Syntax

Dr. Liam O'Connor
University of Edinburgh LFCS
UNSW, Term 3 2020

Concrete Syntax

Arithmetic Expressions

$$\frac{a\ \text{Atom} \qquad b\ \text{PExp}}{a \times b\ \text{PExp}} \qquad \frac{a\ \text{PExp} \qquad b\ \text{SExp}}{}$$

All the syntax we have seen so far is *concrete syntax*. Concrete syntax is described by judgements on strings, which describe the actual text input by the programmer.

Abstract Syntax

Working with concrete syntax directly is *unsuitable* for both compiler implementation and proofs. Consider:

- $3 + (4 \times$
- $3 + 4 \times 5$
- $(3 + (4 \times$

[1] "There is more than one way to do it".

Abstract Syntax

Working with concrete syntax directly is *unsuitable* for both compiler implementation and proofs. Consider:

- $3 + (4 \times$
- $3 + 4 \times 5$
- $(3 + (4 \times$

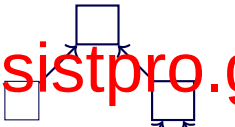
TIMTOWTDI[1] makes life harder for us. Different derivati
semantic program. We would like a representation of progra
possible, removing any extraneous information. Such a repr

abstract syntax.

[1] "There is more than one way to do it".

Abstract Syntax

Typically, the *abstract syntax* of a program is represented as a tree rather than as a string.



Writing trees in our inference rules would rapidly become unwieldy, however. We shall define a term language in which to express trees.

Terms

Definition

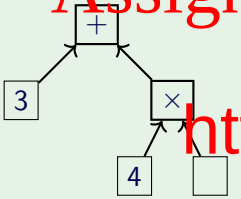
In this course, a *term* is a structure that can either be a symbol, like `Plus` or `Times` or 3; or a compound, w
subterms, all in par

$$t ::= \texttt{Symbol} \mid (\texttt{Symbol}\ t_1\ t_2\ \ldots)$$

These particular terms are also known as *s-expr* y be thought of a subset of Haskell where the only kinds of expressions allowed are literals and data constructors.

Term Examples

Example



Armed with an appropriate Haskell data declara straightforwardly:

$$\begin{array}{lcl} \texttt{data}\ \mathit{Exp} & = & \texttt{Plus}\ \mathit{Exp}\ \mathit{Exp} \\ & \mid & \texttt{Times}\ \mathit{Exp}\ \mathit{Exp} \\ & \mid & \texttt{Num}\ \mathit{Int} \end{array}$$

Concrete to Abstract

Concrete Syntax

$$\frac{i \in \mathbb{Z}}{i\ \text{SExp}} \qquad \frac{a\ \text{SExp}}{e\ \text{Atom}} \qquad \frac{e\ \text{Atom}}{e\ \text{PExp}} \qquad \frac{e\ \text{PExp}}{e\ \text{SExp}}$$

Abstract Syntax

$$\frac{i \in \mathbb{Z}}{(\texttt{Num}\ i)\ \text{AST}} \qquad \frac{a\ \text{AST} \qquad b\ \text{AS}}{(\texttt{Plus}\ a\ b)\ \text{AST}} \qquad \frac{}{(\texttt{Times}\ a\ b)\ \text{AST}}$$

Now we have to specify a *relation* to connect the two!

Relations

Up until now, most judgements we have used have been *unary* — corresponding to a set of satisfying objects.

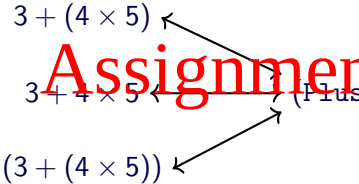
It's also possible for a judgement to express a relationship between two objects (a *binary* judgem

Example (Rela

- 4 *divides* 16 (binary)
- `mail` *is an anagram of* `liam` (binary)
- 3 *plus* 5 *equals* 8 (ternary)

n-ary judgements where $n \geq 2$ are sometimes called *relations*, and correspond to an n-tuple of satisfying objects.

Parsing Relation



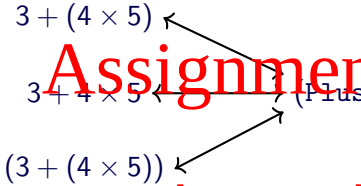
$$\frac{}{i \text{ Atom}}$$

$$\frac{a \text{ Atom} \qquad b}{a \times b \text{ PExp}}$$

$$\frac{a \text{ PExp} \qquad b \text{ SExp}}{a + b \text{ SExp}}$$

$$\frac{e \text{ SExp}}{(e) \text{ Atom}} \qquad \frac{e \text{ Atom}}{e \text{ PExp}} \qquad \frac{e \text{ PExp}}{e \text{ SExp}}$$

Parsing Relation



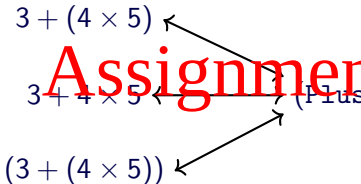
$$\frac{}{i \text{ Atom} \longleftrightarrow (\texttt{Num}\ i)}$$

$$\frac{a \text{ Atom} \qquad b}{a \times b \text{ PExp}}$$

$$\frac{a \text{ PExp} \qquad b \text{ SExp}}{a + b \text{ SExp}}$$

$$\frac{e \text{ SExp}}{(e) \text{ Atom}} \qquad \frac{e \text{ Atom}}{e \text{ PExp}} \qquad \frac{e \text{ PExp}}{e \text{ SExp}}$$

Parsing Relation



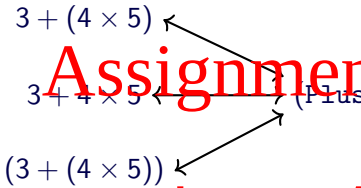
$$\frac{}{i \text{ Atom} \longleftrightarrow (\texttt{Num}\ i)\ \text{AST}}$$

$$\frac{a \text{ Atom} \longleftrightarrow a' \text{ AST} \qquad b \text{ PExp} \longleftrightarrow b' \text{ AST}}{a \times b \text{ PExp} \longleftrightarrow (\texttt{Times}\ a'\ b') \text{ AST}}$$

$$\frac{a \text{ PExp} \qquad b \text{ SExp}}{a + b \text{ SExp}}$$

$$\frac{e \text{ SExp}}{(e) \text{ Atom}} \qquad \frac{e \text{ Atom}}{e \text{ PExp}} \qquad \frac{e \text{ PExp}}{e \text{ SExp}}$$

Parsing Relation



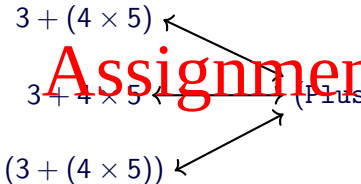
$$\frac{}{i \text{ Atom} \longleftrightarrow (\texttt{Num}\ i)}$$

$$\frac{a \text{ Atom} \longleftrightarrow a' \text{ AST} \qquad b}{a \times b \text{ PExp} \longleftrightarrow (\texttt{Time}}$$

$$\frac{a \text{ PExp} \qquad b \text{ SExp}}{a + b \text{ SExp}}$$

$$\frac{e \text{ SExp}}{(e) \text{ Atom}} \qquad \frac{e \text{ Atom}}{e \text{ PExp}} \qquad \frac{e \text{ PExp}}{e \text{ SExp}}$$

Parsing Relation



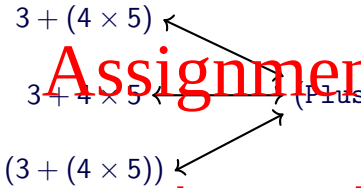
$$\frac{}{i \text{ Atom} \longleftrightarrow (\texttt{Num}\ i)}$$

$$\frac{a \text{ Atom} \longleftrightarrow a' \text{ AST} \qquad b}{a \times b \text{ PExp} \longleftrightarrow (\texttt{Time}}$$

$$\frac{a \text{ PExp} \longleftrightarrow a' \text{ AST} \qquad b \text{ SExp} \longleftrightarrow b' \text{ AST}}{a + b \text{ SExp} \longleftrightarrow (\texttt{Plus}\ a'\ b') \text{ AST}}$$

$$\frac{e \text{ SExp}}{(e) \text{ Atom}} \qquad \frac{e \text{ Atom}}{e \text{ PExp}} \qquad \frac{e \text{ PExp}}{e \text{ SExp}}$$

Parsing Relation



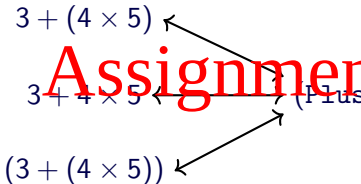
$$\frac{}{i \text{ Atom} \longleftrightarrow (\texttt{Num}\ i)}$$

$$\frac{a \text{ Atom} \longleftrightarrow a' \text{ AST} \qquad b}{a \times b \text{ PExp} \longleftrightarrow (\texttt{Time}}$$

$$\frac{a \text{ PExp} \longleftrightarrow a' \text{ AST} \qquad b \text{ SExp} \longleftrightarrow b' \text{ AST}}{a + b \text{ SExp} \longleftrightarrow (\texttt{Plus}\ a'\ b') \text{ AST}}$$

$$\frac{e \text{ SExp} \longleftrightarrow a' \text{ AST}}{(e) \text{ Atom} \longleftrightarrow a' \text{ AST}} \qquad \frac{e \text{ Atom}}{e \text{ PExp}} \qquad \frac{e \text{ PExp}}{e \text{ SExp}}$$

Parsing Relation



$$\frac{}{i\ \text{Atom} \longleftrightarrow (\texttt{Num}\ \ldots)}$$

$$\frac{a\ \text{Atom} \longleftrightarrow a'\ \text{AST} \qquad b}{a \times b\ \text{PExp} \longleftrightarrow (\texttt{Time}}$$

$$\frac{a\ \text{PExp} \longleftrightarrow a'\ \text{AST} \qquad b\ \text{SExp} \longleftrightarrow b'\ \text{AST}}{a + b\ \text{SExp} \longleftrightarrow (\texttt{Plus}\ a'\ b')\ \text{AST}}$$

$$\frac{e\ \text{SExp} \longleftrightarrow a'\ \text{AST}}{(e)\ \text{Atom} \longleftrightarrow a'\ \text{AST}} \qquad \frac{e\ \text{Atom} \longleftrightarrow a\ \text{AST}}{e\ \text{PExp} \longleftrightarrow a\ \text{AST}} \qquad \frac{e\ \text{PExp} \longleftrightarrow a\ \text{AST}}{e\ \text{SExp} \longleftrightarrow a\ \text{AST}}$$

Relations as Algorithms

The *parsing relation* $\longleftrightarrow$ is an extension of our existing concrete syntax rules.
Therefore it is unambiguous, just as those rules are.
Furthermore, the abstract syntax for a particular concrete syntax can be unambiguously $\longleftrightarrow$.

Relations as Algorithms

The *parsing relation* $\longleftrightarrow$ is an extension of our existing concrete syntax rules.
Therefore it is unambiguous, just as those rules are.
Furthermore, the abstract syntax for a particular concrete syntax can be unambiguously $\longleftrightarrow$.

An Algorithm

To determine the a

1. Derive the left hand side of the $\longleftrightarrow$ (the co l reaching axioms.
2. Fill in the right hand side of the $\longleftrightarrow$ (the ab ting at the axioms.

This process of converting concrete to abstract syntax is called *parsing*.

Example

Rules

$$\frac{}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{\dfrac{}{1\ \mathsf{P}} \qquad \dfrac{}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i}\qquad \frac{a\ \mathsf{S}}{a\ \mathsf{A}}\qquad \frac{e\ \mathsf{A}}{e\ \mathsf{P}}\qquad \frac{\dfrac{}{1\ \mathsf{A}}\qquad \dfrac{}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i}\ \mathsf{S} \qquad \frac{a\ \mathsf{S}}{a\ \mathsf{A}} \qquad \frac{e\ \mathsf{A}}{e\ \mathsf{P}}$$

$$\dfrac{\dfrac{\dfrac{}{1\ \mathsf{A}}}{1\ \mathsf{P}} \qquad \dfrac{\dfrac{}{2 \times 3\ \mathsf{P}}}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\dfrac{\dfrac{\dfrac{}{1\ \mathsf{A}}}{1\ \mathsf{P}} \qquad \dfrac{\dfrac{\dfrac{}{2\ \mathsf{A}}}{2 \times 3\ \mathsf{P}}}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{\quad \rightarrow a'} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{\quad} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{\quad} \qquad \frac{\quad}{b')}$$

$$\dfrac{\dfrac{\dfrac{}{1\ \mathsf{A}}}{1\ \mathsf{P}} \qquad \dfrac{\dfrac{}{2\ \mathsf{A}} \qquad \dfrac{}{2 \times 3\ \mathsf{P}}}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{a \to a'} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{(a, b')} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{(a, b')}$$

$$\dfrac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}}}{1\ \mathsf{P}}}{1 + 2 \times 3\ \mathsf{S}} \qquad \dfrac{\dfrac{\dfrac{}{2\ \mathsf{A}}}{2 \times 3\ \mathsf{P}}}{2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{\quad} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{\quad} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{\quad}$$

$$\rightarrow a \qquad \rightarrow a' \qquad b')$$

$$\frac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}}}{1\ \mathsf{P} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}} \qquad \dfrac{\dfrac{}{2\ \mathsf{A}}}{2 \times 3\ \mathsf{P}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{} \qquad \frac{}{\rightarrow a}$$

$$\frac{}{b')} \qquad \frac{}{}$$

$$\frac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}} \quad \dfrac{\dfrac{}{2\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 2} \quad \dfrac{}{2 \times 3\ \mathsf{P}}}{1\ \mathsf{P} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}} \quad \dfrac{}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{\quad \to a'} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{\quad} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{\quad} \qquad \frac{\quad}{b')}$$

$$\frac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}} \quad \dfrac{\dfrac{}{2\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 2} \quad \dfrac{}{\quad}}{2 \times 3\ \mathsf{P}}}{1\ \mathsf{P} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}} \quad \dfrac{}{\quad}}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{} \qquad \frac{}{\rightarrow a} \qquad \frac{}{b')}$$

$$\dfrac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}} \quad \dfrac{\dfrac{}{2\ \mathsf{A} \longleftrightarrow (\mathtt{Num}\ 2} \quad \dfrac{}{2 \times 3\ \mathsf{P}}}{1\ \mathsf{P} \longleftrightarrow (\mathtt{Num}\ 1)\ \mathsf{AST}} \quad \dfrac{}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{a \to a'} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{e\ \mathsf{P} \longleftrightarrow a} \qquad \frac{}{(a\ b)}$$

$$\dfrac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\texttt{Num}\ 1)\ \texttt{AST}}}{1\ \mathsf{P} \longleftrightarrow (\texttt{Num}\ 1)\ \texttt{AST}} \qquad \dfrac{\dfrac{\dfrac{}{2\ \mathsf{A} \longleftrightarrow (\texttt{Num}\ 2}}{2 \times 3\ \mathsf{P} \longleftrightarrow (\texttt{Times}\ (\texttt{Num}\ 2)\ (\texttt{Num}\ 3))\ \texttt{AST}}}{2 \times 3\ \mathsf{S}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{a \to a'} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{(a, b')} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{(a, b')}$$

$$\frac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\texttt{Num}\ 1)\ \texttt{AST}}}{1\ \mathsf{P} \longleftrightarrow (\texttt{Num}\ 1)\ \texttt{AST}} \qquad \frac{\dfrac{\dfrac{}{2\ \mathsf{A} \longleftrightarrow (\texttt{Num}\ 2}}{2 \times 3\ \mathsf{P} \longleftrightarrow (\texttt{Times}\ (\texttt{Num}\ 2)\ (\texttt{Num}\ 3))\ \texttt{AST}}}{2 \times 3\ \mathsf{S} \longleftrightarrow (\texttt{Times}\ (\texttt{Num}\ 2)\ (\texttt{Num}\ 3))\ \texttt{AST}}}{1 + 2 \times 3\ \mathsf{S}}$$

Example

Rules

$$\frac{i \in \mathbb{Z}}{i} \qquad \frac{a\ \mathsf{S} \longleftrightarrow a'}{} \qquad \frac{e\ \mathsf{A} \longleftrightarrow a}{} \qquad \frac{e\ \mathsf{P} \longleftrightarrow a}{} \qquad \frac{}{\rightarrow a} \qquad \frac{}{b')}$$

$$\frac{\dfrac{\dfrac{}{1\ \mathsf{A} \longleftrightarrow (\texttt{Num}\ 1)\ \mathsf{AST}} \quad \dfrac{\dfrac{}{2\ \mathsf{A} \longleftrightarrow (\texttt{Num}\ 2)} \quad \dfrac{}{2 \times 3\ \mathsf{P} \longleftrightarrow (\texttt{Times}\ (\texttt{Num}\ 2)\ (\texttt{Num}\ 3))\ \mathsf{AST}}}{2 \times 3\ \mathsf{S} \longleftrightarrow (\texttt{Times}\ (\texttt{Num}\ 2)\ (\texttt{Num}\ 3))\ \mathsf{AST}}}{1 + 2 \times 3\ \mathsf{S} \longleftrightarrow (\texttt{Plus}\ (\texttt{Num}\ 1)\ (\texttt{Times}\ (\texttt{Num}\ 2)\ (\texttt{Num}\ 3)))\ \mathsf{AST}}$$

The Inverse

What about the inverse operation to parsing?

Unparsing

Unparsing, also called *pretty-printing*, is the process of starting with the abstract syntax on the right h ing to synthesise a concrete syntax o

The Inverse

What about the inverse operation to parsing?

Unparsing

Unparsing, also called *pretty-printing*, is the process of starting with the abstract syntax on the right h ing to synthesise a concrete syntax o

Problem

There are many concrete syntaxes for a given abstract syntax. T
non-deterministi

While it is desirable to have:

$$parse \circ unparse = id$$

It is not usually true that:

$$unparse \circ parse = id$$

Example

Going from right to left requires some formatting guesswork t

Algorithms to do this can get quite involved!

Let's implement a parser for arithmetic. to coding

Adding Let

Let us extend our arithmetic expression language with variables, including a `let` construct to give them values.

Concrete Syntax

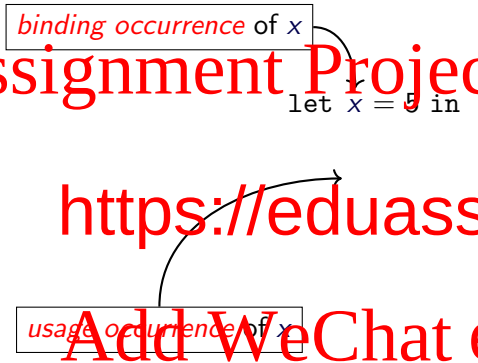
Example

```
let x = 3 in
  x + 4
end
```

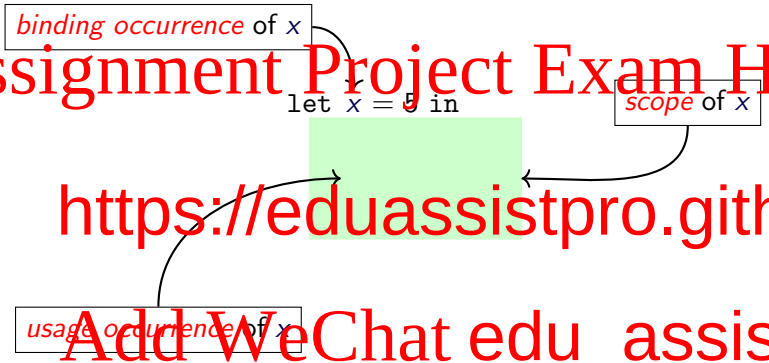
```
let x = 3 in
  let y = 4 in x + y end
end
```

Scope

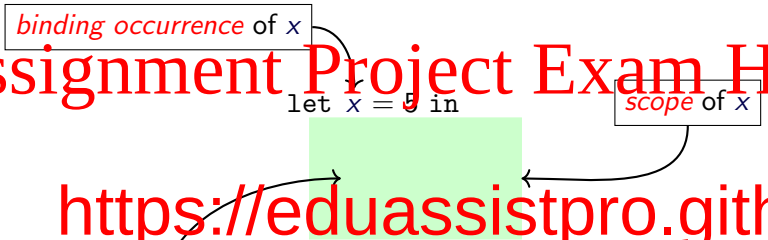
Scope



Scope



Scope



The
ion
use
resolution. Usually this is done statically. If no binding can be found, an *out of scope* error is raised.

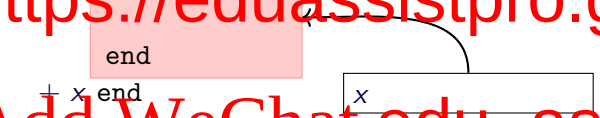
Shadowing

What does this program evaluate to?


```
        end
    + x end
```

Shadowing

What does this program evaluate to?



This program results i

α-equivalence

What is the difference between these two programs?


```
let x = 5 in          let a = 5 in
  let x = 2 in          let y = 2 in
```

α-equivalence

What is the difference between these two programs?


```
let x = 5 in          let a = 5 in
  let x = 2 in          let y = 2 in
```


They are semantically identical, but differ in the choice of boun
expressions are called α-equivalent.

We write $e_1 \equiv_\alpha e_2$ if e_1 is α-equivalent to e_2. The relation $\equiv_\alpha$ is an *equivalence relation*. That is, it is *reflexive*, *transitive* and *symmetric*.

The process of consistently renaming variables that preserves α-equivalence is called *α-renaming*.

Substitution

A variable x is *free* in an expression e if x occurs in e but is not bound in e.

Example (Free

The variable x

Substitution

A variable x is *free* in an expression e if x occurs in e but is not bound in e.

Example (Free

The variable x

A *substitution*, x ment
of all free occurrences of x in e with the term t.

Example (Simple substitution)

$(5 \times x + 7)[x := y \times 4]$ is the same as $(5 \times (y$

Problems with substitution

Consider these two α-equivalent expressions.

$\texttt{let } y = 5 \texttt{ in } y \times x + 7 \texttt{ end}$

and

What happens if you apply the substitution $[x :$

Problems with substitution

Consider these two α-equivalent expressions.

$$\texttt{let } y = 5 \texttt{ in } y \times x + 7 \texttt{ end}$$

and

$$\texttt{let } z = 5 \texttt{ in } z \times (y \times 3) + 7 \texttt{ end}$$

What happens if you apply the substitution $[x :$ You get two non-α-equivalent expressions!

$$\texttt{let } y = 5 \texttt{ in } y \times (y$$

This problem is called *capture*.

Variable Capture

Capture can occur for a substitution $e[x := t]$ whenever there is a bound variable in the expression e with the same name as a free variable occuring in t.

Fortunately

It is always possible to

- α-rename the offending bound variable to an unused na
- If you have access to the free variable's definition, renami
- Use a different abstract syntax representation that ma (More on this later).

Abstract Syntax for Variables

We shall extend our AST and parsing relation to include a definition for Let and variables.

Let Syntax

$$\frac{}{x \text{ Atom} \longleftrightarrow (\texttt{Var}\ x)\ \text{AST}}$$

$$\frac{x \text{ Ident} \quad e_1 \text{ SExp} \longleftrightarrow a_1 \text{ AST}}{\texttt{let}\ x = e_1\ \texttt{in}\ e_2\ \texttt{end Atom} \leftarrow}$$

First Order Abstract Syntax

Consider the following two pieces of abstract syntax:

```
(Let "x" (Num 5) (Plus (Num 4) (Var "x")))
```


This demonstrat

First Order Abstract Syntax

Consider the following two pieces of abstract syntax:

```
(Let "x" (Num 5) (Plus (Num 4) (Var "x")))
```


This demonstrat

1. Substitution capture is a problem.

First Order Abstract Syntax

Consider the following two pieces of abstract syntax:

```
(Let "x" (Num 5) (Plus (Num 4) (Var "x")))
```


This demonstrat

1. Substitution capture is a problem.
2. α-equivalent expressions are not equal. Determining i α-equivalent requires us to search for a consistent les

First Order Abstract Syntax

Consider the following two pieces of abstract syntax:


```
(Let "x" (Num 5) (Plus (Num 4) (Var "x")))
```


This demonstrat

1. Substitution capture is a problem.
2. α-equivalent expressions are not equal. Determining i α-equivalent requires us to search for a consistent les
3. No distinction is made between binding and usage occurrences of variables. This means that we must define substitution by hand on each type of expression we introduce.

First Order Abstract Syntax

Consider the following two pieces of abstract syntax:

```
(Let "x" (Num 5) (Plus (Num 4) (Var "x")))
```


This demonstrat

1. Substitution capture is a problem.
2. α-equivalent expressions are not equal. Determining i α-equivalent requires us to search for a consistent les
3. No distinction is made between binding and usage occurrences of variables. This means that we must define substitution by hand on each type of expression we introduce.
4. Scoping errors cannot be easily detected — malformed syntax is easy to write.

de Bruijn Indices

One popular approach to address the first issue is *de Bruijn indices*.

Key Idea

1. Remove all i
2. Replace the i is we must skip in o


```
(Let "a" (Num 5)
 (Let "y" (Num 2)
  (Plus (Var "a") (Var "y"))))
```

de Bruijn Indices

One popular approach to address the first issue is *de Bruijn indices*.

Key Idea

1. Remove all i
2. Replace the i is we must skip in o


```
(Let "a" (Num 5)
 (Let "y" (Num 2)
  (Plus (Var "a") (Var "y"))))
```

```
(Let (Num 5)
 (Let (Num
  (Plus (Var 1) (Var 0))))
```

de Bruijn Indices

One popular approach to address the first issue is *de Bruijn indices*.

Key Idea

1. Remove all i
2. Replace the i is we must skip in o


```
(Let "a" (Num 5)                (Let (Num 5)
 (Let "y" (Num 2)                (Let (Num
  (Plus (Var "a") (Var "y"))))    (Plus (Var 1) (Var 0))))
```

Debruijnification

Algorithm

Given a piece of an convert to de Bruijn indices, ack at each `Let` and p is encountered, replace the variable name with its first position in the stack (starting at the top of the stack).

This approach naturally handles shadowing. It's also possibl Bruijn indices going in the other direction (from the bottom of the stack, upwards).

de Bruijn Substitution

Substitution is now capture avoiding by definition.

$$
\begin{array}{lcl}
(\texttt{Num}\ i)[n := t] & = & (\texttt{Num}\ i) \\
(\texttt{Plus}\ a\ b)[n := t] & = & (\texttt{Plus}\ a[n := t]\ b[n := t]) \\
(\texttt{Times}\ a\ b)[n := t] & = & (\texttt{Times}\ a[n := t]\ b[n := t])
\end{array}
$$

de Bruijn Substitution

Substitution is now capture avoiding by definition.

$$
\begin{array}{lcl}
(\texttt{Num}\ i)[n := t] & = & (\texttt{Num}\ i) \\
(\texttt{Plus}\ a\ b)[n := t] & = & (\texttt{Plus}\ a[n := t]\ b[n := t]) \\
(\texttt{Times}\ a\ b)[n := t] & = & (\texttt{Times}\ a[n := t]\ b[n := t])
\end{array}
$$

de Bruijn Substitution

Substitution is now capture avoiding by definition.

$$
\begin{array}{lcl}
(\texttt{Num}\ i)[n := t] & = & (\texttt{Num}\ i) \\
(\texttt{Plus}\ a\ b)[n := t] & = & (\texttt{Plus}\ a[n := t]\ b[n := t]) \\
(\texttt{Times}\ a\ b)[n := t] & = & (\texttt{Times}\ a[n := t]\ b[n := t])
\end{array}
$$

$$(\texttt{Let}\ e_1\ e_2)[n := t] \quad = \quad (\texttt{Let}\ e_1[$$

de Bruijn Substitution

Substitution is now capture avoiding by definition.

$$
\begin{array}{lcl}
(\texttt{Num}\ i)[n := t] & = & (\texttt{Num}\ i) \\
(\texttt{Plus}\ a\ b)[n := t] & = & (\texttt{Plus}\ a[n := t]\ b[n := t]) \\
(\texttt{Times}\ a\ b)[n := t] & = & (\texttt{Times}\ a[n := t]\ b[n := t]) \\
\\
(\texttt{Let}\ e_1\ e_2)[n := t] & = & (\texttt{Let}\ e_1[
\end{array}
$$

Where $e_{\uparrow n}$ is an *up-shifting* operation defined as follo

$$
\begin{array}{lcl}
(\texttt{Num}\ i)_{\uparrow n} & = & (\texttt{Num}\ i) \\
(\texttt{Plus}\ a\ b)_{\uparrow n} & = & (\texttt{Plus}\ a_{\uparrow n}\ \ \uparrow n \\
(\texttt{Times}\ a\ b)_{\uparrow n} & = & (\texttt{Times}\ a_{\uparrow n}\ b_{\uparrow n}) \\
(\texttt{Var}\ m)_{\uparrow n} & = & \begin{cases} (\texttt{Var}\ (m+1)) & \text{if } m \geq n \\ (\texttt{Var}\ m) & \text{otherwise} \end{cases} \\
(\texttt{Let}\ e_1\ e_2)_{\uparrow n} & = & (\texttt{Let}\ e_{1\uparrow n}\ e_{2\uparrow n+1})
\end{array}
$$

Examining de Bruijn indices

How do de Bruijn indices stack up against our explicit names in terms of the problems we identified?

1. Substituti

Examining de Bruijn indices

How do de Bruijn indices stack up against our explicit names in terms of the problems we identified?

1. Substituti
2. α-equival

Examining de Bruijn indices

How do de Bruijn indices stack up against our explicit names in terms of the problems we identified?

1. Substituti
2. α-equival
3. We still must define substitution machinery by hand for e

Examining de Bruijn indices

How do de Bruijn indices stack up against our explicit names in terms of the problems we identified?

1. Substituti
2. α-equival
3. We still must define substitution machinery by hand for e
4. It is still possible to make malformed syntax – indices that o example.

Examining de Bruijn indices

How do de Bruijn indices stack up against our explicit names in terms of the problems we identified?

1. Substituti
2. α-equival
3. We still must define substitution machinery by hand for e
4. It is still possible to make malformed syntax – indices that o example.

Two out of four isn't bad, but can we do better by changing the term language?

Higher Order Terms

We shall change our term language to include built-in notions of variables and binding.

$$t \quad ::= \quad \texttt{Symbol} \qquad \textit{(symbols)}$$

As in Haskell, we shall say that application is left-associative, so


```
(Plus (Num 3) (Num 4)) = ((Pl
```

Now the binding and usage occurrences of variables are distinguished from regular symbols in our term language. Let's see what this lets us do...

Representing Let

$$\frac{a_1\ \mathsf{AST} \qquad a_2\ \mathsf{AST}}{(\mathtt{Let}\ a_1\ (x.\ a_2))\ \mathsf{AST}}$$

We no longer need a r
terms.

Representing Let

$$\frac{a_1\ \text{AST} \qquad a_2\ \text{AST}}{(\text{Let}\ a_1\ (x.\ a_2))\ \text{AST}}$$

We no longer need a r
terms.

How would we repr

$$\begin{array}{rcl} \text{data}\ AST & = & \text{Num} \\ & | & \text{Plus} \\ & | & \text{Time} \\ & | & \text{Let}\ A \end{array}$$

Representing Let

$$\frac{a_1 \text{ AST} \qquad a_2 \text{ AST}}{(\texttt{Let } a_1 \ (x. \ a_2)) \text{ AST}}$$

We no longer need a r
terms.

How would we repr

$$\begin{aligned} \text{data } AST &= \text{Num } Int \\ &\mid \text{Plus } AST \\ &\mid \text{Times } A \\ &\mid \text{Let } AST \end{aligned}$$

So `let` $x = 3$ `in` $x + 2$ `end` becomes, in Haskell:

$$(\texttt{Let } (\texttt{Num } 3) \ (\lambda x \rightarrow \texttt{Plus } x \ (\texttt{Num } 2))$$

Substitution

We can now define substitution across all terms in the meta-logic:

$$\texttt{Symbol1}[x := e] = \texttt{Symbol2}$$

$$y[x := e] = \begin{cases} e & \text{if } y = x \\ \text{undefin} & \text{othe} \end{cases}$$

$$(y.\ t)[x := e] = (y.\ t[x$$

Substitution

We can now define substitution across all terms in the meta-logic:

$$
\begin{array}{lcl}
\texttt{Symbol}[x := e] & = & \texttt{Symbol} \\
y[x := e] & = & \begin{cases} e & \text{if } y = x \\ \end{cases} \\
(y.\ t)[x := e] & = & (y.\ t[x \\
 & & \text{undefin} \quad \text{othe}
\end{array}
$$

Where $\mathrm{FV}(\cdot)$ is the set of all free variables in a term:

$$
\begin{array}{lcl}
\mathrm{FV}(\texttt{Symbol}) & = & \emptyset \\
\mathrm{FV}(x) & = & \{x\} \\
\mathrm{FV}(t_1\ t_2) & = & \mathrm{FV}(t_1) \cup \mathrm{FV}(t_2) \\
\mathrm{FV}(x.\ t) & = &
\end{array}
$$

Substitution

We can now define substitution across all terms in the meta-logic:

$$
\begin{array}{lcl}
\texttt{Symbol}[x := e] & = & \texttt{Symbol} \\
y[x := e] & = & \begin{cases} e & \text{if } y = x \\ \text{undefin} & \text{othe} \end{cases} \\
(y.\ t)[x := e] & = & (y.\ t[x
\end{array}
$$

Where $\mathrm{FV}(\cdot)$ is the set of all free variables in a term:

$$
\begin{array}{lcl}
\mathrm{FV}(\texttt{Symbol}) & = & \emptyset \\
\mathrm{FV}(x) & = & \{x\} \\
\mathrm{FV}(t_1\ t_2) & = & \mathrm{FV}(t_1) \cup \mathrm{FV}(t_2) \\
\mathrm{FV}(x.\ t) & = & \mathrm{FV}(t) \setminus \{x\}
\end{array}
$$

Cheating Outrageously

Substitution capture is still a problem in the substitution we just defined but it is not our problem. Because substitution is defined in the meta-language, it's the job of the implementors of t

- When Hask capture.
- When we are doing proofs in our meta-logic, there is no impl can just say that we assume α-equivalent ter me that variables are always renamed to avoid capture.

So, we have solved the problem by making it someone else's problem. **Outrageous cheating!**

Evaluating All Approaches

	FOAS		EOAS	
	Proofs	Haskell	Strings	de Bruijn
Ca				lved
α-				lved
Ge				blem
M				blem

Evaluating All Approaches

	FOAS		EOAS	
	Proofs	Haskell	Strings	de Bruijn
Ca				lved
α-				lved
Ge				blem
M				blem

- In embedded languages and in pen and paper proofs, HOAS

Evaluating All Approaches

	FOAS		EOAS	
	Proofs	Haskell	Strings	de Bruijn
Ca				lved
α-				lved
Ge				blem
M				blem

- In embedded languages and in pen and paper proofs, HOAS
- In conventional language implementations and mac s, de Bruijn indices are more popular.

Evaluating All Approaches

	HOAS		FOAS	
	Proofs	Haskell	Strings	de Bruijn
Capture	Cheat	Cheat	Problem	Solved
α-Eq				lved
M				blem

- In embedded languages and in pen and paper proofs, HO
- In conventional language implementations and mac s, de Bruijn indices are more popular.
- In your assignments, strings will be used