How We Define a *Syntactically* Valid *X*

Ten examples of tokens of a C-like programming language are:

`;   <   --   -   )   {   IDENTIFIER   UNSIGNED-INT-LITERAL   while   if`

Each token T is a set of strings of characters; each member of that set is called an ***instance*** of T. Token instances are also called ***lexemes***. *Different tokens have no instances in common*!

Note: In sec. 2.3 of Sethi, the tokens **`IDENTIFIER`** and **`UNSIGNED-INT-LITERAL`** are called **name** and **number**, and a token instance is called a ***spelling***.

Java Examples 3 instances of **`IDENTIFIER`** are: **`x`** **`prevVal`** **`pi_2`**

3 instances of **`UNSIGNED-INT-LITERAL`** are: **`23`** **`1_275_113`** **`0b10010`**

A ***lexical syntax specification*** of a programming language specifies its tokens and the sequence of token instances into which any given piece of source code should be decomposed.

Very commonly, the only tokens that have more than one instance are **`IDENTIFIER`** and tokens such as **`UNSIGNED-INT-LITERAL`** or **`STRING-LITERAL`** whose instances are literals.

An ***instance of a sequence of tokens*** $T_1 \ldots T_n$ is a piece of source code that should be decomposed into a sequence of token instances $t_1 \ldots t_n$ in which each t_i is an instance of T_i. If a piece of source code is an instance of a sequence of tokens $T_1 \ldots T_n$, then we say $T_1 \ldots T_n$ is the ***sequence of tokens*** of that piece of source code.

Java Example

`x23 = 4;` is an instance of: **`IDENTIFIER = UNSIGNED-INT-LITERAL ;`**

`IDENTIFIER = UNSIGNED-INT-LITERAL ;` is the sequence of tokens of: **`x23 = 4;`**

To define a “syntactically valid *X*”, where *X* is a language construct (e.g., *X* = “Java source file” or *X* = “Java while statement”), the language designer first formulates a definition of a ***syntactically valid sequence of tokens for*** *X* in such a way that the following is true:

> A sequence of tokens T_1 ... T_n is ***syntactically valid for X*** if (and, roughly speaking, only if) T_1 ... T_n is the sequence of tokens of a possibly legal *X*.

“a possibly legal *X*” means a piece of text that either is a legal *X* or would be a legal *X* in an appropriate context (e.g., with the right variable declarations). “roughly speaking” allows some exceptions to the condition’s “only if” part. Such definitions are commonly written using **BNF**, **EBNF**, or some related notation.

Java Example The following sequence of 9 tokens

`IDENTIFIER [ IDENTIFIER ] = IDENTIFIER / UNSIGNED-INT-LITERAL ;`

is ***syntactically valid for a Java assignment statement***, since it is the sequence of tokens of this possibly legal Java assignment statement: `a[y] = b/2;`

A piece of source code is a ***syntactically valid*** *X* if and only if its sequence of tokens is syntactically valid for *X*.

Java Example `b[b] = c/0;` is a ***syntactically valid*** Java assignment statement, as it is another instance of the above sequence of 9 tokens. (Note, however, that `b[b] = c/0;` is certainly ***not*** a possibly legal Java assignment!)

If the only tokens that have more than one instance are `IDENTIFIER` and tokens whose instances are literals of various kinds, then a *syntactically valid X* is, roughly speaking, a piece of source code that can be transformed to a *possibly legal X* by making zero or more changes of the following kinds:

- **Replace an identifier with another (e.g., `b[b] = c/0;` ⇒ `a[b] = c/0;`).**
- **Replace a literal with another of the same kind (e.g., `a[b] = c/0;` ⇒ `a[b] = c/2;`).**