The \texttt{abc} scanner and parser

Laurie Hendren and the \texttt{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
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(""))
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`.
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.
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

\[ \langle \text{type
declarati}on \rangle ::= \langle \text{aspect
declarati}on \rangle \]

\[ \langle \text{class
dernembe}dratio}n \rangle ::= \langle \text{aspect
declarati}on \rangle \\
| \quad \langle \text{pointcut
declarati}on \rangle \]

\[ \langle \text{interface
dernembe}dratio}n \rangle ::= \langle \text{aspect
declarati}on \rangle \\
| \quad \langle \text{pointcut
declarati}on \rangle \]

\[ \langle \text{method_invocation} \rangle ::= 'proceed' '(' \langle \text{argument
testis_opt} \rangle ')' \]
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

\[
\langle \text{basic\_pointcut\_expr} \rangle ::= \\
\quad \text{'}(\langle \text{pointcut\_expr} \rangle \text{')} \mid \\
\quad \text{'}\text{call} \text{'}(\langle \text{method\_constructor\_pattern} \rangle \text{')} \\
\quad \text{'}\text{execution} \text{'}(\langle \text{method\_constructor\_pattern} \rangle \text{')} \\
\quad \text{'}\text{initialization} \text{'}(\langle \text{constructor\_pattern} \rangle \text{')} \\
\quad \text{'}\text{preinitialization} \text{'}(\langle \text{constructor\_pattern} \rangle \text{')} \\
\quad \text{'}\text{staticinitialization} \text{'}(\langle \text{classname\_pattern\_expr} \rangle \text{')} \\
\quad \text{'}\text{get} \text{'}(\langle \text{field\_pattern} \rangle \text{')} \\
\quad \text{'}\text{set} \text{'}(\langle \text{field\_pattern} \rangle \text{')} \\
\quad \text{'}\text{handler} \text{'}(\langle \text{classname\_pattern\_expr} \rangle \text{')} \ldots
\]
\[
\langle \text{basic\_pointcut\_expr} \rangle ::= \ldots
\]

| 'adviceexecution' '(' ')'
| 'within' '(' \langle \text{classname\_pattern\_expr} \rangle ')' 
| 'withincode' '(' \langle \text{method\_constructor\_pattern} \rangle ')' 
| 'cflow' '(' \langle \text{pointcut\_expr} \rangle ')' 
| 'cflowbelow' '(' \langle \text{pointcut\_expr} \rangle ')' 
| 'if' '(' \langle \text{expression} \rangle ')' 
| 'this' '(' \langle \text{type\_id\_star} \rangle ')' 
| 'target' '(' \langle \text{type\_id\_star} \rangle ')' 
| 'args' '(' \langle \text{type\_id\_star\_list\_opt} \rangle ')' 
| \langle \text{name} \rangle '(' \langle \text{type\_id\_star\_list\_opt} \rangle ')' }
Specific Patterns

\[ \langle method\_constructor\_pattern \rangle ::= \]
\[ \langle method\_pattern \rangle \]
\[ \mid \langle constructor\_pattern \rangle \]

\[ \langle method\_pattern \rangle ::= \]
\[ \langle modifier\_pattern\_expr \rangle \langle type\_pattern\_expr \rangle \]
\[ \langle classtype\_dot\_id \rangle \]
\[ '(', \langle formal\_pattern\_list\_opt \rangle ')', \langle throws\_pattern\_list\_opt \rangle \]
\[ \mid \langle type\_pattern\_expr \rangle \langle classtype\_dot\_id \rangle \]
\[ '(', \langle formal\_pattern\_list\_opt \rangle ')', \langle throws\_pattern\_list\_opt \rangle \]
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](http://abc.comlab.ox.ac.uk/doc)