New `.key` Syntax

Wojciech Mostowski

`http://www.key-project.org`

Lökeberg, June 8th, 2005

Overview

Overview

Yes, we do have new `.key` syntax.

Overview

Seriously:

- What has changed
- Why it had to be changed
- How it has changed
- Comments, insights
- Wrap-up

What

The KeY logic input syntax:

- Rule files
- User problem files
- Prover input via taclet instantiation window
- Not the schematic Java syntax

```
!self = null -> <{
    self.aMethod();
}> all i:int. self.a[i] = 0
```

- How much? – Almost everything

Why

- Keywords – strings that were keywords in the taclet language could not be used/referred to in the `.key` files:

```
program variables {
    mypackage.program.MainClass self;
}
```

- Antlr parser inheritance – cool feature, but in our case required huge amounts of duplicated code – SWE horror

- Code Maintainability

Why cont'd

- Lack of verbosity – ambiguous syntax, difficult for the parser, and for the user:

```
mypackage.MyClass::instance
```

– static attribute?
– query?
– function symbol?

```
obj.pack.obj.MyClass::obj.pack.obj.MyClass::pack.pack.Obj::myclass
```

```
a[{var t}exp]
```

– box modality?

- Infix operators (pretty syntax)

- The power of KeY was limited by the syntax!

Why cont'd

- Lack of verbosity – ambiguous syntax, difficult for the parser, and for the user:

```
mypackage.MyClass::instance
```

– static attribute?
– query?
– function symbol?

```
obj.pack.obj.MyClass::obj.pack.obj.MyClass::pack.pack.Obj::myclass
```

```
a[{var t}exp]
```

– box modality?

- Infix operators (pretty syntax)

- The power of KeY was limited by the syntax!

New Syntax

119 side branch check-ins later. . .

How – Keywords, Identifiers, Numbers

- All keywords start with '\', no multi-word keywords:

```
schema variables
```
$\longrightarrow$
```
\schemaVariables
```

Exceptions: `#inType, #isObject, true, false`

- '+' and '~' not allowed in identifiers, digits cannot start an identifier (one exception), numbers are separate tokens, hex notation:

```
name, name_a, name1, $name, #name, $name1, ...
name@pre, name1@pre, name_a@pre, <name>
1(2(#))
12345, 0x12df
\name, \name_a
\name1, n$ame, n#ame, <name1>, 00000+++++~~~~~
```

How – Modalities

- **Short:**

```
\< Java block \> formula
\[ Java block \] formula
\[[ Java block \]] formula
```

- **Long:**

```
\diamond Java block \endmodality (formula)
\box Java block \endmodality (formula)
```

Also `\throughout`, `\diamond_tra`, etc.

- **Very long, schema modalities:**

```
\modality{diamond} Java \endmodality (formula)
\modality{#allmodal} Java \endmodality (formula)
```

How – File Headers

- **Pointing to Java source:**

```
\javaSource "path1", "path2", ... ;
```

- **Options (declaring and choosing):**

```
\optionsDecl {
  cat1:{choice1_1, choice1_2};
  cat2:{choice2_1, choice2_2};
}
\withOptions cat1:choice1, cat2:choice2 ;
```

- **Prover settings:** `\setttings { "..." }`

- **Others:** `\heuristicsDecl, \include, \includeLDTS`

How – Schema Variables

- Global:

```
\schemaVariables {
   \modalOperator {op1, op2 } #var1, #var2;
   \term (\rigidTerm) SortName #var1, #var2;
   \formula (\rigidFormula) #var1, #var2;
   \variables SortName #var1, #var2;
   \depending SortName #var1, #var2;
   \modifies #var1, #var2;
   \program(List) ProgramSVSort #var1, #var2;      }
```

- (Pseudo-)local:

```
my_rule {
   \schemaVar \formula #formula;
   \find (...#formula...)   ...   }
```

How – Binding Expressions

- **Single variable:**

```
\bindingOp Sort v; ...
```

- **Multiple variables:**

```
\bindingOp (Sort v1; Sort v2) ...
```

- **Schema variables:**

```
\bindingOp #var; ...
\bindingOp (#var1; #var2) ...
```

- `\bindingOp`: `\bind`, `\forall`, `\exists`, `\for`, `\ifEx`
- **Substitutions:**

```
{\subst (\substWary) #v; term1} term2
```

How – Sort Names

- **Fully qualified:**

```
package1.package2.SortName
```

- **Arrays and sets, convenient version:**

```
package1.package2.SortName[]
package1.package2.SortName{}
```

- **Arrays and sets, less convenient, but real version:**

```
ArrayOfpackage1.package2.SortName
SetOfpackage1.package2.SortName
```

- **Fully qualified sort and class name can occur wherever you would expect**

How – Attributes, Queries, Functions

- **Attributes and queries:**

```
obj.attr1@(package1.Class1).attr2@(package2.Class2)
obj.query@(package.Class)
```

- **Static attributes and queries:**

```
package.Class.attr
package.Class.query(...)
```

- **Special function names:**

```
valid.Sort::fname
ArrayOfpackage.Class::instance(obj) = TRUE
what.Ever::my_very_own_invented_name
```

How – Infix Operators

- Change in function names:

`neg`	$\longrightarrow$	`neglit`
`~m`	$\longrightarrow$	`neg`
`~d`	$\longrightarrow$	`sub`
`+`	$\longrightarrow$	`add`

- Infix operators:

unary –	`neg`, `neglit`	
`*`, `/`, `%`	`mul`, `div`, `mod`	(associate to the right)
`+`, `-`	`add`, `sub`	(associate to the right)
`<`, `<=`, `>`, `>=`	`lt`, `leq`, `gt`, `geq`	

How – Infix Operators

- Change in function names:

`neg`	$\longrightarrow$	`neglit`
`~m`	$\longrightarrow$	`neg`
`~d`	$\longrightarrow$	`sub`
`+`	$\longrightarrow$	`add`

- Infix operators:

unary –	`neg`, `neglit`	
`*`, `/`, `%`	`mul`, `div`, `mod`	(associate to the right)
`+`, `-`	`add`, `sub`	(associate to the right)
`<`, `<=`, `>`, `>=`	`lt`, `leq`, `gt`, `geq`	

```
\forall(int i; int j) (i > 0 & j >= 0 -> i + j >= i)
```

How – Other Stuff

- `\varcond` in taclets: look it up
- Conditional expressions:

```
\if (...)  \then (...)  \else (...)
```

- Quantified array indices in `\modifies` clauses:

```
a[ind1 .. ind2], a[*] (a[0..a.length-1])
```

- Type casts (not supported by the logic yet):

```
((valid.Sort)obj).attr
```

Results and Side Effects

- By making the parser stricter:
  - found bugs in rules
  - found bugs in tests!!!
- Freed some characters (~, `) to be possibly used in the future
- Introducing new program schema variables and logic meta operators is much simpler: no changes to the parser required
- (Obviously) Got rid of parser inheritance
- All the other problems gone

Statistics

	Old Parser	New Parser
antlr (`.g`) code	202 kB	
generated Java code		
compiled byte-code		

Old Parser = `lexer.g` + `decls.g` + `terms.g` + `globalDeclarationTerms.g` + `taclets.g` + `problem.g`

New Parser = `lexer.g` + `keyparser.g`

Statistics

	Old Parser	New Parser
antlr (.g) code	202 kB	138 kB
generated Java code		
compiled byte-code		

Old Parser = `lexer.g` + `decls.g` + `terms.g` + `globalDeclarationTerms.g` + `taclets.g` + `problem.g`

New Parser = `lexer.g` + `keyparser.g`

Statistics

	Old Parser	New Parser
antlr (.g) code	202 kB	138 kB
generated Java code	876 kB	
compiled byte-code		

Old Parser = `lexer.g` + `decls.g` + `terms.g` + `globalDeclarationTerms.g` + `taclets.g` + `problem.g`

New Parser = `lexer.g` + `keyparser.g`

Statistics

	Old Parser	New Parser
antlr (.g) code	202 kB	138 kB
generated Java code	876 kB	320 kB
compiled byte-code		

Old Parser = `lexer.g` + `decls.g` + `terms.g` + `globalDeclarationTerms.g` + `taclets.g` + `problem.g`

New Parser = `lexer.g` + `keyparser.g`

Statistics

	Old Parser	New Parser
antlr (.g) code	202 kB	138 kB
generated Java code	876 kB	320 kB
compiled byte-code	568 kB	

Old Parser = `lexer.g` + `decls.g` + `terms.g` + `globalDeclarationTerms.g` + `taclets.g` + `problem.g`

New Parser = `lexer.g` + `keyparser.g`

Statistics

	Old Parser	New Parser
antlr (`.g`) code	202 kB	138 kB
generated Java code	876 kB	320 kB
compiled byte-code	568 kB	200 kB

Old Parser = `lexer.g` + `decls.g` + `terms.g` + `globalDeclarationTerms.g` + `taclets.g` + `problem.g`

New Parser = `lexer.g` + `keyparser.g`

Further Changes

- Don’t like something?
- Ideas for small improvements?
- Go to the WiKi page:

  `http://i12www.ira.uka.de/~klebanov/`
  `keywiki/index.cgi?ParserImprovements`

- Small things still to be done, individuals assigned

Apologies

- To all the people that got their projects and side branches broken because of the syntax change

Apologies

- To all the people that got their projects and side branches broken because of the syntax change
- To everybody for making you re-edit all your private `.key` files

Apologies

- To all the people that got their projects and side branches broken because of the syntax change
- To everybody for making you re-edit all your private `.key` files
- To German and Swedish keyboard users...

Acknowledgements

For help and motivation: to Richard and Philipp