Chapter 1: Syntax and Semantics

1. Consider the language of binary numerals introduced in Section 1.1. Construct an abstract syntax tree for the expression $1 + 11 + 1 \times 10$. Do the usual precedence rules for $\times$ and $+$ (“$\times$ binds stronger than +”) allow only one such tree? If not, give all possible trees.

2. Define the abstract syntax of a language of decimal numerals. This language has a domain $Digit$ whose elements are the decimal digits 0, 1, …, 9, and a domain $Num$ whose elements are non-empty sequences of decimal digits, such as 0, 99, or 271. Give this language a semantics by defining functions $[\cdot]: Digit \to \mathbb{N}$ and $[\cdot]: Num \to \mathbb{N}$ that map digits respectively numerals to natural numbers.

3. Define the abstract syntax of a language with a single domain $Exp$ of arithmetic expressions with constants 0 and 1 and addition, negation, subtraction, multiplication, and division. Give this language a semantics by defining a function $[\cdot]: Exp \to \mathbb{Q} \cup \{\text{nan}\}$ that maps every expression to a rational number or to the special constant nan (“not a number”). This special value is the result of division by zero or the result of any operation whose operand is not a number; thus we have, e.g., $[1 + (1/(1 - 1))] = \text{nan}$.

4. Consider the language of binary numerals introduced in Section 1.1. Define by structural induction a function $[\cdot]: Numeral \to Expression$ that syntactically “simplifies” numerals by removing leading occurrences of 0, e.g., $[00010] = 10$. Based on this function, define a function $[\cdot]: Expression \to Expression$ that syntactically “simplifies” arithmetic expressions by considering the equational laws $n + 0 = 0 + n = n$, $0 \cdot n = n \cdot 0 = 0$, and $1 \cdot n = n \cdot 1 = n$. For example, we have $[1 \cdot 000 + 011] = 11$.

5. Define the abstract syntax of a language of number and list expressions. This language has a single domain $Exp$ with constants 0 and 1 and the usual operations for addition and multiplication (these expressions denote natural numbers); furthermore, the domain has constant nil (the empty list), a binary function cons (which prepends a number to a list), a unary function head (which returns the first number of a list), and a unary function tail (which returns the remainder of a list). Give this language a type system with judgements $E : \text{num}$ (“$E$ is a number expression”) and $E : \text{list}$ (“$E$ is a list expression”). Show the derivation of the judgement $\text{head(tail(cons(1,cons(1+1,nil))))} : \text{num}$.

6. Define the abstract syntax of a numeric expression language with three domains $Ident$, $Decl$ and $Exp$. The domain $Ident$ contains infinitely many identifiers that are not further specified. The domain $Decl$ consists of sequences of definitions of form $I_1 = E_1, \ldots, I_n = E_n$ with $n \geq 1$ (this domain is modeled by one constructor that constructs a sequence of a single declaration $I = E$ and a constructor that adds such a declaration to another sequence). The domain $Exp$ is constructed from constants 0 and 1, identifiers, operations for addition and multiplication and a “block expression” let $D$ in $E$ where $D$ is a declaration sequence and $E$ is an expression. An example expression is let $I_1 = I_0 + 1, I_2 = I_1 \times 1$ in $I_0 + I_1 \times I_2$. 

Wolfgang Schreiner

March 31, 2021
7. Consider the numeric expression language of Exercise 6. Give this language a type system with the judgements $I_0 \vdash D : \text{dec}(I_0)$ (“$D$ is a well-formed list of declarations that extends the set of declared identifiers $I_0$ to the set $I_0'$”) and $I_0 \vdash E : \exp$ (“given the set of declared identifiers $I_0$, $E$ is a well-formed expression”). Show how by this type system the judgement $\{I_0\} \vdash \mathrm{let} \ I_1 = I_0 + 1, I_2 = I_1 \times 1 \in I_0 + I_1 \times I_2 : \exp$ can be derived.

8. Define the abstract syntax of a language whose phrases are “bit matrices” of arbitrary dimension. In detail, this language has a domain $\mathit{Bit}$ whose only values are the constants 0 and 1. The domain $\mathit{Row}$ consists of finite sequences $[b_1, \ldots, b_m]$ of of $m \geq 1$ bits and the domain $\mathit{Matrix}$ consists of finite sequences $[r_1, \ldots, r_n]$ of $n \geq 1$ rows. Give this language a type system with a judgement $\mathit{r} : \mathit{row}(m)$ (“$\mathit{r}$ is a row of length $m$”) and $\mathit{m} : \mathit{matrix}(n,m)$ (“$\mathit{m}$ is a matrix with $n$ rows and $m$ columns”). Show how the judgement $[[0,1,0],[1,1,0]] : \mathit{matrix}(2,3)$ can be derived. Give this language a semantics that determines the number of bits in a row respectively matrix by functions $\exp : \mathit{Row} \rightarrow \mathbb{N}$ and $\exp : \mathit{Matrix} \rightarrow \mathbb{N}$ such that $\exp([1,1,0]) = 2$ and $\exp([[0,1,0],[1,1,0]]) = 3$.

9. Define the abstract syntax of a language with a single domain $\exp$ of arithmetic expressions with constants 1 and 1.0 and addition, negation, subtraction, multiplication, and division. Give this language a type system with two judgements $e : \mathit{int}$ and $e : \mathit{real}$ interpreted as “$e$ is an integer expression” and “$\mathit{r}$ is a real expression”, respectively; this type system has axioms $\mathit{int}$ and $\mathit{real}$; its rules assign to to the result of any operation an integer type only if all operands are integer expressions (otherwise the result is a real expression).

Give this language a denotational semantics by defining a function $\exp : \exp \rightarrow \mathbb{R} \cup \{\mathit{nan}\}$. Prove by rule induction that, if we can derive $e : \mathit{int}$, then we indeed have $\exp(e) \in \mathbb{Z} \cup \{\mathit{nan}\}$ (see Exercise 3 for the interpretation of $\mathit{nan}$).

10. Define the abstract syntax of a language of a hand-held calculator by which the user can evaluate a sequence of arithmetic expressions. This language has a domain $\exp$ of arithmetic expressions that contains the constants 0, 1, the constant $\mathit{\$}$, and addition and multiplication (here $\mathit{\$}$ represents the value of the previously evaluated expression, more on this below). Furthermore, there is a domain $\mathit{Seq}$ that contains all expression sequences of form $E_1; \ldots; E_n$ where $n \geq 0$ (this domain is modeled by the empty sequence constructor $\_\_\_\_\_\_\_\$ and the constructor $E; Es$ that prepends expression $E$ to sequence $Es$).

Give this language a semantics by defining a function $\exp : \exp \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ that maps an expression to a function over the natural numbers. Here an application $\exp(E)(n)$ receives the value $n$ of the expression that was evaluated immediately before $E$ (i.e., $n$ is the value of $\mathit{\$}$) and returns the value of $E$. Likewise define a function $\exp : \exp \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ that maps an expression sequence to a function from the natural numbers to a sequence of such numbers. Here $\exp(Es)(n)$ receives the value $n$ of the expression that was evaluated immediately before $Es$ and returns the values of the expressions in the sequence. Thus we have, e.g., the evaluation $\exp(1+\mathit{1\$}; (1+\mathit{1\$})\times\mathit{\$}; 1+\mathit{\$})(0) = [2, 6, 7]$ (which provides the initial value 0 for constant $\mathit{\$}$).