Alongside algorithms, formal languages are one of the big topics of theoretical computer science. A formal language is constrained by a defined set of rules specific to the language and is typically understandable by a computer. Formal methods, as techniques for dealing with formal languages, are widely used in computer science, but also outside of it.

1 Syntax and semantics of formal languages

A formal language is specified by its syntax and semantics. Syntax describes the structure of expressions in the language, but does not say anything about their meaning. Semantics, on the other hand, assigns a meaning to syntactically valid expressions. For example:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kristóf</td>
<td>the person</td>
</tr>
<tr>
<td>Kristóf is a student at IST</td>
<td>true</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>XIII</td>
<td>13</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
</tr>
</tbody>
</table>

To translate syntax into semantic objects we use semantic brackets: \[[ ]\]

The brackets work like this: \[[1101]] \rightarrow 13

In general, for formal but also natural languages it holds that:

**Def 1:** A sentence is a sequence of letters from an alphabet.

**Def 2:** A language is a set of sentences.

One way to define languages is to use rules:

**Def 3:** A rule consists of two parts:

1. A (possibly empty) set of premises
2. A conclusion

**Def 4:** Given a rule set \( R \), the set theory(\( R \))(set of theorems of \( R \)) is the smallest set \( S \), such that for every rule \( r \in R \): if all premises of \( r \) are in \( S \), then the conclusion of \( r \) is also in \( S \).

We now give several examples of language defining systems:
1.1 Example 1: Language of binary numbers

Alphabet: \{0,1\}

Rules:

- **r1**: no premises, conclusion 0
- **r2**: no premises, conclusion 1
- **r3**: premise x, conclusion x0
- **r4**: premise x, conclusion x1

\[ R = \{ r1,...r4 \} \]

Claim: Set theory\((R)\) = Set of binary numbers

Other notation for rules: \( \frac{\text{premise}}{\text{conclusion}} \)

Our four rules in new notation:

\[
\begin{array}{c}
0 \\
1 \\
x0 \\
x1
\end{array}
\]

Is the sentence 1101 in the set theory\((R)\)? For a sentence to be in the set, we need to be able to derive it using the rules from \(R\):

\[
\frac{x}{1} \quad \frac{x0}{110} \quad \frac{x1}{1101}
\]

1101 therefore is a valid syntactic expression of our language. We now define a semantics of the language:

\[
\text{[Binary number]} \rightarrow \text{Decimal number}
\]

If the syntax is defined by rules, then semantics can be defined inductively.

- **r1**: \([0] = 0\)
- **r2**: \([1] = 1\)
- **r3**: \([x0] = 2*[x]\)
- **r4**: \([x1] = 2*[x]+1\)

With semantics defined, we can translate 1101 into a semantic object:

\[
[1101] = 2*[110] + 1 = 2*(2*[11]) + 1 = 2*(2*(2*[1]+1)) + 1 = 2*(2*(2*1+1)) + 1 = 13
\]

1.2 Example 2: Language of arithmetic expressions

Alphabet: \{natural numbers, +,\cdot\}

Let \(n\) be a natural number. We can then form rules:

- **r1**: \(n\)
We define the semantics of the language as follows:

\[
\begin{align*}
[n] &= n \\
[x + y] &= [x] + [y] \\
[x * y] &= [x] * [y]
\end{align*}
\]

Although this set of rules looks logical, it has one important flaw. Namely, when we try to derive a simple arithmetic expression \(2 + 3 \cdot 4\), we see that there are two possible derivations:

\[
\begin{align*}
\text{(r1)} & \quad 2 \\
\text{(r2)} & \quad 3 \\
\text{(r3)} & \quad 4 \\
\text{(r1)} & \quad 2 + 3 \\
\text{(r2)} & \quad 3 \\
\text{(r3)} & \quad 4
\end{align*}
\]

Sets of rules that can produce several derivation of the same expressions are called ambiguous and should be avoided. There are several ways to solve this problem:

1.2.1 Solution 1: Sentences as trees

We can define a language in which arithmetic expressions are written in the form of trees:

\[
\begin{align*}
\text{r1: } & - \\
\text{r2: } & \frac{x}{y} \\
\text{r3: } & \frac{x}{y}
\end{align*}
\]

The derivation of our expression then is:

\[
\begin{align*}
\text{(r1)} & \quad 2 \\
\text{(r2)} & \quad 3 \\
\text{(r3)} & \quad 4 \\
\text{(r1)} & \quad 2 + 3 \\
\text{(r3)} & \quad 4 \\
\end{align*}
\]

This set of rules is no longer ambiguous, but requires arithmetic expressions represented in the form of trees, which is not how we typically write them.
1.2.2 Solution 2: Enforcing parentheses

Instead of representing expressions in the form of trees, we can use rules with parentheses:

\[ \begin{align*}
  & r_1: \quad n \\
  & r_2: \quad \frac{x \ y}{(x) + (y)} \\
  & r_3: \quad \frac{x \ y}{(x) \cdot (y)}
\end{align*} \]

This solution requires our expression to be in the form \((2) + ((3)+4))\), which seems more natural than trees, but requires unnecessary parentheses.

1.2.3 Solution 2: Types

In formal languages, we can classify words into different types. (Just as in natural languages we have types as verbs, nouns etc.). Using types, we can form less ambiguous sets of rules.

We will introduce three types: Sentence, Product, Sum:

Rules for sentences: 
\[ \begin{align*}
  & r_1: \quad n : \text{sent} \\
  & r_2: \quad \frac{x : \text{prod}}{(x) : \text{sent}} \\
  & r_3: \quad \frac{x : \text{sum}}{(x) : \text{sent}}
\end{align*} \]

Rules for sums: 
\[ \begin{align*}
  & r_4: \quad x : \text{sent} \\
  & r_5: \quad \frac{x \ y : \text{sent}}{x + y : \text{sum}} \\
  & r_6: \quad x : \text{sum} \\
  & r_7: \quad \frac{x \ y : \text{sent}}{x \cdot y : \text{prod}}
\end{align*} \]

The derivation of our arithmetic expression is as follows:

\[ \frac{\frac{n : \text{sent}}{(r_1)}}{\frac{2 + 3 : \text{sum}}{(r_3)}} \frac{\frac{(2 + 3) : \text{sent}}{(r_7)}}{\frac{(2 + 3) \cdot 4 : \text{prod}}{(r_2)}} \frac{\frac{(2 + 3) \cdot 4 : \text{sum}}{(r_1)}}{4 : \text{sent}} \]

These 7 rules are not ambiguous, but still require parentheses around sums for correct translation into semantic objects.

Take-home question: How can we modify the rules so that the parentheses are not required?

The semantics should reflect the types of the rules, for example:

\[ [x \ast y : \text{prod}] = [[x : \text{sent}] \ast [y : \text{sent}]] \]
\[ [[(x) : \text{sent}] = [x : \text{prod}] \]
\[ [[[x) : \text{sent}] = [x : \text{sum}] \]
\[ \ldots \]