Reminder of Important Definitions

Grammars

A *Context Free Grammar (CFG)* consists of four components:

- An alphabet of *terminal* symbols.
- A finite, non-empty set of *non-terminal* symbols, disjoint from the terminals.
- A finite set of *production rules*.
- A designated non-terminal called the *start symbol*.

A *sentential form*, usually α, β, γ and so on, is just a finite sequence of terminals and nonterminals.

The sentential form α can make a **derivation step** to β, written $\alpha \to \beta$, just if:

- α has shape $\gamma_1 \, X \, \gamma_2$ and β has shape $\gamma_1 \, \delta \, \gamma_2$
- and there is a production rule $X ::= \delta$ in the grammar

A *derivation sequence* is a non-empty sequence of sentential forms α_1, α_2, ... α_{k-1}, α_k in which consecutive elements of the sequence are derivation steps:

$$\alpha_1 \to \alpha_2 \to \cdots \to \alpha_{k-1} \to \alpha_k$$

A sentential form β is *derivable* from α, written $\alpha \to^* \beta$ just if there is a derivation sequence starting with α and ending with β.

We say that a word w is in the *language of a grammar* G with start symbol S, and write $w \in L(G)$ just if $S \to^* w$.

Nullable

On nonterminals:

$$\mathsf{Nullable}(X) \text{ iff } X \to^* \epsilon$$

On sentential forms:

$$\mathsf{Nullable}_s(\alpha) = \begin{cases} \mathsf{true} & \text{if } \alpha = \epsilon \\ \mathsf{false} & \text{if } \alpha \text{ is of shape } a\beta \\ \mathsf{Nullable}(X) \wedge \mathsf{Nullable}_s(\beta) & \text{if } \alpha \text{ is of shape } X\beta \end{cases}$$

First

On nonterminals:

$$\mathsf{First}(X) = \{a \mid \exists \beta. \, X \to^* a\beta\}$$

On sentential forms:

$$\mathsf{First}_s(\alpha) = \begin{cases} \emptyset & \text{if } \alpha = \epsilon \\ \{a\} & \text{if } \alpha \text{ is of shape } a\beta \\ \mathsf{First}(X) & \text{if } \alpha \text{ is of shape } X\beta \text{ and } \neg\mathsf{Nullable}(X) \\ \mathsf{First}(X) \cup \mathsf{First}_s(\beta) & \text{if } \alpha \text{ is of shape } X\beta \text{ and } \mathsf{Nullable}(X) \end{cases}$$

Follow

On nonterminals:

$$\mathsf{Follow}(X) = \{a \mid \exists \alpha\beta.\, S \to^* \alpha X a \beta\}$$

Parse Tables and LL(1)

We define the *parsing table*, usually T, for a given grammar as a 2d array indexed by pairs of a nonterminal and a terminal. Each entry $T[X, a]$ is a set of production rules from the grammar, such that some rule $X \longrightarrow \beta$ is in the set $T[X, a]$ just if, either:

1. $a \in \mathsf{First}_s(\beta)$
2. or, $\mathsf{Nullable}_s(\beta)$ and $a \in \mathsf{Follow}(X)$

A grammar whose parse table contains at most one rule in each cell is called *LL(1)*.

Abstract Syntax of arithmetic expressions

An *arithmetic expression* is a tree described by the following grammar:

$$A \longrightarrow n \mid x \mid A + A \mid A - A \mid A * A$$

where n ranges over integer literals, and x ranges over variables. Parentheses are used to resolve ambiguity and to indicate the structure of the tree. We write $\mathcal{A}$ for the set of arithmetic expressions.

Abstract Syntax of Boolean expressions

A *Boolean expression* is a tree described by the following grammar.

$$B \longrightarrow \mathsf{false} \mid \mathsf{true} \mid !B \mid B \mathbin{\&\&} B \mid B \parallel B \mid A = A \mid A \leq A$$

Parentheses are used to resolve ambiguity and to indicate the structure of the tree. We write $\mathcal{B}$ for the set of Boolean expressions.

Abstract Syntax of statements

A *statement* is a tree described by the following grammar:

$$S \longrightarrow \mathsf{skip} \mid x \leftarrow A \mid S; S \mid \mathsf{if}\ B\ \mathsf{then}\ S\ \mathsf{else}\ S \mid \mathsf{while}\ B\ S$$

Braces "$\{\cdots\}$" are used to resolve ambiguity and to indicate the structure of the tree. We write $\mathcal{S}$ for the set of statements.

While Program Semantics

A *state* is a total function from the set $\mathsf{State} = \mathsf{Var} \to \mathbb{Z}$, where Var is the set of variables. We write $[x_1 \mapsto v_1, x_2 \mapsto v_2, \ldots, x_n \mapsto v_n]$ to indicate the state that maps the variable $x_i \in \mathsf{Var}$ to the value $v_i \in \mathbb{Z}$ for all $i \leq n$. By convention, any variable not explicitly mentioned by a given state σ is assigned the value 0.

For a given state $\sigma \in \mathsf{State}$, we write $\sigma[x \mapsto v]$ for some variable $x \in \mathsf{Var}$ and $v \in \mathbb{Z}$ to denote the state that maps the variable x to v and any other variable y to the value $\sigma(y)$.

Semantics of arithmetic expressions

The denotation function for arithmetic expressions $[\![\cdot]\!]_{\mathcal{A}} \in \mathcal{A} \to (\mathsf{State} \to \mathbb{Z})$, which is defined by recursion in Figure 1. We say that two arithmetic expressions $e_1, e_2 \in \mathcal{A}$ are *semantically equivalent* if, and only if, $[\![e_1]\!]_{\mathcal{A}}(\sigma) = [\![e_2]\!]_{\mathcal{A}}(\sigma)$ for all states $\sigma \in \mathsf{State}$.

$$
\begin{aligned}
[\![n]\!]_{\mathcal{A}}(\sigma) &= n \\
[\![x]\!]_{\mathcal{A}}(\sigma) &= \sigma(x) \\
[\![e_1 + e_2]\!]_{\mathcal{A}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) + [\![e_2]\!]_{\mathcal{A}}(\sigma) \\
[\![e_1 - e_2]\!]_{\mathcal{A}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) - [\![e_2]\!]_{\mathcal{A}}(\sigma) \\
[\![e_1 * e_2]\!]_{\mathcal{A}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) \cdot [\![e_2]\!]_{\mathcal{A}}(\sigma)
\end{aligned}
$$

Figure 1: Definition of the denotational semantics of arithmetic expressions.

Semantics of Boolean expressions

The denotation function for Boolean expressions $[\![\cdot]\!]_{\mathcal{B}} \in \mathcal{B} \to (\mathsf{State} \to \mathbb{B})$ is defined by recursion in Figure 2. We say that two Boolean expressions e_1, $e_2 \in \mathcal{B}$ are *semantically equivalent* if, and only if, $[\![e_1]\!]_{\mathcal{B}}(\sigma) = [\![e_2]\!]_{\mathcal{B}}(\sigma)$ for all states $\sigma \in \mathsf{State}$.

$$
\begin{aligned}
[\![\mathsf{false}]\!]_{\mathcal{B}}(\sigma) &= \bot \\
[\![\mathsf{true}]\!]_{\mathcal{B}}(\sigma) &= \top \\
[\![!e]\!]_{\mathcal{B}}(\sigma) &= \neg [\![e]\!]_{\mathcal{B}}(\sigma) \\
[\![e_1 \;\&\&\; e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{B}}(\sigma) \wedge [\![e_2]\!]_{\mathcal{B}}(\sigma) \\
[\![e_1 \parallel e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{B}}(\sigma) \vee [\![e_2]\!]_{\mathcal{B}}(\sigma) \\
[\![e_1 = e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) = [\![e_2]\!]_{\mathcal{A}}(\sigma) \\
[\![e_1 \leq e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{A}}(\sigma) \leq [\![e_2]\!]_{\mathcal{A}}(\sigma)
\end{aligned}
$$

Figure 2: Definition of the denotational semantics of Boolean expressions.

Semantics of statements

The operational semantics relation $\Downarrow \subseteq \mathcal{S} \times \mathsf{State} \times \mathsf{State}$ is defined inductive by the rules in Figure 3. We say that two statements S_1, $S_2 \in \mathcal{S}$ are *semantically equivalent* if, and only if:

$$S_1,\, \sigma_1 \Downarrow \sigma_2 \Leftrightarrow S_2,\, \sigma_1 \Downarrow \sigma_2$$

for any two states σ_1, $\sigma_2 \in \mathsf{State}$.

$$\frac{}{\mathsf{skip},\, \sigma \Downarrow \sigma} \qquad \frac{}{x \leftarrow e,\, \sigma \Downarrow \sigma[x \mapsto [\![e]\!]_{\mathcal{A}}(\sigma)]}$$

$$\frac{S_1,\, \sigma_1 \Downarrow \sigma_2 \quad S_2,\, \sigma_2 \Downarrow \sigma_3}{S_1; S_2,\, \sigma_1 \Downarrow \sigma_3} \qquad \frac{S_1,\, \sigma_1 \Downarrow \sigma_2}{\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2,\, \sigma_1 \Downarrow \sigma_2}\, [\![e]\!]_{\mathcal{B}}(\sigma_1) = \top$$

$$\frac{S_2,\, \sigma_1 \Downarrow \sigma_2}{\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2,\, \sigma_1 \Downarrow \sigma_2}\, [\![e]\!]_{\mathcal{B}}(\sigma_1) = \bot \qquad \frac{}{\mathsf{while}\ e\ \mathsf{do}\ s,\, \sigma \Downarrow \sigma}\, [\![e]\!]_{\mathcal{B}}(\sigma) = \bot$$

$$\frac{S,\, \sigma_1 \Downarrow \sigma_2 \quad \mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma_2 \Downarrow \sigma_3}{\mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma_1 \Downarrow \sigma_3}\, [\![e]\!]_{\mathcal{B}}(\sigma_1) = \top$$

Figure 3: Definition of the operational semantics of statements.

Computable Functions

We write $[\mathtt{x} \mapsto n]$ for the state that maps the variable $\mathtt{x}$ to the number $n \in \mathbb{N}$, and every other variable to 0.

A 'while' program $\mathtt{S}$ *computes* a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ (with respect to $\mathtt{x}$) just if $f(m) \simeq n$ exactly when $\langle S, [\mathtt{x} \mapsto m] \rangle \Downarrow [\mathtt{x} \mapsto n]$.

A function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is *computable* just if there is a program S that computes f with respect to the variable $\mathtt{x}$.

Predicates

The *characteristic function* of U is the function

$$\chi_U : \mathbb{N} \to \mathbb{N}$$
$$\chi_U(n) = \begin{cases} 1 & \text{if } n \in U \\ 0 & \text{if } n \notin U \end{cases}$$

The *semi-characteristic function* of U is the partial function

$$\xi_U : \mathbb{N} \rightharpoonup \mathbb{N}$$
$$\xi_U(n) \begin{cases} \simeq 1 & \text{if } n \in U \\ \uparrow & \text{otherwise} \end{cases}$$

A predicate $U \subseteq \mathbb{N}$ is *decidable* just if its characteristic function $\chi_U : \mathbb{N} \to \mathbb{N}$ is computable.

The 'while' program that computes the characteristic function χ_U of a predicate $U \subseteq \mathbb{N}$ is called a *decision procedure*. Any predicate for which there is no decision procedure is called *undecidable*.

A predicate $U \subseteq \mathbb{N}$ is *semi-decidable* just if its semi-characteristic function ξ_U is computable.

The *Halting Problem* is the following predicate:

$$\mathsf{HALT} = \{\langle \ulcorner S \urcorner, n \rangle \mid [\![S]\!]_{\mathtt{x}}(n) \downarrow\}$$

Bijections

A function $f : A \to B$ is *injective* (or 1-1) just if for any $a_1, a_2 \in \mathcal{A}$ we have that $f(a_1) = f(a_2)$ implies $a_1 = a_2$. We sometimes write $f : A \rightarrowtail B$ whenever f is an injection.

A function $f : A \to B$ is *surjective* just if for any $b \in \mathcal{B}$ there exists $a \in \mathcal{A}$ such that $f(a) = b$. We sometimes write $f : A \twoheadrightarrow B$ whenever f is a surjection.

A function $f : A \to B$ is a *bijection* just if it is both injective and surjective.

Let $f : A \to B$ be a function. f is an *isomorphism* just if it has an *inverse*. That is, if there exists a function $f^{-1} : B \to A$ such that:

- for all $a \in \mathcal{A}$ we have $f^{-1}(f(a)) = a$
- for all $b \in \mathcal{B}$ we have $f(f^{-1}(b)) = b$

Encoding Data

A *pairing function* is a bijection $\mathbb{N} \times \mathbb{N} \xrightarrow{\cong} \mathbb{N}$. We assume that we have a fixed pairing function

$$\langle -, - \rangle : \mathbb{N} \times \mathbb{N} \xrightarrow{\cong} \mathbb{N}$$

with the following inverse:

$$\mathsf{split} : \mathbb{N} \xrightarrow{\cong} \mathbb{N} \times \mathbb{N}$$

Reflections

Suppose we have two bijections:

$$\phi : A \xrightarrow{\cong} \mathbb{N} \quad \psi : B \xrightarrow{\cong} \mathbb{N}$$

The *reflection* of $f : A \rightharpoonup B$ under (ϕ, ψ) is the function

$$\tilde{f} : \mathbb{N} \rightharpoonup \mathbb{N}$$
$$\tilde{f}(n) = \psi(f(\phi^{-1}(n)))$$

Gödel Numbering

Let **Stmt** be the set of Abstract Syntax Trees of While. We assume that we have a Gödel numbering

$$\ulcorner - \urcorner : \mathbf{Stmt} \xrightarrow{\cong} \mathbb{N}$$

which encodes While programs as natural numbers.

A *code transformation* is a function $f : \mathbf{Stmt} \to \mathbf{Stmt}$.

Universal Function

The *universal function*, U, is defined as follows:

$$U : \mathbf{Stmt} \times \mathbb{N} \rightharpoonup \mathbb{N}$$
$$U(P, n) = [\![P]\!]_{\mathtt{x}}(n)$$

Reductions

Let $U, W \subseteq \mathbb{N}$ be predicates, and let $f : \mathbb{N} \to \mathbb{N}$. The function f is a *many-one reduction* from U to W just if it is computable, and it is also the case that

$$n \in U \Leftrightarrow f(n) \in W$$

We may write $f : U \lesssim V$ (read "f is a reduction from U to V").