Lecture 32 — December 2, 2010

Patrick Lam *version 1*

Other Domain-Specific Languages

I'm going to wrap up by discussing a few more domain-specific languages. These come from Chapter 13 of *Language Implementation Patterns* by Terence Parr.

Protein Structures. Recall that cells use RNA to produce proteins. RNA consists of four bases: A, C, G and U. RNA is linear, but folds in various interesting ways. "Folded RNA secondary structures" contain pairs between Gs and Cs and between As and Us, like in the following diagram:

We can use compiler techniques to determine if the RNA has the appropriate pairs. This is just the matching-parentheses problem: compare G–C to (–) and A–U to [–]. What technique should we use?

We can also add parser actions or build an AST.

Languages to make pictures. The book suggests that one might want to use an English-like scripting language for pictures, e.g.

```
x = Cube; y = Sphere; y.radius = 20
draw x
draw y to the left of x and behind x
```

There are also a couple of real languages in this space, including TikZ and Processing[1]:

```
\begin{tikzpicture}[shorten >=1pt,auto,
                    node distance=0.5cm and 2cm,
                    semithick,initial text=]
  \node[bw] (s)
                    {web server};
  \node[bw] (pp) [right= of s]
                    {PHP parser};
  \node[bw,text width=6em]   (byte)
                   [right= of pp]
                    {PHP bytecode interpreter};
  \node[bw] (lib) [below= of byte]
                    {PHP library (C)};
  \draw (pp.north west)+(-0.5,0.5)
      rectangle ($(lib.south east)+(1,-0.5)$);
  \node (php) [above= of pp] {PHP};

  \path (s) edge node {} (pp);
  \path (pp) edge node {} (byte);
  \path (byte) edge node {} (lib);
\end{tikzpicture}
```

```
void draw() {
  background(0);

  // Update the angle
  angle += 0.005;

  // Rotate around the center axis
  translate(width/2, 0, -128);
  rotateY(angle);
  translate(-extrude.width/2, 100, -128);

  // Display the image mass
  for (int y = 0; y < extrude.height; y++) {
    for (int x = 0; x < extrude.width; x++) {
      stroke(values[x][y]);
      point(x, y, -values[x][y]);
    }
  }
}
```

TikZ is an internal domain-specific language embedded into LaTeX (props to the author), while Processing is an external domain-specific language. Both TikZ and Processing are compiled via source-to-source translations: TikZ to TeX backends, and Processing to Java (e.g. to embed on webpages). Which compiler components do we need to process each of these languages?

XML. Of course there are XML parsers, but you don't necessarily need to use an XML parser to get data out of an XML file.

- Sometimes you can use `grep` to extract data from XML, e.g. by matching on the < that starts a tag.
- It's also possible to program a system which responds to callbacks on entry to certain XML tags, as in the Simple API for XML (SAX).
- If you want to recognize all of the text which is not in a tag ("whatever" in WIG), you can parse tags and get rid of them, leaving you with the rest of the document.
- To guarantee well-formedness of the XML, in the sense of it having matching open and close tags, you can lex the XML and keep a stack of tags (like what we're doing with `enter` and `leave` methods in our Visitors).

Java Extensions. Another use of compiler technology is to extend existing programming languages, perhaps to make a domain-specific language out of an existing language.

There are tools for extending existing grammars: JastAdd and Polyglot. I've used them to write supersets of Java with additional features. Again, the idea is a source-to-source translation into base Java, with some static analysis along the way.

[1] `http://www.processing.org`