The iRODS Rule Language

Hao Xu
xuh@email.unc.edu
2014/6/18

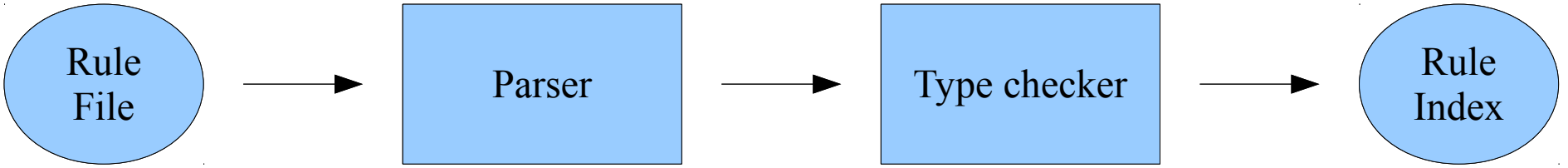
Outline

- Overview
- New Features
- Examples

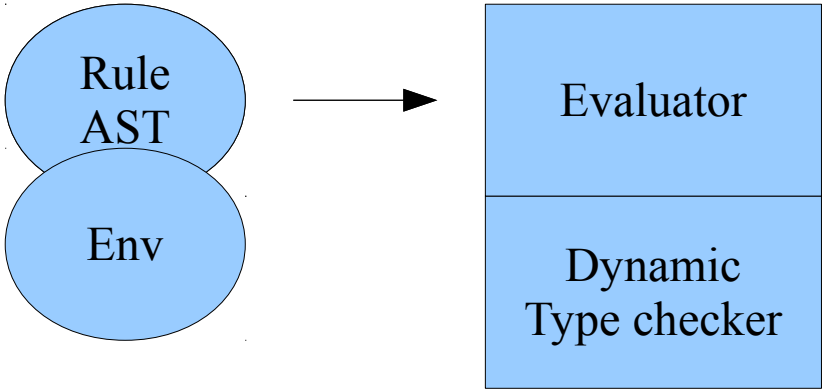
Overview

Overview

Compile Time



Runtime



Comments

- Comments starts with #
- Comments ends in EOL

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

Variables

- Local Var, Param: *A
- Session Var: $userNameClient (this can be extend through the DVN file)

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

- Convert to string using str()
- String concatenation using ++
- Variable names are interpreted in Strings

  “a is *x” ↔ “a is ”++str(*x)

  “x*y*z” ↔ “x*y”++str(*z)

Rule

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

Example

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

Another Example

```
acPostProcForPut {  ← this is a PEP
    postProcForPut;
}
postProcForPut {
    on(include($objPath)) {
        postProcForCreateCommon($objPath, $dataSize, true);
        # preview must be generated after the file is uploaded
        genPreviewGen($objPath, *previewThumbPath, *previewPath);
    }
}
```

.re vs .r

- System rule
  - $IRODS_CONFIG/<filename>.re
  - Enable by adding <filename> to server.config's reRuleSet property
  - Available to all
  - Trigger by PEP or called by other rules
  - No “main” rule
- User rule
  - *.r
  - Submit by irule
  - Available to this session only
  - Specify input/output
  - If contain multiple rules, the first rule is the “main” rule
  - “main” rule cannot have any parameters

Type System

- Discover some bugs before rules are executed
  - For example: bool + int
  - postProcForCreateCommon : input string * input double * output boolean -> integer
- Make it easy to write microservices
  - RE takes care of type checking/conversion
- Types of system microservices are known statically
- User defined microservices are dynamically typed
- Mixing dynamic typing with static typing
- Type Inference for variables

New Features

Enhanced KeyValuePair Support

- Dot Operator:

  *kvp.<key>

  *kvp.<key> = <value>

  where both <key> and <value> are string expressions, if <key> is an identifier then quotes can be omitted

- Loop over keys:

```
foreach(*key in *kvp) {
    writeLine(“stdout”, “key: “++*key++” value: “++*kvp.*key);
}
```

Language Integrated Gen Query (LIGQ)

```
*a = SELECT META_DATA_ATTR_NAME WHERE
DATA_NAME = 'data_name';


foreach(*row in *a) {
    writeLine(“stdout”, *row.META_DATA_ATTR_NAME);
}
```

Collection Spider

```
foreach(*path in /tempZone/home/*usrDir) {
    writeLine(“stdout”, *path);
}
```

recursively list all files under the collection and its subcollections

Microservice Plugin

```
MICROSERVICE_BEGIN(
  add,
  INT, a, INPUT,
  INT, b, INPUT,
  INT, c, OUTPUT ALLOC)

  c = a + b;
MICROSERVICE_END
```

Advanced Features

- Type declaration
- Rule condition indexing
- Function syntax:
  - fundef, if-then-else, let, match, tuples, constant definition
- Logical Programming:
  - backtracking, recovery action, errorcode, errormsg, fail, failmsg, cut, ApplyAllRules
- Documentation: https://wiki.irods.org/index.php/Changes_and_Improvements_to_the_Rule_Language_and_the_Rule_Engine

Examples

Examples

- Factorial
- Eight Queens Puzzle
- Wolf, Sheep, and Cabbage
- Fibonacci

Factorial

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

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

Factorial

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

```
factorial(*n) =
    if *n == 0 then 1 else factorial(*n - 1) * *n;
```

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)));
}
```

Fibonacci

```
fib(*fs, *i, *n) {
    if(*i == 0) {
        *fs = list(0);
    } else if(*i == 1) {
        *fs = list(1, 0);
    } else {
        *fs = cons(elem(*fs,0)+elem(*fs,1), *fs);
    }
    if(*i < *n) {
        fib(*fs, *i+1, *n);
    }
}
```

$$F_i = \begin{cases} 0, i = 0 \\ 1, i = 1 \\ F_{i-1} + F_{i-2}, i > 1 \end{cases}$$