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 before execution 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

Type checks are performed only at run-time

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

Examples

Syntax error

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

Static type 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
```

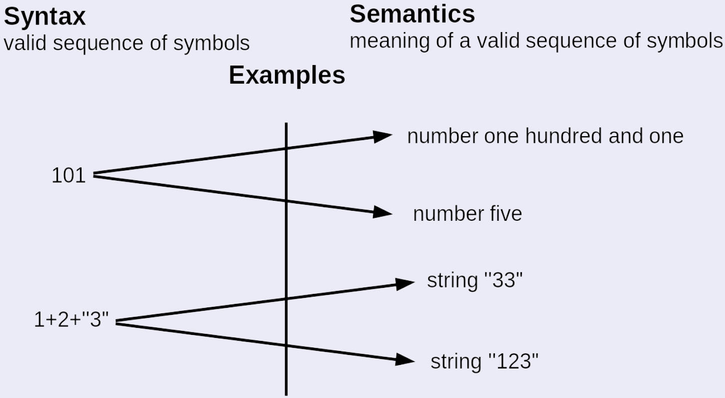
Static and dynamic errors

Remark

- static type errors
  - detected before execution
  - only in statically typed languages
- dynamic errors
  - detected during execution
  - in all languages

Syntax versus semantics

Few examples



Syntax

Definition of alphabet

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

Examples

$A_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $A_2 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}\}$

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 i such that $1 \leq i \leq n$
- 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: $[1..0] = \emptyset$, no numbers i such that $1 \leq i \leq 0$
- there exists a unique empty string, that is, the 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 $A = \{\mathtt{a}, \ldots, \mathtt{z}\} \cup \{\mathtt{A}, \ldots, \mathtt{Z}\}$

The function $u : [1..1] \to A$ s.t. $u(1) = \mathtt{s}$

More concrete representation: `"s"`

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

String concatenation

Definition

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

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):

- $u^0 = \epsilon$ (the empty string)
- $u^{n+1} = u \cdot u^n$

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

Standard sets of strings

Definition of A^n, A^+ and A^*

Let A be an alphabet

- A^n = the set of all strings over A of length n
- A^+ = the set of all strings over A of length greater than 0
- A^* = the set of all strings over A

Provable facts

- $A^0 = \{\epsilon\}$
- $A^+ = \bigcup_{n>0} A^n$
- $A^* = \bigcup_{n\geq 0} A^n = A^0 \cup A^+$

Formal language: syntactic notion of language

Definition

A language L over an alphabet A is a subset of A^*

Example

The set *Id* of all identifiers

- $A = \{\mathtt{a},\ldots,\mathtt{z}\} \cup \{\mathtt{A},\ldots,\mathtt{Z}\} \cup \{\mathtt{0},\ldots,\mathtt{9}\}$ (definition of the alphabet)
- $Id = \{\ \texttt{"a"}, \texttt{"b"}, \ldots, \texttt{"a0"}, \texttt{"a1"}, \ldots\}$ (definition of the language)

Problem

Is it possible to define L in a finite way?
Solution: define L as the composition of simpler languages

Operators to compose languages

- union: $L_1 \cup L_2$
- concatenation: $L_1 \cdot L_2 = \{u \cdot w \mid u \in L_1, w \in L_2\}$

Intuition

Union

$L = L_1 \cup L_2$: any string of L is a string of L_1 or a string of L_2

Example:

$Lett = \{ \texttt{"a"}, \texttt{...}, \texttt{"z"} \} \cup \{ \texttt{"A"}, \texttt{...}, \texttt{"Z"} \}$

Concatenation

$L = L_1 \cdot L_2$: any string of L is a string of L_1 followed by a string of L_2

Examples:

- $\{\texttt{"a"},\texttt{"ab"}\} \cdot \{\epsilon,\texttt{"1"}\} = \{\texttt{"a"},\texttt{"ab"},\texttt{"a1"},\texttt{"ab1"}\}$
- $Id = Lett \cdot A^*$ with $A = \{\texttt{a},\texttt{...},\texttt{z}\} \cup \{\texttt{A},\texttt{...},\texttt{Z}\} \cup \{\texttt{0},\texttt{...},\texttt{9}\}$