Syntactic Objects

Surface Syntax

if true then false else true

Surface Syntax

sort

if true then false else true : Exp

Surface Syntax

sort

if true then false else true : Exp

true : Exp

false : Exp

Surface Syntax

$$\mathsf{if}\ -\ \mathsf{then}\ -\ \mathsf{else}\ - : (\mathsf{Exp} \times \mathsf{Exp} \times \mathsf{Exp}) \to \mathsf{Exp}$$

$$\mathsf{true} : \mathsf{Exp}$$

$$\mathsf{false} : \mathsf{Exp}$$

$$\mathsf{if\ true\ then\ false\ else\ true} : \mathsf{Exp}$$

Abstract Syntax

$$\mathsf{if}(\,-\,,\;-\,,-\,) : (\mathsf{Exp} \times \mathsf{Exp} \times \mathsf{Exp}) \to \mathsf{Exp}$$

$$\mathsf{true} : \mathsf{Exp}$$

$$\mathsf{false} : \mathsf{Exp}$$

$$\mathsf{if}(\mathsf{true}, \mathsf{false}, \mathsf{true}) : \mathsf{Exp}$$

Abstract Syntax Trees

$$\mathsf{plus} : \mathsf{Exp} \times \mathsf{Exp} \to \mathsf{Exp}$$

$$\mathsf{num} : \mathbb{N} \to \mathsf{Exp}$$

$$\mathsf{num}[42] : \mathsf{Exp}$$

$$\mathsf{num}[4] : \mathsf{Exp}$$

$$\mathsf{plus}(\mathsf{num}[4], \mathsf{num}[42]) : \mathsf{Exp}$$

Abstract Syntax Trees: Variables

$$\text{times} : \text{Exp} \times \text{Exp} \to \text{Exp}$$

$$\text{num} : \mathbb{N} \to \text{Exp}$$

variable $\longrightarrow$ $x : \text{Exp}$

$$\text{num}[4] : \text{Exp}$$

$$\text{times}(x, \text{num}[4]) : \text{Exp}$$

Abstract Syntax Trees: Variable Substitution

$$\mathsf{plus} : \mathsf{Exp} \times \mathsf{Exp} \to \mathsf{Exp}$$

$$\mathsf{times} : \mathsf{Exp} \times \mathsf{Exp} \to \mathsf{Exp}$$

$$\mathsf{num} : \mathbb{N} \to \mathsf{Exp}$$

$$\mathsf{num}[42] : \mathsf{Exp}$$

$$\mathsf{num}[4] : \mathsf{Exp}$$

$$\mathsf{times}(\mathsf{plus}(\mathsf{num}[4], \mathsf{num}[42]), \mathsf{num}[4]) : \mathsf{Exp}$$

Abstract Syntax Trees: Variable Substitution

$$\mathsf{plus} : \mathsf{Exp} \times \mathsf{Exp} \to \mathsf{Exp}$$

$$\mathsf{times} : \mathsf{Exp} \times \mathsf{Exp} \to \mathsf{Exp}$$

$$\mathsf{num} : \mathbb{N} \to \mathsf{Exp}$$

$$\mathsf{num}[42] : \mathsf{Exp}$$

$$\mathsf{num}[4] : \mathsf{Exp}$$

$$\mathsf{times}(\mathsf{plus}(\mathsf{num}[4], \mathsf{num}[42]), \mathsf{num}[4]) : \mathsf{Exp}$$

Abstract Syntax Trees Defined

Let S be a finite set of **sorts**, $\{O_s\}_{s \in S}$ be a sort-indexed family of operators o of sort s with arity ar$(o) = (s_1, \ldots, s_n)$, and $\{\mathcal{X}_s\}_{s \in S}$ be a sort-indexed family of variables x of sort s. The family $A[\mathcal{X}] = A[\mathcal{X}_s]_{s \in S}$ of abstract syntax trees (ASTs) of sort s is defined as follows:

- if $x \in \chi_s$, then $x \in A[\mathcal{X}]_s$
- if $o \in O_s$ with ar$(o) = (s_1, \ldots, s_n)$ and $a_1 \in A[\mathcal{X}]_{s_1}, \ldots, a_n \in A[\mathcal{X}]_{s_n}$, then $o(a_1, \ldots, a_n) \in A[\mathcal{X}]_s$

Abstract Syntax Trees Defined

Let S be a finite set of **sorts**, $\{O_s\}_{s \in S}$ be a sort-indexed family of operators o of sort s with arity $\mathsf{ar}(o) = (s_1, \ldots, s_n)$, and $\{\mathcal{X}_s\}_{s \in S}$ be a sort-indexed family of variables x of sort s. The family $A[\mathcal{X}] = A[\mathcal{X}_s]_{s \in S}$ of abstract syntax trees (ASTs) of sort s is defined as follows:

- if $x \in \mathcal{X}_s$, then $x \in A[\mathcal{X}]_s$
- if $o \in O_s$ with $\mathsf{ar}(o) = (s_1, \ldots, s_n)$ and $a_1 \in A[\mathcal{X}]_{s_1}, \ldots, a_n \in A[\mathcal{X}]_{s_n}$, then $o(a_1, \ldots, a_n) \in A[\mathcal{X}]_s$

Exercise: Come up with three example ASTs.

Abstract Syntax Trees: Structural Induction

To show that a property P for every AST it suffices to show P(a) holds for every $a \in A[\mathscr{X}]$, which holds when:

1. (Base Case) if $x \in \mathscr{X}_s$, then P$_s(x)$, and
2. (Step Case) if $o \in O_s$ with ar$(o) = (s_1, \ldots, s_n)$, then if P$_{s_1}(a_1), \ldots,$ P$_{s_n}(a_n)$ all hold, then P$_s(o(a_1, \ldots, a_n))$ holds.

Abstract Syntax Trees: Structural Induction

Lemma: If $\mathscr{X} \subseteq \mathscr{Y}$, then $A[\mathscr{X}] \subseteq A[\mathscr{Y}]$.

Proof. By structural induction.

1. (Base Case) If $x \in \mathscr{X}_s$ which implies that $x \in A[\mathscr{X}]_s$, then by assumption $x \in \mathscr{Y}$, and hence, by definition $x \in A[\mathscr{Y}]_s$.

2. (Step Case) Suppose $o \in \mathscr{O}_s$, $\mathsf{ar}(o) = (s_1, \ldots, s_n)$, $\mathscr{X} \subseteq \mathscr{Y}$. Then by induction: $a_1 \in A[\mathscr{X}]_{s_1} \iff a_1 \in A[\mathscr{Y}], \ldots, a_n \in A[\mathscr{X}] \iff a_n \in A[\mathscr{Y}]_{s_n}$. Then by definition $o(a_1, \ldots, a_n) \in A[\mathscr{X}]_s \iff o(a_1, \ldots, a_n) \in A[\mathscr{Y}]_s$.

Abstract Syntax Trees: Substitution

Variables are given their meaning through

$$
\begin{aligned}
&[\mathsf{num}[42]/x](\mathsf{plus}(x, \mathsf{mult}(\mathsf{num}[3], x)) \\
&\quad = \mathsf{plus}([\mathsf{num}[42]/x]x, [\mathsf{num}[42]/x]\mathsf{mult}(\mathsf{num}[3], x)) \\
&\quad = \mathsf{plus}([\mathsf{num}[42]/x]x, \mathsf{mult}([\mathsf{num}[42]/x]\mathsf{num}[3], [\mathsf{num}[42]/x]x)) \\
&\quad = \mathsf{plus}(\mathsf{num}[42], \mathsf{mult}(\mathsf{num}[3], \mathsf{num}[42]))
\end{aligned}
$$

Abstract Syntax Trees: Substitution

Substitution $[b_1/x]b_2 \in A[\mathscr{X}]_{s_2}$ on any AST $A[\mathscr{X}]$ where x is a variable of sort s_1, $b_1 \in A[\mathscr{X}]_{s_1}$, $b_2 \in A[\mathscr{X}, x]_{s_2}$ is defined as follows:

1. $[b_1/x]x = b_1$
2. $[b_1/x]y = y$, when $x \neq y$
3. $[b_1/x]o(a_1, \ldots, a_n) = o([b_1/x]a_1, \ldots, [b_1/x]a_n)$

Abstract Syntax Trees: Extension

Let $\mathscr{X}$ be a sort-indexed family of variables. Then: $(\mathscr{X}, x)$ where x is a variable of sort s such that $x \notin \mathscr{X}_s$, to stand for the sort-indexed family $\mathscr{Y}$ such that $\mathscr{Y}_s = \mathscr{X}_s \cup \{x\}$ and $\mathscr{Y}_{s'} = \mathscr{X}_{s'}$ for all $s' \neq s$.

This is also known a **adjoining.**

Bound Variables: Examples

Swift:

```
func exampleFunc(arg1: T1,...,argi: Ti) -> T {
  ...
}
```

Bound Variables: Examples

C#:

```
T exampleFunc(T1 arg1, ..., Ti argi) {
  ...
}
```

Bound Variables: Examples

Javascript:

```
function exampleFunc(arg1, ..., argi) {
    ...
}
```

Bound Variables: Examples

Lets:

```
let x = 2 + 2 in x * 6
```

Bound Variables: Examples

Lets:

```
let x = 2 + 2 in x * 6
```

```
let x be 2 + 2 in x * 6
```

Abstract Binding Trees: Example

Lets:

```
let x = 2 + 2 in x * 6
```

```
let x be 2 + 2 in x * 6
```

```
let(2 + 2, x.x * 6) : Exp

  ar(let) = (Exp, (Exp)Exp)
```

Abstract Binding Trees: Example

Lets:

```
let x = 2 + 2 in x * 6
```

```
let x be 2 + 2 in x * 6
```

```
let(2 + 2, x.x * 6) : Exp

  ar(let) = (Exp, (Exp)Exp)
```

Valence

Abstract Binding Trees: Valence

A **valence** of the form $(s_1, \ldots, s_i)s$ specifies an argument of sort s which binds i-variables of sorts $s_1, \ldots, s_i$ within it.

- $\overrightarrow{s}$ denotes a sequence of sorts $s_1, \ldots, s_i$
- $\overrightarrow{x}$ denotes a sequence of variables $x_1, \ldots, x_i$
- $\overrightarrow{x}$ has sort $\overrightarrow{s}$ denotes that each x_i of $\overrightarrow{x}$ having sort s_i of $\overrightarrow{s}$.

Abstract Binding Trees: Example

Lets:

```
let(2 + 2, x.x * 6) : Exp

  ar(let) = (Exp, (Exp)Exp)
```

Valence

Abstract Binding Trees Defined(?)

Let S be a finite set of **sorts**, $\{O_s\}_{s \in S}$ be a sort-indexed family of operators, and $\{\chi_s\}_{s \in S}$ be a sort-indexed family of variables x of sort s. The family $B[\chi] = B[\chi_s]_{s \in S}$ of abstract syntax trees (BSTs) of sort s is defined as follows:

- if $x \in \chi_s$, then $x \in B[\mathscr{X}]_s$
- if $o \in O_s$ with $\mathsf{ar}(o) = ((\overrightarrow{s_1})s_1, \ldots, (\overrightarrow{s_n})s_n)$ and if, for each $1 \leq i \leq n$, $\overrightarrow{x_i}$ is of sort $\overrightarrow{s_i}$ and $a_i \in B[\chi, \overrightarrow{x_i}]_{s_i}$, then $o(\overrightarrow{x_1} \,.\, a_1, \ldots, \overrightarrow{x_n} \,.\, a_n) \in B[\chi]_s$

Abstract Binding Trees: Binders

```
let(42,(x,y).y+1)
```

Binders specify the names of bound variables.

Binders in BSTs are denoted by "x." "

Abstract Bindings Trees: Binders

Binder

```
let(42,(x,y).y+1)
```

Binders specify the names of bound variables.

Binders in BSTs are denoted by "x ."

Abstract Binding Trees: Free vs Bound

```
let(42,x.x+1)
```

```
let(42,(x,y).y+1)
```

```
let(42,x.y+1)
```

```
let(let(42,x.x*2),(x,y).let(x+1,x.y+x+1))
```

Abstract Binding Trees: Free vs Bound

x is bound

```
let(42,x.x+1)
```

x is bound
y is bound

```
let(42,(x,y).y+1)
```

x is bound
y is free

```
let(42,x.y+1)
```

x is bound

x is bound

x is bound
y is bound

```
let(let(42,x.x*2),(x,y).let(x+1,x.y+x+1))
```

Abstract Binding Trees Defined(?)

What's wrong with this definition?

Let S be a finite set of **sorts**, $\{O_s\}_{s \in S}$ be a sort-indexed family of operators, and $\{\chi_s\}_{s \in S}$ be a sort-indexed family of variables. The family $B[\chi] = B[\chi_s]_{s \in S}$ of abstract syntax trees (BSTs) of sort s is defined as follows:

- if $x \in \chi_s$, then $x \in B[\mathcal{X}]_s$
- if $o \in O_s$ with $\mathsf{ar}(o) = ((\overrightarrow{s_1})s_1, \ldots, (\overrightarrow{s_n})s_n)$ and if, for each $1 \leq i \leq n$, $\overrightarrow{x_i}$ is of sort $\overrightarrow{s_i}$ and $a_i \in B[\mathcal{X}, \overrightarrow{x_i}]_{s_i}$, then $o(\overrightarrow{x_1} \,.\, a_1, \ldots, \overrightarrow{x_n} \,.\, a_n) \in B[\mathcal{X}]_s$

Abstract Binding Trees Defined(?)

```
let(plus(2,3),x.let(5,x.times(x,2))
```

Let S be a finite set of **sorts**, $\{O_s\}_{s \in S}$ be a sort-indexed family of operators, and $\{\chi_s\}_{s \in S}$ be a sort-indexed family of variables. The family $B[\chi] = B[\chi_s]_{s \in S}$ of abstract syntax trees (BSTs) of sort s is defined as follows:

- if $x \in \chi_s$, then $x \in B[\mathscr{X}]_s$
- if $o \in O_s$ with ar$(o) = ((\overrightarrow{s_1})s_1, \ldots, (\overrightarrow{s_n})s_n)$ and if, for each $1 \leq i \leq n$, $\overrightarrow{x_i}$ is of sort $\overrightarrow{s_i}$ and $a_i \in B[\mathscr{X}, \overrightarrow{x_i}]_{s_i}$, then $o(\overrightarrow{x_1} . a_1, \ldots, \overrightarrow{x_n} . a_n) \in B[\mathscr{X}]_s$

Abstract Binding Trees Defined(!)

Let S be a finite set of **sorts**, $\{O_s\}_{s \in S}$ be a sort-indexed family of operators, and $\{\mathcal{X}_s\}_{s \in S}$ be a sort-indexed family of variables. The family $B[\mathcal{X}] = B[\mathcal{X}_s]_{s \in S}$ of abstract syntax trees (BSTs) of sort s is defined as follows:

- if $x \in \chi_s$, then $x \in B[\mathcal{X}]_s$
- if $o \in O_s$ with ar$(o) = ((\overrightarrow{s_1})s_1, \ldots, (\overrightarrow{s_n})s_n)$ and if, for each $1 \leq i \leq n$ and for each renaming $\pi_i : \overrightarrow{x} \leftrightarrow \overrightarrow{x}'$, where $\overrightarrow{x}' \notin \mathcal{X}$, and $\overrightarrow{x_i}$ is of sort $\overrightarrow{s_i}$ and $\pi_i \cdot a_i \in B[\mathcal{X}, \overrightarrow{x_i'}]_{s_i}$, then $o(\overrightarrow{x_1} . a_1, \ldots, \overrightarrow{x_n} . a_n) \in B[\mathcal{X}]_s$

Abstract Binding Trees Defined(!)

```
let(plus(2,3),x.let(5,x.times(x,2))

let(plus(2,3),y.let(5,z.times(z,2))
```

Let S be a finite set of **sorts**, $\{O_s\}_{s \in S}$ be a sort-indexed family of operators, and $\{\mathscr{X}_s\}_{s \in S}$ be a sort-indexed family of variables. The family $B[\mathscr{X}] = B[\mathscr{X}_s]_{s \in S}$ of abstract syntax trees (BSTs) of sort s is defined as follows:

- if $x \in \chi_s$, then $x \in B[\mathscr{X}]_s$
- if $o \in O_s$ with $\mathsf{ar}(o) = ((\overrightarrow{s_1})s_1, \ldots, (\overrightarrow{s_n})s_n)$ and if, for each $1 \leq i \leq n$ and for each renaming $\pi_i : \overrightarrow{x} \leftrightarrow \overrightarrow{x}'$, where $\overrightarrow{x}' \notin \mathscr{X}$, and $\overrightarrow{x_i}$ is of sort $\overrightarrow{s_i}$ and $\pi_i \cdot a_i \in B[\mathscr{X}, \overrightarrow{x_i'}]_{s_i}$, then $o(\overrightarrow{x_1} \,.\, a_1, \ldots, \overrightarrow{x_n} \,.\, a_n) \in B[\mathscr{X}]_s$

Abstract Bindings Trees: Structural Induction

To show that a property P for every BST it suffices to show P$[\mathscr{X}](a)$ holds for every $a \in B[\mathscr{X}]$, which holds when:

1. (Base Case) if $x \in \chi_s$, then P$[\mathscr{X}]_s(x)$, and
2. (Step Case) if $o \in O_s$ with ar$(o) = ((\overrightarrow{s_1})s_1, \ldots, ((\overrightarrow{s_n})s_n)$, and foreach $1 \leq i \leq n$, we have P$[\mathscr{X}, \overrightarrow{x_i'}]_{s_i}(\pi_i \cdot a_i)$ hold for every renaming $\pi_i : \overrightarrow{x_i} \leftrightarrow \overrightarrow{x_i'}$, then P$[\mathscr{X}]_s(o(\overrightarrow{x_1} . a_1, \ldots, \overrightarrow{x_n} . a_n))$ holds.

Abstract Binding Trees: Substitution

Substitution $[b_1/x]b_2 \in B[\mathscr{X}]_{s_2}$ on any AST $B[\mathscr{X}]$ where x is a variable of sort s_1, $b_1 \in B[\mathscr{X}]_{s_1}$, $b_2 \in B[\mathscr{X}, x]_{s_2}$ is defined as follows:

1. $[b_1/x]x = b_1$
2. $[b_1/x]y = y$, when $x \neq y$
3. $[b_1/x]o(\overrightarrow{x_1} . a_1, \ldots, \overrightarrow{x_n} . a_n) = o(\overrightarrow{x_1} . a_1', \ldots, \overrightarrow{x_n} . a_n')$, where for each $1 \leq i \leq n$, we require that $\overrightarrow{x_i} \notin \mathsf{FV}(b_1)$, and we set $a_i' = [b_1/x]a_i$ if $x \notin \overrightarrow{x_i}$ and $a_i' = a_i$ otherwise.

Abstract Binding Trees: Substitution

Substitution $[b_1/x]b_2 \in B[\mathscr{X}]_{s_2}$ on any AST $B[\mathscr{X}]$ where x is a variable of sort s_1, $b_1 \in B[\mathscr{X}]_{s_1}$, $b_2 \in B[\mathscr{X}, x]_{s_2}$is defined as follows:

1. $[b_1/x]x = b_1$
2. $[b_1/x]y = y$, when $x \neq y$
3. $[b_1/x]o(\overrightarrow{x_1} \,.\, a_1, \ldots, \overrightarrow{x_n} \,.\, a_n) = o(\overrightarrow{x_1} \,.\, a_1', \ldots, \overrightarrow{x_n} \,.\, a_n')$, where for each $1 \leq i \leq n$, we require that $\overrightarrow{x_i} \notin$ FV(b), and we set $a_i' = [b_1/x]a_i$ if $x \notin \overrightarrow{x_i}$ and $a_i' = a_i$ otherwise.

The capture avoidance condition.

Abstract Binding Trees: α-Equivalence

$a =_\alpha b$:

1. $x =_\alpha x$
2. $o(\overrightarrow{x_1} . a_1, \ldots, \overrightarrow{x_n} . a_n) =_\alpha o(\overrightarrow{x_1} . a_1', \ldots, \overrightarrow{x_n} . a_n')$, if for every $1 \leq i \leq n$, $\pi_i \cdot a_i =_\alpha \pi_i' \cdot a_i'$ for all fresh renaming $\pi_i : \overrightarrow{x_i} \leftrightarrow \overrightarrow{z_i}$ and $\pi_i' : \overrightarrow{x_i'} \leftrightarrow \overrightarrow{z_i}$.

All BSTs are identified up to α-equivalence.