PARSING RELATIONS

A mapping : concrete syntax $\longmapsto$ abstract syntax specified in terms of an inductive definition of the terms in your language.

Gives a straightforward algo. for converting b/w concrete and abstract syntax:

1. Do the derivation of the term using inference rules
2. Use the parsing relation starting from the top of the derivation tree

Ex: Abstract syntax for `let x = 5 in x+2 end`:

$$\dfrac{\dfrac{}{x\ \text{Ident}} \quad \dfrac{\dfrac{5 \in \mathbb{Z}}{5\ \text{Atom} \hookrightarrow \text{Num}\ 5\ \text{AST}}}{5\ \text{PExp} \hookrightarrow \text{Num}\ 5\ \text{AST}}}{5\ \text{SExp} \hookrightarrow \text{Num}\ 5\ \text{AST}} \qquad \dfrac{\dfrac{\dfrac{}{x\ \text{Ident}}}{x\ \text{Atom} \hookrightarrow \text{Var}\ \text{"x"}\ \text{AST}}}{x\ \text{PExp} \hookrightarrow \text{Var}\ \text{"x"}\ \text{AST}} \quad \dfrac{\dfrac{\dfrac{2 \in \mathbb{Z}}{2\ \text{Atom} \hookrightarrow \text{Num}\ 2\ \text{AST}}}{2\ \text{PExp} \hookrightarrow \text{Num}\ 2\ \text{AST}}}{2\ \text{SExp} \hookrightarrow \text{Num}\ 2\ \text{AST}}}{x+2\ \text{SExp} \hookrightarrow \text{Plus}\ (\text{Var}\ \text{"x"})\ (\text{Num}\ 2)\ \text{AST}}$$

$$\texttt{let x = 5 in x+2 end}\ \text{Atom} \hookrightarrow \text{Let}\ \text{"x"}\ (\text{Num}\ 5)\ (\text{Plus}\ (\text{Var}\ \text{"x"})\ (\text{Num}\ 2))\ \text{AST}$$

SUBSTITUTION

A way of replacing occurrences of free variables w/ expressions.

$e[x := t]$

- e: ambient expression
- x: variable
- t: Expression to replace free occurrences of x with

[x is free in e $\Rightarrow$ x is not a bound variable in e]

Capture: If t has a free variable w/ same name as a bound variable in e, we get an expr. w/ potentially diff. semantics!

Q: Can we avoid This from happening? How?

Ans: Yes, α-rename any bound vars. in e which clash with t and Then do The substitution

Ex: Do These substitutions:

① (Let x (y. (Plus (Num 1) x))) [x := y] (capture!)
$\equiv_\alpha$ (Let x (z. (Plus (Num 1) x))) [x := y]
= (Let y (z. (Plus (Num 1) y)))

② (let y (z. (Plus (Num 1) z))) [x := y]
= itself

③ (Let x (z. (Plus x z))) [x := y]
= (Let y (z. (Plus y z)))

④ (Let x (x. (Plus (Num 1) x))) [x := y] (free; not free; not free)
= (Let y (x. (Plus (Num 1) x)))

SEMANTICS

Goal: Specify how terms in some language have meaning

This week: Operational semantics (meaning ⟹ value)

$\longmapsto$	$\Downarrow$
Small step semantics = step-by-step eval. = ~ship b/w states of an expression.	Big step semantics = Overall eval. = Direct ~ship b/w expressions and values

Both presented using natural deduction!

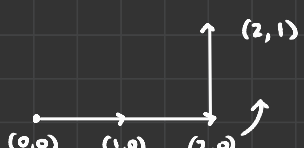
There is also denotational semantics: not as important for us, and axiomatic semantics (next week).

Ex: Operational Semantics for robot instructions

R ::= move; R | turn; R | stop

- move; R ↳ move 1 unit in the facing dir.
- turn; R ↳ turn 90° counter-clockwise
- stop ↳ terminal inst.

Example program: move; move; turn; move; stop



Start @ (0,0)
Facing east

(Exercise: try this w/ the big + small step semantics we write!)

(a) Small-step semantics for R.

States: $\Sigma = \{ (\underline{p}, \underline{d}, i) : \underline{p}, \underline{d} \in \mathbb{Z}^2,\ i \in R \}$

pos. $\underline{p}$, dir $\underline{d}$, instructions left to execute i

Initial states: $I = \{ ((\binom{0}{0}), (\binom{1}{0}), i) : i \in R \}$

Final states: $F = \{ (\underline{p}, \underline{d}, \text{stop}) : \underline{p}, \underline{d} \in \mathbb{Z}^2 \}$

Transitions: $R_{90} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{pmatrix}$, rotation by 90° ↺

$$\frac{}{(\underline{p}, \underline{d}, \text{move}; i) \mapsto (\underline{p}+\underline{d}, \underline{d}, i)} \qquad \frac{}{(\underline{p}, \underline{d}, \text{turn}; i) \longmapsto (\underline{p}, R_{90}\,\underline{d}, i)}$$

(b) Big-step semantics for R.

Evaluable expressions: $E = \Sigma$ from before

Values: $V = \{ (\underline{p}, \underline{d}) : \underline{p}, \underline{d} \in \mathbb{Z}^2 \}$

Evaluation rules:

$$\frac{}{(\underline{p}, \underline{d}, \text{stop}) \Downarrow (\underline{p}, \underline{d})} \qquad \frac{(\underline{p}+\underline{d}, \underline{d}, i) \Downarrow (\underline{p}', \underline{d}')}{(\underline{p}, \underline{d}, \text{move}; i) \Downarrow (\underline{p}', \underline{d}')}$$

$$\frac{(\underline{p}, R_{90}\,\underline{d}, i) \Downarrow (\underline{p}', \underline{d}')}{(\underline{p}, \underline{d}, \text{turn}; i) \Downarrow (\underline{p}', \underline{d}')}$$

λ-CALCULUS (untyped)

A very minimal model of computation. Inspiration and basis for functional programming!

Also a prog. lang in its own right:

$$t := \text{Symbol} \mid x \mid t_1\ t_2 \mid \lambda x.\ t$$

Symbol: ↳ numbers, ... ; x: ↳ variables ; $t_1\ t_2$: ↳ application (of functions) ; $\lambda x.\ t$: ↳ abstraction (functions)

Three rules of equivalence/reduction:

① α-equiv: $\lambda x.f\,x \equiv_\alpha \lambda y.f\,y$ etc.

② β-reduction: $(\lambda x.t)\,u \mapsto_\beta t[x := u]$

③ η-reduction: $(\lambda x.f\,x) \mapsto_\eta f$

Normal form: cannot be βη-reduced anymore

Major fact: β-reduction is a confluence (Church-Rosser Theorem)
Consequence: terms always eval. to the same thing irresp. of the method

Ex. Reduction of a term:

$$(\lambda n\,f\,x.\,f\,(n\,x\,x))(\lambda f\,x.\,f) \equiv_\alpha (\lambda n\,f\,x.\,f\,(n\,x\,x))(\lambda g\,y.\,g)$$

$$\begin{aligned}
&\mapsto_\beta \lambda f\,x.\,f\,((\lambda g\,y.\,g)\,x\,x)\\
&\mapsto_\beta \lambda f\,x.\,f\,((\lambda y.\,x)\,x)\\
&\mapsto_\beta \lambda f\,x.\,f\,x\\
&\equiv \lambda f.\,\lambda x.\,f\,x\\
&\mapsto_\eta \lambda f.\,f
\end{aligned}$$