

Symbols

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

Symbols are sometimes called **names** or **atoms** or **identifiers**.

For $x \text{ sym}$ and $y \text{ sym}$, the judgement $x \# y$ says that x and y are distinct symbols.

1

Characters and Alphabets

An alphabet Σ is a countable set of symbols or **characters**.

- ASCII or UniCode.
- Binary digits.

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

2

Strings

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

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

3

Inductive Definition of Strings

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

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

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

$$c_1 \text{ char}, \dots, c_n \text{ char} \vdash s \text{ str}$$

which we abbreviate as $\Sigma \vdash s \text{ str}$

Explicit mention of Σ is often suppressed when it is clear from context.

4

String Induction

The principle of rule induction specializes to **string induction**.

Specifically, we can prove that every string s has a property \mathcal{P} by showing that

- ε has property \mathcal{P} ;
- if s has property \mathcal{P} and $c \text{ char}$, then $c \cdot s$ has property \mathcal{P} .

Essentially induction on string length without making length explicit.

5

Example: String Concatenation

Consider the inductive definition of string concatenation:

$$\frac{}{\varepsilon \hat{\cdot} s = s \text{ str}} \quad \frac{s_1 \hat{\cdot} s_2 = s \text{ str}}{(c \cdot s_1) \hat{\cdot} s_2 = c \cdot s \text{ str}}$$

By string induction on the first argument, can prove that the judgement form $s_1 \hat{\cdot} s_2 = s \text{ str}$ has mode $(\forall, \forall, \exists!)$.

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

6

String Notation

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

- Juxtaposition of characters: $abcd$ for $a \cdot (b \cdot (c \cdot (d \cdot \varepsilon)))$.
- Juxtaposition for concatenation: $abcd$ for $ab \hat{\cdot} cd$.
- Any possible concatenation: $ab \hat{\cdot} cd$ or $a \hat{\cdot} bed$ or $abcd \hat{\cdot} \varepsilon$ or ...

7

Abstract Syntax Trees

Let \mathcal{O} be a countable set of **operators**.

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

- Arity = 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

8

Abstract Syntax Trees

The set $\mathcal{T} = \mathcal{T}(\mathcal{O}, \Omega)$ is the set of **abstract syntax trees (ast's)** built from operators in \mathcal{O} :

$$\mathcal{T} = \{ o(t_1, \dots, t_k) \mid o \in \mathcal{O}, \text{ar}(o) = k, t_1, \dots, t_k \in \mathcal{T} \}.$$

The ast's t_1, \dots, t_k are **sub-trees** of $o(t_1, \dots, t_k)$.

9

AST's (Terms)

Examples:

- **zero** has arity 0, **succ(-)** has arity 1.
- **empty** has arity 0, **node(-, -)** has arity 2.

So, for example, **node(empty, empty)** is an ast.

10

Inductive Definition of Abstract Syntax Trees

The judgement $o(a_1, \dots, a_k) \text{ ast}$ is inductively defined by the following rules:

$$\frac{\Omega \vdash \text{ar}(o) = k \quad \frac{a_1 \text{ ast} \quad \dots \quad a_k \text{ ast}}{o(a_1, \dots, a_k) \text{ ast}}}{o(a_1, \dots, a_k) \text{ ast}}$$

The base case is for operators of arity **zero**, in which case the rule has no premises.

11

Structural Induction

The principle of rule induction specializes to **structural induction** when applied to abstract syntax trees over signature Ω

Specifically, we can prove that a ast has a property \mathcal{P} by showing for each operator o in \mathcal{O} that:

- Given that $\Omega \vdash \text{ar}(o) = k$
- if a_1 ast has property \mathcal{P} and \dots and a_k ast has property \mathcal{P} , then $o(a_1, \dots, a_k)$ ast has property \mathcal{P} .

When k is zero, reduces to showing $o()$ has property \mathcal{P}

12

Example: AST Height

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

$$\frac{\text{hgt}(a_1) = n_1 \quad \dots \quad \text{hgt}(a_k) = n_k \quad \max(n_1, \dots, n_k) = n \quad \text{ar}(o) = k}{\text{hgt}(o(a_1, \dots, a_k)) = \text{succ}(n)}$$

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

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

13

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 Ω can be described using a hypothetical judgement such as

$$x_1 \text{ ast}, \dots, x_k \text{ ast} \vdash_{\Omega} a \text{ ast}$$

where the x_1, \dots, x_k are pairwise distinct symbols

14

Variables and Substitution

Judgements can be extended to account for variables by use of hypotheses.

For example:

$$\text{hgt}(x_1) = n_1, \dots, \text{hgt}(x_k) = n_k \vdash \text{hgt}(a) = n$$

Properties of such judgements can be proved by using structural induction over ASTs. (See, for example, Lemma 5.1 on page 42 in Harper.)

15

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 Ω) of the following form:

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

16

Inductive Definition of Substitution

Lemma 1 (Substitution)

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

$$[a/x]x_1 = x_1, \dots, [a/x]x_k = x_k, [a/x]x = a \vdash [a/x]b = c$$

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 $[a_1, \dots, a_k/x_1, \dots, x_k]b$, is defined similarly.

17

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 α -**equivalence** and **capture-avoiding substitution**.

18

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.

19

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.

20

Abstract Binding Trees

Abstract binding trees are built from a name-indexed family of operators, called **abstractors**, which have the form $x.a$ where x name and a ast

In $x.a$, the name x is **bound** in a , which is the **scope** of x . The bound name x is only meaningful inside a and is **distinct** from any other names currently in scope.

21

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 Ω consists of a finite set of judgements of the form $\text{ar}(o) = (n_1, \dots, n_k)$, where n_i nat

For example, the arity $(0, \dots, 0)$ 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.

22

Abstract Binding Trees

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

$$\{x_1, \dots, x_k\} | x_1 \text{ abt}^0, \dots, x_k \text{ abt}^0 \vdash a \text{ abt}^n$$

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

We sometimes use $a \text{ abt}$ as shorthand for $a \text{ abt}^0$.

23

Abstract Binding Trees

Abstract binding trees inherently include **variables**. We let \mathcal{X} stand for the parameter (variable) list and \mathcal{A} stand for the corresponding finite set of assumptions of the form $x \text{ abt}^0$, one for each element of \mathcal{X} .

Using these notational shortcuts, we can write

$$\{x_1, \dots, x_k\} | x_1 \text{ abt}^0, \dots, x_k \text{ abt}^0 \vdash a \text{ abt}^n$$

as

$$\mathcal{X} | \mathcal{A} \vdash a \text{ abt}^n$$

which, when \mathcal{X} is clear from context, can be further abbreviated

$$\mathcal{A} \vdash a \text{ abt}^n$$

In such cases we then write $x \# \mathcal{A}$ to mean $x \notin \mathcal{X}$, where \mathcal{X} is the set of parameters governed by \mathcal{A} .

24

Inductive Definition of Abstract Binding Trees

The judgement $\mathcal{X} | \mathcal{A} \vdash a \text{ abt}^n$ is inductively defined by the following rules:

$$\frac{\mathcal{X}, x | \mathcal{A}, x \text{ abt}^0 \vdash x \text{ abt}^0}{\mathcal{X} | \mathcal{A} \vdash a_1 \text{ abt}^{n_1} \quad \dots \quad \mathcal{X} | \mathcal{A} \vdash a_k \text{ abt}^{n_k}} \frac{\text{ar}(o) = (n_1, \dots, n_k)}{\mathcal{X} | \mathcal{A} \vdash o(a_1, \dots, a_k) \text{ abt}^0}$$

$$\frac{\mathcal{X}, x' | \mathcal{A}, x' \text{ abt}^0 \vdash [x' \leftrightarrow x] a \text{ abt}^n \quad (x' \notin \mathcal{X})}{\mathcal{X} | \mathcal{A} \vdash x.a \text{ abt}^{n+1}}$$

25

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 $\mathcal{X} | \mathcal{A} \vdash a \text{ abt}^n$ has a property P it suffices to show:

- $\mathcal{X} | \mathcal{A}, x \text{ abt} \vdash x \text{ abt}^0$ has property P .
- For any o with $\Omega \vdash \text{ar}(o) = (n_1, \dots, n_k)$, if $\mathcal{X} | \mathcal{A} \vdash a_1 \text{ abt}^{n_1}$ has property P and \dots and $\mathcal{X} | \mathcal{A} \vdash a_k \text{ abt}^{n_k}$ has property P , then $\mathcal{X} | \mathcal{A} \vdash o(a_1, \dots, a_k) \text{ abt}^0$ has property P .
- If $\mathcal{X}, x' | \mathcal{A}, x' \text{ abt}^0 \vdash [x' \leftrightarrow x] a \text{ abt}^n$ has property P for some/any $x' \notin \mathcal{X}$, then $\mathcal{X} | \mathcal{A} \vdash x.a \text{ abt}^{n+1}$ has property P .

26

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

$$\text{sz}(a \text{ abt}^n) = s$$

or more generally the parametric hypothetical judgement:

$$\text{sz}(x_1 \text{ abt}^0) = 1, \dots, \text{sz}(x_k \text{ abt}^0) = 1 \vdash \text{sz}(a \text{ abt}^n) = s$$

which, by taking the parameter list to be implicit and letting \mathcal{S} stand for the corresponding finite set of assumptions of the form $\text{sz}(x \text{ abt}^0) = 1$, one for each element of the parameter list, we can further abbreviate as:

$$\mathcal{S} \vdash \text{sz}(a \text{ abt}^n) = s$$

27

Example: ABT Size

Here is the inductive definition of $\mathcal{S} \vdash \text{sz}(a \text{ abt}^n) = s$:

$$\frac{\mathcal{S}, \text{sz}(x \text{ abt}^0) = 1 \vdash \text{sz}(x \text{ abt}^0) = 1}{\mathcal{S} \vdash \text{sz}(a_1 \text{ abt}^{n_1}) = s_1 \quad \dots \quad \mathcal{S} \vdash \text{sz}(a_m \text{ abt}^{n_m}) = s_m \quad s = s_1 + \dots + s_m + 1}$$

$$\frac{\mathcal{S}, \text{sz}(o(a_1, \dots, a_m) \text{ abt}^0) = s}{\mathcal{S} \vdash \text{sz}(o(a_1, \dots, a_m) \text{ abt}^0) = s}$$

$$\frac{\mathcal{S}, \text{sz}(x' \text{ abt}^0) = 1 \vdash \text{sz}([x' \leftrightarrow x] a \text{ abt}^n) = s}{\mathcal{S} \vdash \text{sz}(x.a \text{ abt}^{n+1}) = s + 1}$$

The size of an ABT counts each variable as 1 and adds 1 for each operator and abstractor in the ABT.

28

Example: ABT Size

Theorem 2 (ABT Size)

Every well-formed ABT has a unique size. If $x_1 \text{ abt}^0, \dots, x_k \text{ abt}^0 \vdash a \text{ abt}^n$, then there exists a unique s nat such that

$$\text{sz}(x_1 \text{ abt}^0) = 1, \dots, \text{sz}(x_k \text{ abt}^0) = 1 \vdash \text{sz}(a \text{ abt}^n) = s$$

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. ■

29

Apartness

The relation of a name x **lying apart from** an abstract binding tree a says that x is a **free name** in a . The apartness judgement $\mathcal{A} \vdash x \# a \text{ abt}^n$ where $\mathcal{A} \vdash x \text{ abt}$ is inductively defined by the following rules:

$$\frac{x \# y}{\mathcal{A} \vdash x \# y \text{ abt}^0}$$

$$\frac{\mathcal{A} \vdash x \# a_1 \text{ abt}^{n_1} \quad \dots \quad x \# \mathcal{A} \vdash a_k \text{ abt}^{n_k}}{\mathcal{A} \vdash x \# o(a_1, \dots, a_k) \text{ abt}^0}$$

$$\frac{\mathcal{A}, y \text{ abt} \vdash x \# a \text{ abt}^n}{\mathcal{A} \vdash x \# y.a \text{ abt}^{n+1}}$$

30

Renaming of Bound Names

Two abstract binding trees are α -**equivalent** if they differ at most in the choice of bound variable names. The α -**equivalence** judgement is inductively defined by the following rules:

$$\frac{\mathcal{A}, x \text{ abt}^0 \vdash x =_\alpha x \text{ abt}^0}{\mathcal{A} \vdash a_1 =_\alpha b_1 \text{ abt}^{n_1} \quad \dots \quad \mathcal{A} \vdash a_k =_\alpha b_k \text{ abt}^{n_k}}{\mathcal{A} \vdash o(a_1, \dots, a_k) =_\alpha o(b_1, \dots, b_k) \text{ abt}^0}$$

$$\frac{\mathcal{A}, z \text{ abt} \vdash [z \leftrightarrow x]a =_\alpha [z \leftrightarrow y]b \text{ abt}^n}{\mathcal{A} \vdash x.a =_\alpha y.b \text{ abt}^{n+1}}$$

In the last rule we tacitly assume $z \# \mathcal{A}$. We may abbreviate $\mathcal{A} \vdash a =_\alpha b \text{ abt}^n$ by $\mathcal{A} \vdash a =_\alpha b$ or just $a =_\alpha b$ when n (and \mathcal{A}) is clear from context.

31

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 $\mathcal{A} \vdash [a/x]b = c \text{ abt}^n$ is inductively defined by the following rules:

$$\frac{}{\mathcal{A} \vdash [a/x]x = a \text{ abt}^0}$$

$$\frac{x \# y}{\mathcal{A} \vdash [a/x]y = y \text{ abt}^0}$$

$$\frac{\mathcal{A} \vdash [a/x]b_1 = c_1 \text{ abt}^{n_1} \quad \dots \quad \mathcal{A} \vdash [a/x]b_k = c_k \text{ abt}^{n_k}}{\mathcal{A} \vdash [a/x]o(b_1, \dots, b_k) = o(c_1, \dots, c_k) \text{ abt}^0}$$

$$\frac{\mathcal{A}, y' \text{ abt} \vdash [a/x]([y' \leftrightarrow y]b) = b' \text{ abt}^n \quad y' \# \mathcal{A} \quad y' \neq x}{\mathcal{A} \vdash [a/x]y.b = y'.b' \text{ abt}^n}$$

32

Re-Use of Bound Names

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

33

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 **let** that binds that variable.

34

Renaming Bound Variables

We'll make use of **identification up to renaming**, or α -**conversion**.

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

35

Names of Bound Variables

But watch out:

- let x be 10 in (let x be 11 in $x+x$)+ x is the **same** as
let y be 10 in (let x be 11 in $x+x$)+ y .
- but is **different** from let y be 10 in (let x be 11 in $y+y$)+ y .

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