The abc scanner and parser

Laurie Hendren and the **abc** team

Challenges

- Unambiguous LALR(1) grammar for the complete AspectJ language that is a natural extension of the Java grammar. (easy to understand and extend)
- Express as much of the language specification in the grammar as possible (for example, differentiate in the grammar where class pattern is required and where a general type pattern is allowed).
- Handle the different sublanguages and associated reserved words in a well-defined manner.

abc Solution Overview

- Jflex-based scanner that is built on top of Polyglot's Java scanner.
- **abc**'s scanner uses state to distinguish between different scanning contexts.
  `abc/src/abc/aspectj/parse/aspectj.flex`
- LALR(1) grammar expressed as a clean extension to Polyglot's base Java grammar (originally defined by Scott Ananian - JavaCup)
  `abc/src/abc/aspectj/parse/java12.cup`
  `abc/src/abc/aspectj/parse/aspectj.ppg`

Scanning AspectJ

- Really three different sublanguages:
  1. normal Java code
  2. aspect declarations
  3. pointcut definitions
- Different sub-languages have different lexical structure, for example

  `if*.*1.Foo+.new(..)`

  Java: reserved("if"), op("*"), op("."), op("*"),float(1.0), id("Foo"), op("+"), reserved("new"), op("("), op("."), op("."), op(")")

  Pointcut: IdPat("if*"), op("."), IdPat("*1"), op("."), Id("Foo"), op("+"), reserved("new"), op("("), op(".."), op(")")

abc Scanner Uses States

- Scanner maintains a stack of states.
- New state is pushed when entry into lexical scope is detected, and the scanner is put into the new state.
- When the end of a lexical state is detected, state is popped from the stack and scanner put into the state now at the top of the stack.
- Four major states, each state has well-defined entry/exit points, and its own lexical structure, including specific reserved words defined for that state.
- A reserved word is easily associated to two different token types, based on current state of the scanner. For example, **`if`** can have two different token types, one for the regular **`if`** and one for the pointcut **`if`**.

Scanner States

Java: Default state, **`aspect`**, **`privileged`**, and **`pointcut`** are reserved words. This state is entered at **`class`** or **`interface`** and exited at matching **`}`**. (finding the matching } requires a nesting counter)

Aspect: Begins at the **`aspect`** keyword and ends at the end of the aspect declaration's body. Has, in addition to above reserved words, **`after`**, **`around`**, **`before`**, **`declare`**, **`issingleton`**, **`percflow`**, **`percflowbelow`**, **`pertarget`**, **`perthis`**, **`pointcut`**, and **`proceed`**.

abc Scanner States (2)

Pointcut: Four contexts in which pointcut expressions may be found:

per clause: `pertarget ( ..... )`

declare declaration: `declare ..... ;`

body of a pointcut declaration: `pointcut ..... ;`

header of an advice declaration: `after ..... {`

Reserved words in this state are only:
`adviceexecution` **`args`**, **`call`**, **`cflow`**, **`cflowbelow`**, **`error`**, **`execution`**, **`get`**, **`handler`**, **`if`**, **`initialization`**, **`parents`**, **`precedence`**, **`preinitialization`**, **`returning`**, **`set`**, **`soft`**, **`staticinitialization`**, **`target`**, **`this`**, **`throwing`**, **`warning`**, **`within`** and **`withincode`**.

abc Scanner States (3)

PointcutIfExpr: inside a pointcut, an if pointcut has a nested expression, same scanning state as Aspect, but state returns to pointcut state at terminating parenthesis.

```
..... if ( ..... ) .....
```

Defining a LALR(1) grammar as Polyglot ext.

1. Define new alternatives to existing rules in the polyglot Java grammar.
2. Define new grammar productions. (sometimes must accept a slightly too large language and then weed)

All new alternatives

⟨*type_declaration*⟩ ::= ⟨*aspect_declaration*⟩

⟨*class_member_declaration*⟩ ::= ⟨*aspect_declaration*⟩
| ⟨*pointcut_declaration*⟩

⟨*interface_member_declaration*⟩ ::= ⟨*aspect_declaration*⟩
| ⟨*pointcut_declaration*⟩

⟨*method_invocation*⟩ ::= 'proceed' '(' ⟨*argument_list_opt*⟩ ')'

Adding alternatives in Polyglot

```
/* add the possibility of declaring an
    aspect to type_declaration */

extend type_declaration ::=
     aspect_declaration:a
       {: RESULT = a; :}
  ;
```

New aspect-specific productions

```
aspect_declaration ::=
    modifiers_opt:a PRIVILEGED modifiers_opt:a1
      ASPECT:n IDENTIFIER:b
      super_opt:c interfaces_opt:d
      perclause_opt:f
      aspect_body:g
  {: RESULT = parser.nf.AspectDecl(parser.pos(n,g),
                true, a.set(a1), b.getIdentifier(),
                c, d, f, g);
  :}
```

aspect_declaration (continued)

```
|  modifiers_opt:a
     ASPECT:n IDENTIFIER:b
     super_opt:c interfaces_opt:d
     perclause_opt:f
     aspect_body:g
   {: RESULT = parser.nf.AspectDecl(parser.pos(n,g),
                false, a, b.getIdentifier(),
                c, d, f, g);
   :}
;
```

abc grammar includes pointcuts

⟨*basic_pointcut_expr*⟩ ::=

’(’ ⟨*pointcut_expr*⟩ ’)’

| ’call’ ’(’ ⟨*method_constructor_pattern*⟩ ’)’

| ’execution’ ’(’ ⟨*method_constructor_pattern*⟩ ’)’

| ’initialization’ ’(’ ⟨*constructor_pattern*⟩ ’)’

| ’preinitialization’ ’(’ ⟨*constructor_pattern*⟩ ’)’

| ’staticinitialization’ ’(’ ⟨*classname_pattern_expr*⟩ ’)’

| ’get’ ’(’ ⟨*field_pattern*⟩ ’)’

| ’set’ ’(’ ⟨*field_pattern*⟩ ’)’

| ’handler’ ’(’ ⟨*classname_pattern_expr*⟩ ’)’ . . .

(continued)

⟨*basic_pointcut_expr*⟩ ::= . . .

| ’adviceexecution’ ’(’ ’)’

| ’within’ ’(’ ⟨*classname_pattern_expr*⟩ ’)’

| ’withincode’ ’(’ ⟨*method_constructor_pattern*⟩ ’)’

| ’cflow’ ’(’ ⟨*pointcut_expr*⟩ ’)’

| ’cflowbelow’ ’(’ ⟨*pointcut_expr*⟩ ’)’

| ’if’ ’(’ ⟨*expression*⟩ ’)’

| ’this’ ’(’ ⟨*type_id_star*⟩ ’)’

| ’target’ ’(’ ⟨*type_id_star*⟩ ’)’

| ’args’ ’(’ ⟨*type_id_star_list_opt*⟩ ’)’

| ⟨*name*⟩ ’(’ ⟨*type_id_star_list_opt*⟩ ’)’

Specific Patterns

⟨*method_constructor_pattern*⟩ ::=
    ⟨*method_pattern*⟩
| ⟨*constructor_pattern*⟩

⟨*method_pattern*⟩ ::=
    ⟨*modifier_pattern_expr*⟩ ⟨*type_pattern_expr*⟩
    ⟨*classtype_dot_id*⟩
    '(' ⟨*formal_pattern_list_opt*⟩ ')' ⟨*throws_pattern_list_opt*⟩
| ⟨*type_pattern_expr*⟩ ⟨*classtype_dot_id*⟩
    '(' ⟨*formal_pattern_list_opt*⟩ ')' ⟨*throws_pattern_list_opt*⟩

Summing up ....

- State-based scanner, plus LALR(1) grammar:
  - clearly defines lexical scopes and associated reserved words
  - naturally handles different sub-languages in AspectJ
  - clean addition to the base Java grammar
  - easy to understand
  - easy to extend
- More detailed scanning/parsing document at: **`http://abc.comlab.ox.ac.uk/doc`**