Symbols

We assume we have an infinite number of symbols available and write $x \, sym$ to assert that $x$ is a symbol.

Symbols are sometimes called names or atoms or identifiers.

For $x \, sym$ and $y \, sym$, the judgement $x \neq y$ says that $x$ and $y$ are distinct symbols.

Characters and Alphabets

An alphabet $\Sigma$ is a countable set of symbols or characters.

- ASCII or Unicode.
- Binary digits.

The judgement $c \, char$ indicates that $c$ is a character, and $\Sigma$ stands for a finite set of such judgements.

Strings

The set $\Sigma^*$ consists of all finite strings over $\Sigma$, as specified by the judgement $\Sigma \vdash s \, str$.

- Null string $\varepsilon$.
- Single-character string: $c$, where $c \in \Sigma$.
- Append/Juxtaposition: $c \cdot s$, where $c \, char$ and $s \, str$.
- Concatenation: $s_1 \cdot s_2$ where $s_1 \, str$ and $s_2 \, str$.

Inductive Definition of Strings

The judgement $\Sigma \vdash s \, str$ is inductively defined by the following rules:

$$
\frac{}{\Sigma \vdash \varepsilon \, str}
\quad
\frac{\Sigma \vdash c \, str \quad \Sigma \vdash s \, str}{\Sigma \vdash c \cdot s \, str}
$$

The judgement $s \, str$ is relative to the alphabet specified via a hypothetical judgement of the form:

$$
c_1 \, char, \ldots, c_n \, char \vdash s \, str
$$

which we abbreviate as $\Sigma \vdash s \, str$

Explicit mention of $\Sigma$ is often suppressed when it is clear from context.

String Induction

The principle of rule induction specializes to string induction.

Specifically, we can prove that every string $s$ has a property $P$ by showing that

- $\varepsilon$ has property $P$;
- if $s$ has property $P$ and $c \, char$, then $c \cdot s$ has property $P$.

Essentially induction on string length without making length explicit.
Example: String Concatenation

Consider the inductive definition of string concatenation:

\[ \varepsilon S c \cdot s c \cdot d \cdot s t \]

By string induction on the first argument, can prove that the judgement form \( s_1^\ast s_2 = s \) has mode \((\forall, \forall, \exists)\).

That is, the above rule scheme defines a total function.

String Notation

A string can be written in various ways, as is convenient:

- Juxtaposition of characters: \( abed \) for \( a \cdot b \cdot c \cdot \varepsilon \).
- Juxtaposition for concatenation: \( abed \) for \( ab \cdot cd \).
- Any possible concatenation: \( ab \cdot cd \) or \( a \cdot bed \) or \( ab \cdot ed \cdot c \) or \( \ldots \).

Abstract Syntax Trees

Let \( O \) be a countable set of operators.

Let \( \Omega : O \rightarrow \mathbb{N} \) be an assignment of arities to operators, i.e., \( \text{ar}(o) = k \) where \( o \) sym and \( k \) nat. Such an \( \Omega \) is called a signature.

- Arty = number of arguments.
- 0-ary = no arguments.
- If \( \Omega \vdash \text{ar}(o) = k \) and \( \Omega \vdash \text{ar}(o) = k' \) then \( k = k' \) nat

AST’s (Terms)

Examples:

- \( \text{zero} \) has arity 0, \( \text{succ}(\varepsilon) \) has arity 1.
- \( \text{empty} \) has arity 0, \( \text{add}(\varepsilon, \varepsilon) \) has arity 2.

So, for example, \( \text{add}(\text{empty}, \text{empty}) \) is an ast.

Inductive Definition of Abstract Syntax Trees

The judgement \( o(a_1, \ldots, a_k) \) ast is inductively defined by the following rules:

\[
\begin{align*}
\Omega \vdash \text{ar}(o) &= k \\
\vdash a_1 \text{ ast} & \quad \ldots \quad \vdash a_k \text{ ast} \\
\vdash o(a_1, \ldots, a_k) \text{ ast}
\end{align*}
\]

The base case is for operators of arity \( \text{zero} \), in which case the rule has no premises.
Structural Induction

The principle of rule induction specializes to structural induction when applied to abstract syntax trees over signature $\Omega$

Specifically, we can prove that a AST has a property $P$ by showing for each operator $o$ in $O$ that:

- Given that $\Omega \vdash ar(o) = k$
- If $a_1$ ast has property $P$ and ... and $a_k$ ast has property $P$, then $o(a_1, ..., a_k)$ ast has property $P$.

When $k$ is zero, reduces to showing $o$ has property $P$.

Example: AST Height

Consider the inductive definition of the height of an abstract syntax tree:

$$\text{hgt}(a_1) = n_1 \quad \ldots \quad \text{hgt}(a_k) = n_k \quad \text{hgt}(\langle a_1, \ldots, a_k \rangle) = \text{succ}(n)$$

By structural induction we can prove that the judgement has mode $(\forall, \exists)$.

That is, the above rule scheme defines a total function.

Variables and Substitution

We often wish to have variables in abstract syntax trees that will stand for other abstract syntax trees. A variable will be instantiated by substituting an AST for instances of that variable in another AST.

Use of variables in an AST over signature $\Omega$ can be described using a hypothetical judgement such as:

$$a_1 \text{ ast}, \ldots, a_k \text{ ast} \vdash o \text{ ast}$$

where the $a_1, \ldots, a_k$ are pairwise distinct symbols.

Inductive Definition of Substitution

We define the judgement $[a/x]\ b = c$, meaning that $c$ is the result of substituting $a$ for $x$ in $b$ by rules (one for each operator declared in $\Omega$) of the following form:

$$\frac{\Omega \vdash ar(o) = k \quad [a/x]b_1 = c_1 \ldots [a/x]b_k = c_k \quad [a/x]o(b_1, \ldots, b_k) = o(c_1, \ldots, c_k)}{[a/x]\ b}$$

Lemma 1 (Substitution)

If $x_1 \text{ ast}, \ldots, x_k \text{ ast} \vdash a \text{ ast}$ and $x_1 \text{ ast}, \ldots, x_k \text{ ast} \vdash a \text{ ast}$, then there exists a unique $c$ such that

$$[a/x]x_1 = x_1 \ldots [a/x]x_k = x_k \quad [a/x]a = a \vdash [a/x]\ b = c$$

Proof: The proof proceeds by structural induction.

Since the result of substitution is unique, we let $[a/x]\ b$ stand for the unique $c$ such that $[a/x]\ b = c$.

Simultaneous substitution, written $[a_1, \ldots, a_k/x_1, \ldots, x_k] b$, is defined similarly.
Abstract Binding Trees

Abstract binding trees enrich abstract syntax trees with the concepts of binding and scope.

We add a notion of fresh or new names for use within a specified scope.

We also introduce the notions of $\alpha$-equivalence and capture-avoiding substitution.

Binding and Scope in English

Pronouns and demonstratives are analogous to bound variables.

- He, she, it, this, that refer to a noun introduced elsewhere.
- Linguistic conventions establish scope and binding of pronouns and demonstratives.

Natural languages have only a small, fixed number of variables! Confusion is possible.

Binding and Scope in Arithmetic

Add to our language of arithmetic expressions the ability to

- Bind a variable to an expression in a given scope.
- Refer to (the value of) that expression.

There is an unlimited supply of variables. There will be no possibility for confusion.

Abstract Binding Trees

The signature for abstract binding trees specifies the arities of the operators, which include both the number of arguments to the operator and the number of bound names, or valence in each argument.

A abstract binding tree signature $\Omega$ consists of a finite set of judgements of the form $ar(a) = (n_1, \ldots, n_k)$, where $n_i$ nat

For example, the arity $(0, \ldots, 0)$ of length $k$ specifies an operator taking $k$ arguments that bind no variables, and hence is the analog of the arity for an abstract syntax tree operator taking $k$ arguments.

Abstract Binding Trees

Well-formed abstract binding trees over a signature $\Omega$ are specified by a parametric hypothetical judgement of the form:

$$\{x_1, \ldots, x_k\}x_1 \text{ abt}\cdots x_k \text{ abt} \vdash a \text{ abt}$$

The above says that $a$ is an abstract binding tree of valence $n$ with parameters or free names $x_1, \ldots, x_k$.

We sometimes use $a \text{ abt}$ as shorthand for $a \text{ abt}$. 
Abstract Binding Trees

Abstract binding trees inherently include variables. We let \( X \) stand for the parameter (variable) list and \( A \) stand for the corresponding finite set of assumptions of the form \( x \, \text{abt} \), one for each element of \( X \).

Using these notational shortcuts, we can write
\[
\{x_1, \ldots, x_n\}x_1 \, \text{abt}_1, \ldots, x_n \, \text{abt}_n \vdash a \, \text{abt}
\]
as
\[
X, A \vdash a \, \text{abt}
\]
which, when \( X \) is clear from context, can be further abbreviated
\[
A \vdash a \, \text{abt}
\]
In such cases we then write \( x \not\in A \) to mean \( x \not\in X \), where \( X \) is the set of parameters governed by \( A \).

Structural Induction with Binding and Scope

The principle of structural induction for abstract syntax trees extends to abstract binding trees as follows:

To show that \( X, A \vdash a \, \text{abt} \) has a property \( P \) it suffices to show:

- \( X, A, x \, \text{abt} \vdash x \, \text{abt} \) has property \( P \).
- For any \( o \) with \( O \vdash o(o_1, \ldots, o_k) \), if \( X, A \vdash a_1 \, \text{abt} \) has property \( P \) and \( \ldots \) and \( X, A \vdash a_k \, \text{abt} \) has property \( P \), then \( X, A \vdash o(a_1, \ldots, a_k) \, \text{abt} \) has property \( P \).
- If \( X, x', A, x' \, \text{abt} \vdash [x' \mapsto z]a \, \text{abt} \) has property \( P \) for some/any \( x' \not\in X \), then \( X, A \vdash x \, \text{abt} \) has property \( P \).

Example: ABT Size

Here is the inductive definition of \( S \vdash \text{sz}(a \, \text{abt}) = s \):
\[
S, \text{sz}(x \, \text{abt}) = 1 \vdash \text{sz}(x \, \text{abt}) = 1
\]
\[
S \vdash \text{sz}(a_1 \, \text{abt}) = a_1 \quad S \vdash \text{sz}(a_0 \, \text{abt}) = a_0 \quad S = a_1 + \ldots + a_n + 1
\]
\[
S \vdash \text{sz}(o(a_1, \ldots, a_n) \, \text{abt}) = s
\]
\[
S, \text{sz}(x' \, \text{abt}) = 1 \vdash \text{sz}([x' \mapsto z]a \, \text{abt}) = s
\]
\[
\vdash S, \text{sz}(x \, \text{abt}+1) = s + 1
\]
The size of an ABT counts each variable as 1 and adds 1 for each operator and abstractor in the ABT.

Inductive Definition of Abstract Binding Trees

The judgement \( X, A \vdash a \, \text{abt} \) is inductively defined by the following rules:

\[
X, x \vdash a \, \text{abt} \quad x \not\in X
\]
\[
S \vdash \text{sz}(a \, \text{abt}) = s
\]
or more generally the parametric hypothetical judgement:
\[
S \vdash \text{sz}(a \, \text{abt}) = s
\]
which, by taking the parameter list to be implicit and letting \( S \) stand for the corresponding finite set of assumptions of the form \( \text{sz}(a \, \text{abt}) = s \), one for each element of the parameter list, we can further abbreviate as:
\[
S \vdash \text{sz}(a \, \text{abt}) = s
\]

Example: ABT Size

Consider the inductive definition of the size, \( s \) of an abstract binding tree, \( a \), of valence \( n \) by a judgement of the form
\[
\text{sz}(a \, \text{abt}) = s
\]
and 
\[
S \vdash \text{sz}(a \, \text{abt}) = s
\]
which gives:
\[
S, \text{sz}(x \, \text{abt}) = 1 \vdash \text{sz}(x \, \text{abt}) = 1
\]
\[
S \vdash \text{sz}(a_1 \, \text{abt}) = a_1 \quad S \vdash \text{sz}(a_0 \, \text{abt}) = a_0 \quad S = a_1 + \ldots + a_n + 1
\]
\[
S \vdash \text{sz}(o(a_1, \ldots, a_n) \, \text{abt}) = s
\]
\[
S, \text{sz}(x' \, \text{abt}) = 1 \vdash \text{sz}([x' \mapsto z]a \, \text{abt}) = s
\]
\[
S \vdash \text{sz}(x \, \text{abt}+1) = s + 1
\]

Theorem 2 (ABT Size)

Every well-formed ABT \( A \) has a unique size. If \( A \vdash a \, \text{abt} \), then there exists a unique \( s \) such that
\[
\text{sz}(A \, \text{abt}) = s
\]

Proof: The proof proceeds by structural induction on the derivation of the premise. Thus there are three cases, one for each of the rules in the inductive definition of ABT size.

25

26

27

28

29
A apartness

The relation of a name \( x \) being apart from an abstract binding tree \( A \) says that \( x \) is a free name in \( A \). The apartness judgement \( A \vdash x \# A \ abt_\# \) where \( A \vdash a \ abt \) is inductively defined by the following rules:

\[
\begin{align*}
A \vdash x \# y \ abt_\# \to & \quad A \vdash x \# y \ y \ abt_\# \\
A \vdash x \# a_1 \ abt_\# & \quad \cdots & \quad A \vdash x \# a_n \ abt_\# \\
A \vdash x \# \big( a_1, \ldots, a_n \big) \ abt_\# \to & \quad A \vdash x \# \big( a_1, \ldots, a_n \big) \ abt_\# \\
A, y \ abt \vdash x \# A \ abt_\# \to & \quad A \vdash x \# y.A \ abt_\# \\
A \vdash x \# y.a \ abt_\# & \quad A \vdash x \# y.A \ abt_\# \\
\end{align*}
\]

Capture-Avoiding Substitution

Substitution is replacing all occurrences (if any) of a free name in an abstract binding tree by another ABT without violating the scopes of any names. The judgement \( A \vdash [u/z]h = v \ abt \) is inductively defined by the following rules:

\[
\begin{align*}
A \vdash [u/z]y = a \ abt \to & \quad A \vdash [u/z]y = x \ abt \\
x \ # y & \quad A \vdash [u/z]y = y \ abt \\
A \vdash [u/z]b_1 = c_1 \ abt & \quad \cdots & \quad A \vdash [u/z]b_n = c_n \ abt \to \\
A \vdash [u/z](b_1, \ldots, b_n) = (c_1, \ldots, c_n) \ abt & \quad A \vdash [u/z]y' \ abt = y' \ abt & \quad y' \ # x \\quad A \vdash [u/z]y, y' \ abt \\
A \vdash [u/z]b = y, y' \ abt & \quad A \vdash [u/z]b = y, y' \ abt \\
\end{align*}
\]

Renaming Bound Variables

We'll make use of identification up to renaming, or \( \alpha \) conversion.

- The name of a bound variable does not matter.
- Choose a different name to avoid ambiguity.

Scope Resolution

Where is a variable bound?

- **Lexical scope rule:** a variable is bound by the nearest enclosing binding.
  - Proceed upwards through the abstract syntax tree.
  - Find nearest enclosing 1st that binds that variable.

Examples:

- "Parallel" scopes:
  
  (let x be 3 in x + x) • (let x be 4 in x + x + x)

- "Nested" scopes:
  
  let x be 10 in (let x be 11 in x + x) + x

In the last rule we tacitly assume \( x \# A \). We may abbreviate \( A \vdash a =_\alpha b \ abt \) by \( A \vdash a =_\alpha b \) or just \( a =_\alpha b \) when \( a \) and \( A \) is clear from context.
Names of Bound Variables

But watch out:

• let \( x \) be 10 in \( (\text{let } x \text{ be 11 in } x+x) + x \) is the same as
  let \( y \) be 10 in \( (\text{let } x \text{ be 11 in } x+x) + y \).

• but is different from let \( y \) be 10 in \( (\text{let } x \text{ be 11 in } y+y) + y \).

When renaming we must avoid clashes with other variables in the same scope.