RLSRunner and KRunner: Linking Rascal with K for Program Analysis and Execution

Mark Hills, Paul Klint, & Jurgen J. Vinju

2nd International Workshop on the K Framework and its Applications
August 9, 2011
Cheile Gradistei, Romania
Overview
Overview

• Motivation
Overview

• Motivation

• Tool components: Rascal
Overview

• Motivation

• Tool components: Rascal

• Tool components: K
Overview

• Motivation

• Tool components: Rascal

• Tool components: K

• Demo
Overview

• Motivation

• Tool components: Rascal

• Tool components: K

• Demo

• Wrap-up
Motivation: Why This Tool?
Motivation: Why This Tool?

• Many K and K in Maude specifications exist -- want reuse
Motivation: Why This Tool?

• Many K and K in Maude specifications exist -- want reuse

• Integrating with graphical environments currently ad-hoc, bad user experience
Motivation: Why This Tool?

• Many K and K in Maude specifications exist -- want reuse

• Integrating with graphical environments currently ad-hoc, bad user experience

• Want a general method to integrate these specifications with Rascal-based IDEs
Motivation: Why This Tool?

- Many K and K in Maude specifications exist -- want reuse
- Integrating with graphical environments currently ad-hoc, bad user experience
- Want a general method to integrate these specifications with Rascal-based IDEs
- (Personal) Wanted something like this all during my PhD
How Should it Work (RLSRunner)?
How Should it Work (RLSRunner)?
How Should it Work (RLSRunner)?
How Should it Work (RLSRunner)?

```java
function main(void)
begin
    var $m x;
    var $m y;
    var $f z;
    var $s u;

    write $x + $y; # should be fine
    write $x + $z; # should be a type error
    write $x * $z; # should be fine
    write $x * $y * $u; # should be fine
    write $x * $u + $y * $u; # should be fine
    write $x * $u + $z * $y; # should be a type error

    return 0;
end
```
How Should it Work (RLSRunner)?
How Should it Work (RLSRunner)?
How Should it Work (KRunner)?
An Introduction to Rascal

• Rascal: A meta-programming language for source code analysis and transformation

• Based on concepts learned from ASF+SDF, but with a more traditional programming language feel

• Features: parsing, structured control flow, rich data types (algebraic data types, lists, sets, tuples, maps, relations, etc), pattern matching, enumerations, higher order functions, etc
Defining Grammars in Rascal
Tool Components: Rascal (ShellExec)

```rascal
PID pid = createProcess(maudeLocation.path);
writeTo(pid, toRun);
res = readFrom(pid);
killProcess(pid);
```
Tool Components: Rascal (ResourceMarkers)

Rascal

```rascal
data Message = error(str msg, loc at)
  | warning(str msg, loc at)
  | info(str msg, loc at);
```

Rascal

```rascal
import Message;

public void java removeMessageMarkers(loc resourceLoc);

public void java addMessageMarkers(set[Message] markers);
```
Tool Components: Rascal (RLSRunner ADT)

\[
data \text{ RLSRunner} = \text{RLSRun}(\text{loc maudeFile, str(str, list[str]) pre, RLSResult(str) post});
\]
Tool Components: Rascal (Maude-ifier)

```rascal
if ((Program)`<Decl* decls> <FunDecl+ funDecls>` := p)
    return located(p,"Pgm",
                    "__(<showDecls([d|d<-decls]>),
                         <showFunDecls({f|f<-funDecls}>)")");
```

```rascal
syntax Program = Default: Decl* decls FunDecl+ funDecls;
```
Tool Components: Rascal (Returning Results)

```
data RLSResult = SILFAnalysisResult(bool foundErrors, set[Message] messages) ;

void exec(Tree pt, loc l) {
    str pgm = maudeify(pt, true, policy);
    RLSRunner rlsRunner = RLSRun(silfSpec, pre, post);
    RLSResult res = runRLSTask(maudeExec, rlsRunner, pgm);
    if (SILFAnalysisResult(true,msgs) := res)
        addMessageMarkers(msgs);
}
```
public str generateProgramModule(Tree pgm, str topSort, str pgmName,  
   str pgmMod, str syntaxMod) {
    set[str] identifiers = { "<id>" | /Id id <- pgm } - "main";
    str identifierListing =  
      "syntax Id ::= <intcalate(" | ", [ili<-identifiers]) > ";
    str pgmDeclaration = "syntax <topSort> ::= <pgmName>";
    return "kmod <pgmMod> is including <syntaxMod>
      '<identifierListing>
      '<pgmDeclaration>
      ,
      'macro <pgmName> =
        '<pgm>
        ,
        'endkm
      '";
}
Tool Components: K (Rascal Source Locations)

```kmaude
fmod RASCAL-LOCATION is
  including STRING .
  including INT .
  sort RLocation .
  op sl : String Int Int Int Int Int Int -> RLocation .
endfm
```

K/Maude
Tool Components: K (Location Semantics)

\[
\begin{align*}
\text{op currLoc : } & \text{RLocation } \rightarrow \text{State [format (r! o)] } . \\
\text{op rloc : } & \text{RLocation } \rightarrow \text{ComputationItem } . \\
\text{eq } & k(\text{rloc}(\text{RL}) \rightarrow K) \text{ currLoc}(\text{RL}') = k(K) \text{ currLoc}(\text{RL}) . \\
\text{eq } & k(\text{exp(locatedExp}(\text{E}, \text{RL}) \rightarrow K) \text{ currLoc}(\text{RL}') = \\
& k(\text{exp}(\text{E}) \rightarrow \text{rloc}(\text{RL}') \rightarrow K) \text{ currLoc}(\text{RL}) .
\end{align*}
\]
Tool Components: K (Generating Results)

\[
\text{op makeAnalysisMsg : OutputList -> String .}
\]

\[
\text{eq makeAnalysisMsg(\text{warning}(\text{level}(1) \ \text{msgloc}(RL) \ \text{msg}(S) \ WIS), \ OL) =}
\]

\[
("||\text{1:::} + \text{rloc2str}(RL) + "::: + S + "||") + \text{makeAnalysisMsg}(\text{OL}) .
\]