1 GOALS AND PROCESS FOR DESIGNING A LANGUAGE

1.1 Simplicity
Useful especially eases parsing the input and the language implementation. Simplicity generally derives from regularity in the programming language.

1.2 Generality
Since it aids simplicity

1.3 Ability to manipulate programs as data
To ease construction of interpreters and compilers to allow "meta programming". We want to be able to manipulate programs and data easily (meta-programming, genetic programming)

Exercise 1.1 What is meta-programming?

Exercise 1.2 What are meta-programming features in Java and C#?

1.4 Process
We will operate by consensus, if necessary we will vote, or I will select if there’s an arbitrary choice to be made.

2 PARTS OF A LANGUAGE

• means of computation
  – primitive expressions (data)

• means of combination
  – procedure calls, sequencing, loops, container data structures

• means of abstraction
  – procedures, abstract data types, macros

3 DATA OR MEANS OF COMPUTATION

Exercise 3.3 What should be the types in the language?
First, there is a basic difference between primitive types and compound types. Primitive types cannot be broken down further, they are like atoms. Compound types can be thought of as molecules. Consider the following primitive types and compound types as examples.

<table>
<thead>
<tr>
<th>primitive types</th>
<th>compound types</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>vector (arrays)</td>
</tr>
<tr>
<td>char</td>
<td>(linked) lists</td>
</tr>
<tr>
<td>symbol</td>
<td>ports (files)</td>
</tr>
<tr>
<td></td>
<td>stream</td>
</tr>
</tbody>
</table>

We may also need: strings, boolean, procedures.

**Exercise 3.4** How do we know whether we have enough types?

(Time and space efficiency, completeness) sum and product types lists as able to simulate both

**Exercise 3.5** How does simplicity affect the list of types?

key test is simulation: one type be used to simulate another (or vice versa)?

### 3.1 Simulation, Generality

**Definition 3.1** A type $T$ can simulate a type $U$ if using $T$ one can get the same behavior as $U$.

Behavior includes: functionality (inputs, outputs) size (space used) time (complexity of running ops)

**Exercise 3.6** How do we know whether we have too many types?

(User-defined types, simplicity of numeric types)

**Exercise 3.7** What types should be built-in?

Depends on their utility, in efficiency, portability, and whether they need to be used in means of combination (such as Booleans).

### 4 Type Checking and Type Errors

**Exercise 4.8** What is a type error?

**Exercise 4.9** Should there be a static type system, or should there be run-time type checking?

**Definition 4.1** **Dynamic type checking** means type errors are detected (in general) at run-time.

**Definition 4.2** **Static type checking** means type errors are detected before the program is run.
Exercise 4.10 Which is better for users?

Exercise 4.11 Which is more flexible?

\[ (+ 3 (if (zero? 0) 4 'done)) \]

illegal in typed languages

Exercise 4.12 Which suits interpreters better?

Exercise 4.13 Which gives better performance?

4.1 Operations

Exercise 4.14 What literals, constructors, and observers are needed?

Consider the following types and the operations allowed on them as an example.

<table>
<thead>
<tr>
<th></th>
<th>operations</th>
<th>literals</th>
<th>special forms</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>+, -, *, /</td>
<td>1, 1.2, 5.7e4</td>
<td>if</td>
</tr>
<tr>
<td>char</td>
<td>&lt;, &gt;, &lt;=,...</td>
<td>'a', 'b'</td>
<td></td>
</tr>
<tr>
<td>vector</td>
<td>&lt;, &gt;, ...</td>
<td>1, 2, 3</td>
<td>#asym</td>
</tr>
<tr>
<td>symbol</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 4.15 Are any special forms (e.g., &&, || in Java) needed?

5 MEANS OF COMBINATION SYNTAX

5.1 Syntax of expressions

Exercise 5.16 What should the syntax be for expressions?

- consider parsing things like 3 + 4
- parsing, parse trees, operator precedence (high, low)
- recall that an additional goal is to be able to manipulate programs as data

Exercise 5.17 How can we represent parse trees as linear text?
Exercise 5.18 What data structures do we have that would work?

5.2 Commands and combinations of commands

Exercise 5.19 What commands do we need?

Assignment, sequencing, if-then-else, perhaps other special forms

Exercise 5.20 What is an expression language?

Exercise 5.21 What is a statement language?

6 MEANS OF ABSTRACTION

6.1 Naming

Exercise 6.22 How should we name expressions and procedures?

Exercise 6.23 Do we want to have local declarations within expressions?

6.2 Parameterization and function declarations

Exercise 6.24 How should we declare parameters for procedures?

Exercise 6.25 Can we unify the two kinds of declarations?

References