp → Exp [strat (1 0)] .
  op read : → Exp .

  eq input(S:String)
   = #printString(S:String);
     #readInt();
     read .

  eq print(S:String,I:Int)
   = #printString(S:String + string(I:Int,10) + "\n");
     I:Int .

  eq #success ; E:Exp = E:Exp .
  eq #int(I:Int) ; read = I .
endm
```

Java I/O Server



Plan

Introduction
Motivation

Adding I/O to Maude
Scenario
Our solution
Maude I/O Interface Architecture

How it works in K

Conclusions and Future Work

The K definition of EXP
EXP-SYNTAX

```
module EXP−SYNTAX
  syntax Exp ::= #Int
               | Exp "+" Exp [strict hook(#INT:_+Int_)]
               | Exp "*" Exp [strict hook(#INT:_*Int_)]
               | Exp "ifnz" Exp [strict(2)]
               | "nzloop" Exp [prec 0]
               | "input" "(" #String ")"
               | "print" "(" #String "," Exp ")"  [strict(2)]
end module
```

The K definition of EXP
EXP-SEMANTICS

```
module EXP imports EXP−SYNTAX
  configuration ⟨k⟩ $PGM:Exp ⟨/k⟩
                ⟨in stream="stdin"⟩ .List ⟨/in⟩
                ⟨out stream="stdout"⟩ .List ⟨/out⟩

  syntax KResult ::= #Int

  rule nzloop E:Exp  =>  (nzloop E) ifnz  E

  rule _ ifnz  0   =>  0
  rule E:Exp ifnz I:#NzInt  =>  E

  rule ⟨k⟩ print (Str:#String,I:#Int)   =>  I ⋯⟨/k⟩
       ⟨out⟩⋯ .   =>  ListItem(Str) ListItem(I) ListItem("\n") ⟨/out⟩

  syntax Exp ::= "read"
  rule ⟨k⟩ input(Str:#String)   =>  read ⋯⟨/k⟩
       ⟨out⟩⋯ .   =>  ListItem(Str) ⟨/out⟩

  rule ⟨k⟩ read   =>  I ⋯⟨/k⟩
       ⟨in⟩ ListItem(I:#Int)   =>  . ⋯⟨/in⟩
end module
```

Plan

Introduction
Motivation

Adding I/O to Maude
Scenario
Our solution
Maude I/O Interface Architecture

How it works in K

Conclusions and Future Work

Conclusions and Future Work

- Run interactive programs
- Direct access from Maude to `stdin`, `stdout`, `stderr`, and files
- A "friendly" interface
- Extensions: allow access to more URI-specified streams