Different Ways to Build a DSL

Pat Hanrahan

Approaches

External DSL

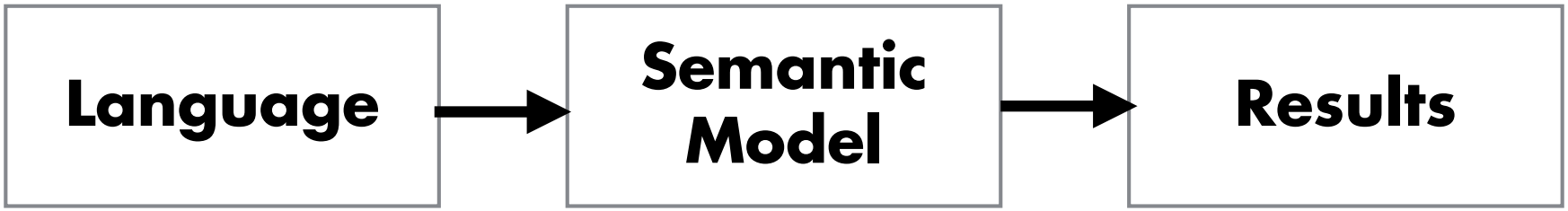
An external DSL is implemented as a standalone language.

Embedded (Internal) DSL

An internal DSL is embedded within a another language. Ideally, the host language has features that make it easy to build DSLs.

External DSLs

Language Implementation



Parse (yacc and lex)

Evaluate

calc.py

lexical analysis
syntactic analysis
interpretation

Advantages

- **Flexibility (*syntax, semantics*)**
- **Simple languages are simple (little languages)**

Disadvantages

- **Yet-Another-Programming-Language**
- **Syntactical cacophony**
- **The slippery slope of generality**
- **Interpretation is slow**
- **Hard to interoperate with other languages**
- **No tool chain: IDE, debugger, profiler, ...**

Embedded DSLs

```
// OpenGL

glMatrixMode(GL_PROJECTION);
glPerspective(45.0);

for( ;; ) {
    glBegin(TRIANGLES);
        glVertex(…);
        glVertex(…);
        …
    glEnd();
}

glSwapBuffers();
```

```
// OpenGL “Grammar”

<Scene> = <BeginFrame> <Camera> <World>
<EndFrame>

<Camera> = glMatrixMode(GL_PROJECTION) <View>
<View> = glPerspective | glOrtho

<World> = <Objects>*
<Object> = <Transforms>* <Geometry>
<Transforms> = glTranslatef | glRotatef | …
<Geometry> = glBegin <Vertices> glEnd
<Vertices> = [glColor] [glNormal] glVertex
```

Fluent Interface

“Composable API Calls”

```
// https://jquery.com/

<ul>
  <li>One</li>
  <li>Two</li>
  <li>Three</li>
</ul>

// turn first element green
$("li:first").css("color", "green");
```

https://www.d3-graph-gallery.com/graph/density_basic.html

http://d3js.org/

```
// Lynq in C#

int count =
    (from character in Characters
    where character.Episodes > 120
    select character).Count();

// Simpler syntax
```

Advantages

- No need to learn another language
- Familiar syntax
- Still have access to general-purpose features
- Can interoperate with other libraries and classes
- Complete tool chain

Disadvantages

- Syntax is rigid and verbose
- Interpreters are still slow
- Hard to debug DSLs using current tool chains
- Hard to limit features in the language
- Still hard to develop

DSL Building Features

Powerful types: Algebraic data types, type classes or classes with inheritance

Polymorphism (multiple interpretations)

Higher-order functions and lambdas

Flexible syntax

Shallow Embedding

A *shallow embedding* is when the expressions are interpreted in the semantics of the base language

calc1.py

- **Direct interpretation of arithmetic**

Deep Embedding

A deep embedding first builds an abstract syntax tree (AST). The abstract syntax tree is typically an algebraic data type. The AST is then evaluated with an interpreter.

calc2.py

- **AST represented as a tuple**

Operator Overloading

https://docs.python.org/2/reference/datamodel.html

“Overloading”

Not all “operations” can be intercepted

- **Arithmetic operators**
- **Iteration operators**
- **Function definition?**
- **Type/class definition?**
- **Equality?**
- **Assignment?**

“Monkey patching” like this can be dangerous

Type-directed embedding

Efficient Interpreters

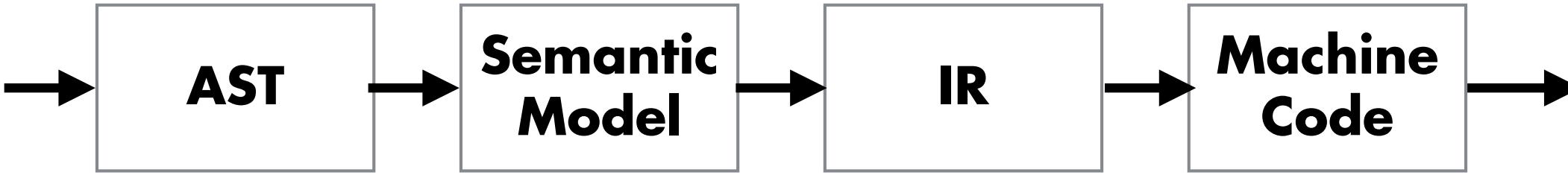
Type safety

- **Base language parses**
- **AST is guaranteed to be well-formed**

Remove overhead of interpretation

- **Typed tagless final interpreters ...**
- **Multi-Stage programming**
- **Partial evaluation**

Language Implementation



Parse

Analysis
Transformation

Optimization
Code Generation

Mini-APL Assignment

Implement a simple array processing language in C++

Simple recursive descent parser that builds an AST (that we provide)

"Lower" the AST to LLVM. Generate efficient code!

Assignment released today, due ...