1 Rule Induction

Below are the proofs done in class, written out formally as you would write them on a homework.

First, here are the rules for our “is XML” judgment.

\[
\begin{align*}
\text{""} & \text{ is XML } \quad \text{(X-1)} \\
\langle T \rangle & \text{ is XML } \quad \text{(X-2)} \\
X & \text{ is XML } \quad \text{(X-3)} \\
X & \text{ is XML } \quad \text{Y is XML } \quad & \text{ X Y is XML } \quad \text{(X-4)}
\end{align*}
\]

Let OpenAngle(X) be the number of open angle brackets in a string X and CloseAngle(X) be the number of close angle brackets in X.

**Theorem 1.** If X is XML, then OpenAngle(X) = CloseAngle(X).

**Proof.** By induction on the derivation of X is XML.

- Case X-1. Then X = "" and OpenAngle(X) = CloseAngle(X) = 0.
- Case X-2. Then X = ⟨T⟩ and OpenAngle(X) = CloseAngle(X) = 1.
- Case X-3. Then X = ⟨T⟩⟨Y⟩ and Y is XML. We have OpenAngle(X) = 2 + OpenAngle(Y) and CloseAngle(X) = 2 + CloseAngle(Y). By induction, OpenAngle(Y) = CloseAngle(Y), so 2 + OpenAngle(Y) = 2 + CloseAngle(Y).
- Case X-4. Then X = YZ and Y is XML and Z is XML. We have OpenAngle(X) = OpenAngle(Y) + OpenAngle(Z) and CloseAngle(X) = CloseAngle(Y) + CloseAngle(Z). By induction, OpenAngle(Y) = CloseAngle(Y) and OpenAngle(Z) = CloseAngle(Z), so OpenAngle(X) = CloseAngle(X).

\[\square\]

Now here are the rules for natural numbers (both constructing a natural number and the greater-than judgment)

\[
\begin{align*}
\text{Zero} & \quad \text{n is a natural number } \quad \text{(zero)} \\
\text{succ} & \quad \text{n + 1 is a natural number } \quad \text{(succ)} \\
\text{ge-nat} & \quad \text{n \geq n } \quad \text{(ge-nat)} \\
\text{ge-succ} & \quad \text{m \geq n } \quad \text{(ge-succ)}
\end{align*}
\]

**Theorem 2.** If n is a natural number then n \geq 0.

**Proof.** By induction on the derivation of n is a natural number.

- Rule zero. Then n = 0. By ge-nat, we have n \geq 0.
- Rule succ. Then n = m + 1 and m is a natural number. By induction, m \geq 0. By ge-succ, we have m + 1 \geq 0.

\[\square\]
2 E Language

2.1 Syntax

We will be working with a small language called \( E \) consisting of integer and string expressions. The grammar below is in BNF (Backus-Naur Form). We use \( e ::= A \mid B \mid \ldots \) to mean that an expression (with metavariable \( e \)) can look like form \( A \) or \( B \), and so on.

\[
e ::= \begin{cases} n & \text{Numbers} \\ "s" & \text{Strings} \\ e + e & \text{Addition} \\ e \cdot e & \text{Concatenation} \\ |e| & \text{String Length} \end{cases}
\]

2.2 Small-step semantics

We also studied the small-step semantics of \( E \). There are two judgments, \( e \text{ val} \) meaning that \( e \) is a value and can’t step anymore, and \( e \mapsto e’ \) meaning that \( e \) steps to \( e’ \). The rules for these judgments are below. Note that in rules S-1 and S-3, when we do \( n_1 + n_2 \) or \(|s|\) (as opposed to \( \overline{n_1} + \overline{n_2} \) and \(|"s"|\)), these are actually taking the mathematical addition of two integers and the actual number of characters in a string literal.

We just use the same symbols for the actual underlying operation and for the syntax of the programming language. Rules S-4 through S-8 are “search” rules that allow us to step subexpressions.

\[
\begin{align*}
(V-1) & \quad \overline{n} \text{ val} \\
(V-2) & \quad "s" \text{ val} \\
(S-1) & \quad n_1 + n_2 \mapsto n_1 + n_2 \\
(S-2) & \quad "s_1" \cdot "s_2" \mapsto "s_1s_2" \\
(S-3) & \quad |"s"| \mapsto |s| \\
(S-4) & \quad e_1 \mapsto e'_1 \\
(S-5) & \quad e_2 \mapsto e'_2 \\
(S-6) & \quad e_1 \cdot e_2 \mapsto e'_1 \cdot e'_2 \\
(S-7) & \quad "s_1" \cdot e_2 \mapsto "s_1" \cdot e'_2 \\
(S-8) & \quad |e| \mapsto |e'| 
\end{align*}
\]