Principles of Programming Languages

Two important problems

- how to provide a precise definition of a programming language?
- how to implement a higher-level programming language?

Formal Specification of Programming Languages

Main parts of a programming language specification

- syntax
- (optional) static semantics
- dynamic semantics

Statically versus Dynamically Typed Languages

Static versus Dynamic

- static: before program execution
- dynamic: during program execution (that is, at run-time)

Statically Typed Languages

A static semantics is provided: rules for checking that

- operators/statements are used with consistent types of values
- variables are declared and used consistently with their declaration
- pros: early error detection, efficiency

Dynamically Typed Languages

- no static semantics is defined
- inconsistent uses of values generate dynamic type errors
- pros: simplicity, expressive power

Examples

Syntax error

```
x = ; // Syntax error in most languages: illegal start of expression
```

Static error

```
int x=0; // Java, statically typed language
if(y<0) x=3; else x="three";
// Static error: incompatible types, String cannot be converted to int
```

Dynamic error

```
x=null;
if(y<0) y=1; else y=x.value;
// Dynamic error if y>=0:
// in Java: Exception in thread "main" java.lang.NullPointerException
// in JavaScript (dynamic language): cannot read property 'value' of
    null
```

Syntax

Definition of alphabet

A *finite non-empty set* of *symbols* A

Definition of string

A string over an alphabet A is a *sequence* $u : [1..n] \rightarrow A$

- $[1..n]$ is the interval of natural numbers m such that $1 \leq m \geq n$
- u is a *total* function
- n is the *length* of u: $length(u) = n$

Syntactic notion of program

A program is a string over an alphabet A

Example of strings

Empty string

- *empty string* $u : [1..0] \to A$
- remark: $[1..0] = \emptyset$
- there exists a unique function $u : \emptyset \to A$
- standard notations for the empty string: ϵ or λ or Λ

A non empty string

Let us consider $A = \{'\mathtt{a}', \ldots, '\mathtt{z}'\} \cup \{'\mathtt{A}', \ldots, '\mathtt{Z}'\}$ (alphabet of lowercase and uppercase English letters)
The function $u : [1..4] \to A$ s.t.

- $u(1) = '\mathtt{W}'$
- $u(2) = '\mathtt{o}'$
- $u(3) = '\mathtt{r}'$
- $u(4) = '\mathtt{d}'$

More concrete representation: `"Word"`

Example of strings

A string of length 1

Let us consider $A = \{'\texttt{a}', \dots, '\texttt{z}'\} \cup \{'\texttt{A}', \dots, '\texttt{Z}'\}$

The function $u : [1..1] \rightarrow A$ s.t. $u(1) = '\texttt{s}'$

More concrete representation: `"s"`

Remark: `"s"` and `'s'` are different: `"s"` is a string, `'s'` is an alphabet symbol

String concatenation

Definition

- $\mathit{length}(u \cdot v) = \mathit{length}(u) + \mathit{length}(v)$
- for all $i \in [1..\mathit{length}(u) + \mathit{length}(v)]$
  $(u \cdot v)(i) = \texttt{if } i <= \mathit{length}(u) \texttt{ then } u(i) \texttt{ else } v(i - \mathit{length}(u))$

Monoids and strings

- concatenation is *associative*, but not *commutative*
- the empty string is the identity element

Iteration of concatenation

u^n defined by induction on n (natural number):

- $u^0 = \epsilon$
- $u^{n+1} = u \cdot u^n$

Intuition: u^n is u concatenated with itself n times