Rule Engine and Rule Language

Hao Xu
xuh@cs.unc.edu
2011/2/17

Goals

- Readability
- Robustness
- Testing
- Performance
- Modularity
- Efficiency of Development

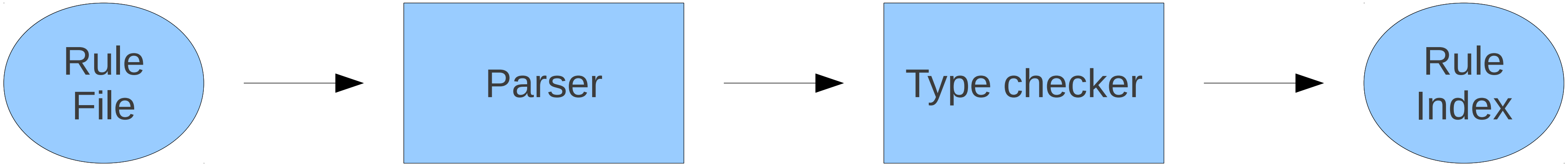
Improvements

- Direct Support for Enhanced Rulegen Syntax
- Comments
- String
- Expression in Arguments
- Variable Expansion
- Type System
- Testing
- Error Messages

- Caching
- Rule Indexing
- Memory Management
- New Datatypes
- New Micro Services
- Directives
- Backward Compatibility

Overview

Compile Time



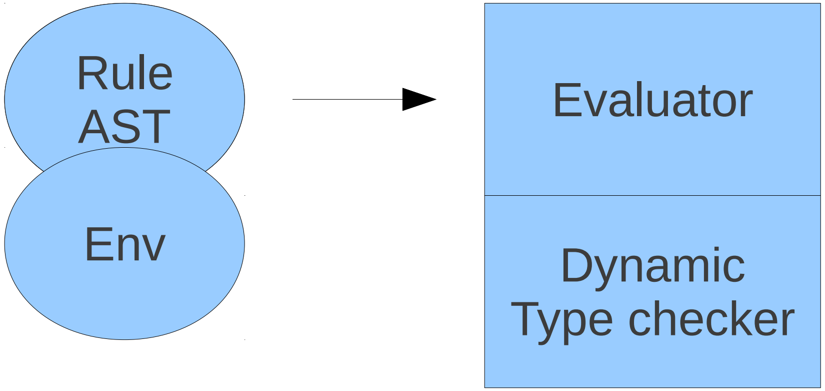
Rule File

Parser

Type checker

Rule Index

Runtime



Rule AST

Env

Evaluator

Dynamic Type checker

Internal Rule Syntax

RuleHead | Condition | Actions | Recovery

acCreateUser | |
acPreProcForCreateUser##acCreateUserF1##
acPostProcForCreateUser |
nop##nop##nop

Rulegen Syntax

```
RuleHead {
    on(Condition) {
        Action ::: Recovery;
        ...
        Action ::: Recovery;
    }
}

acCreateUser {
    acPreProcForCreateUser ::: nop;
    acCreateUserF1 ::: nop;
    acPostProcForCreateUser ::: nop;
}
```

```
sum(*n, *s) {
    *s = 0;
    for(*i=0;*i<*n;*i=*i+1) {
        *s = *s + *i;
    }
}
```

Comments

- Comments starts with #, but not ##
- Comments ends in EOL

```
# this is a comment
*a=1; # this is a comment
```

Strings

- Strings are quoted using either '' or “”

  “xyz” → xyz

  “x'y'z” → x'y'z

- Special characters are escaped, just like in C or Java

  “x\“y\”z” → x“y”z

  “a\tb\tc” → a   b   c

- Variable names are interpreted in Strings

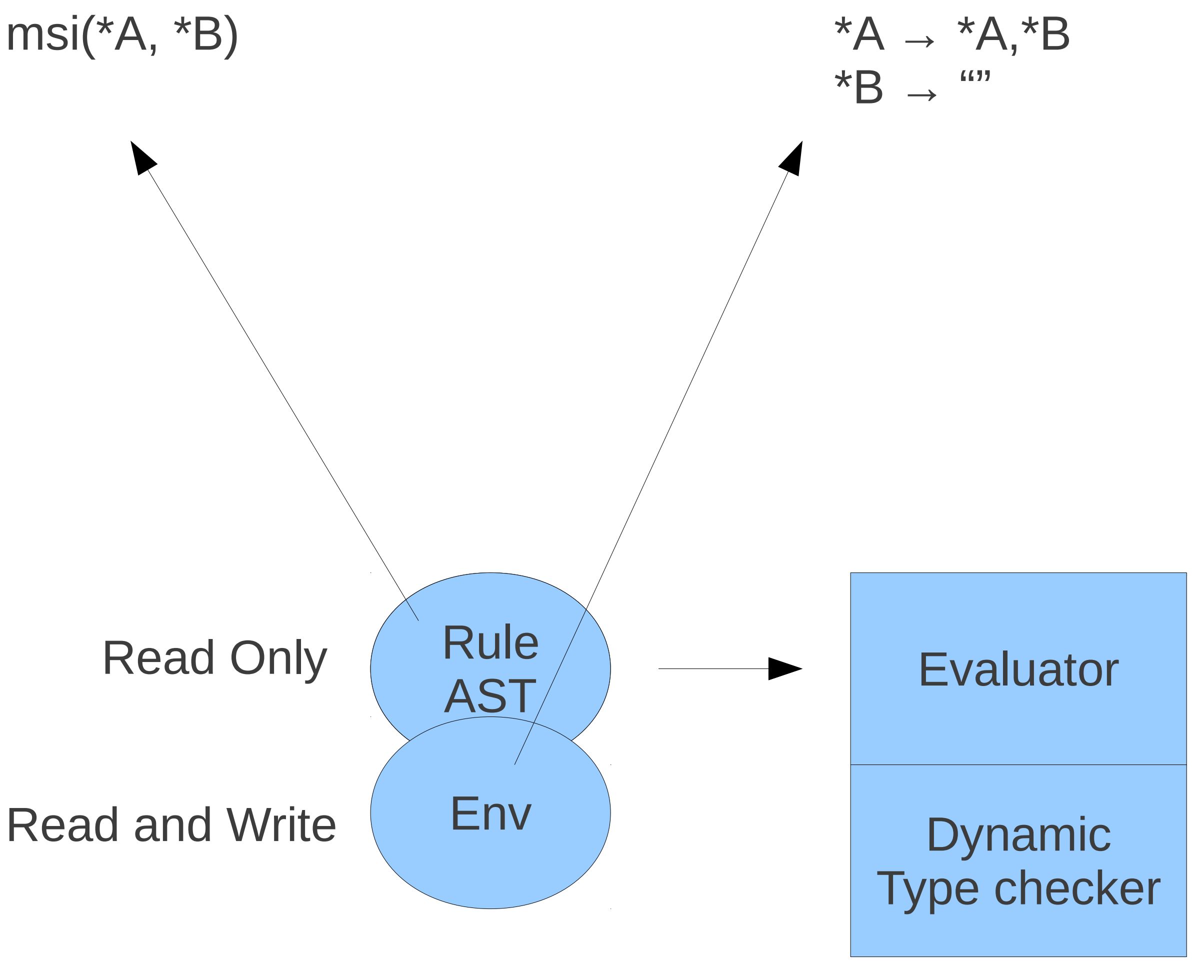
  “a is *x” ↔ “a is ”++str(*x)

  “x*y*z” ↔ “x*y”++str(*z)

Expression in Arguments

- Without support for expression in arguments
  - Either
    - msi(1+2)
    - Write the micro service msi so that it evaluates the parameter.
  - Or
    - *A=1+2
    - msi(*A)
- With support for expression in arguments
  - msi(1+2)
  - The rule engine evaluates the expression

Variable Expansion



msi(*A, *B)

*A → *A,*B
*B → “”

Read Only

Rule AST

Read and Write

Env

Evaluator

Dynamic Type checker

Type System

- Discover some bugs before rules are executed
  - For example: bool + int
- Make it easy to write micro services
  - RE takes care of type checking/conversion
- Types of system micro services are known statically
- User defined micro services are dynamically typed
- Mixing dynamic typing with static typing
- Type Inference for variables

Testing

```
testWsc(*RES) {
    assert("1!=0", *RES);
}
```

Demo

- Unit Testing
- Error Messages
- Factorial
- Eight Queens Puzzle
- Wolf, Sheep, and Cabbage

Factorial

```
factorial(*f,*n) {
    if(*n == 0) then {
        *f = 1;
    } else {
        factorial(*g, *n - 1);
        *f = *g * *n;
    }
}
```

$$n! = \begin{cases} 1, n = 0 \\ n \times (n-1)!, n > 0 \end{cases}$$

Eight Queens Puzzle

1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 1 0

0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0

Eight Queens Puzzle

accept(*board, *a, *b)

printBoard(*board)

updateBoard(*board, *a, *b, *elem, *board2)

Eight Queens Puzzle

```
queens {
   *board = list(
              list(0,0,0,0,0,0,0,0),
              list(0,0,0,0,0,0,0,0),
              list(0,0,0,0,0,0,0,0),
              list(0,0,0,0,0,0,0,0),
              list(0,0,0,0,0,0,0,0),
              list(0,0,0,0,0,0,0,0),
              list(0,0,0,0,0,0,0,0),
              list(0,0,0,0,0,0,0,0)
          );
   tryRow(*board, 0, 0);
}
```

Eight Queens Puzzle

```
tryRow(*board, *a, *b) {
    accept(*board,*a,*b);
    updateBoard(*board, *a, *b, 1, *board2);
    elem(*board, *a+1) ::: if(*a+1==size(*board2)) {printBoard(*board2);};
    tryRow(*board2, *a+1, 0);
}
tryRow(*board, *a, *b) {
    elem(elem(*board, *a),*b+1);
    tryRow(*board, *a, *b+1);
}
```

Wolf, Sheep, and Cabbage

- [W, S, C, H]
- Initial: [1,1,1,1], [0,0,0,0]
- Move the Sheep:
  - [1,1,1,1], [0,0,0,0] → [1,0,1,0], [0,1,0,1]
- Cross the River:
  - [1,1,1,1], [0,0,0,0] → [1,1,1,0], [0,0,0,1]

Wolf, Sheep, and Cabbage

wscSucc(*b)

wscAccept(*a, *b)

wscMove(*a1,*b1,*a2,*b2, *i)

wscNotVisited(*conf, *visited)

Wolf, Sheep, and Cabbage

```
wscTry(*a, *b, *visited) {
    on(wscSucc(*b)==0) { writeLine("stdout", "succ"); }
    or { wscMove(*a, *b, *a2, *b2, 0); wscGoal(*a2, *b2, *visited); }
    or { wscMove(*a, *b, *a2, *b2, 1); wscGoal(*a2, *b2, *visited); }
    or { wscMove(*a, *b, *a2, *b2, 2); wscGoal(*a2, *b2, *visited); }
    or { wscMove(*a, *b, *a2, *b2, 3); wscGoal(*a2, *b2, *visited); }
}
```

Wolf, Sheep, and Cabbage

```
wscGoal(*a, *b, *visited) {
    wscAccept(*a, *b);
    wscNotVisited(list(*a, *b), *visited);
    wscTry(*a, *b, cons(list(*a, *b), *visited));
    writeLine("stdout", str(list(*a,*b)));
}
```