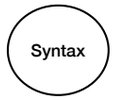
Core Design Concepts Discussed:



Abstract Syntax

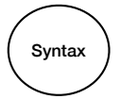
An internal representation of concrete syntax

Harley Eades III

So far we have learned about concrete syntax and their definition using context-free grammars and inductive definitions. Now we switch to another representation called abstract syntax...

Concrete Syntax

Core Design Concepts:



$$b \rightarrow x \mid \mathsf{T} \mid \mathsf{F} \mid \mathsf{if}\ b\ \mathsf{then}\ b\ \mathsf{else}\ b \mid \mathsf{fun}\ x\ \{b\} \mid b(b) \mid (b)$$

$$\frac{}{\mathsf{Concrete}(x)}\ \mathsf{Var} \qquad \frac{\mathsf{Concrete}(b_1) \quad \mathsf{Concrete}(b_2) \quad \mathsf{Concrete}(b_3)}{\mathsf{Concrete}(\mathsf{if}\ b_1\ \mathsf{then}\ b_2\ \mathsf{else}\ b_3)}\ \mathsf{If} \qquad \frac{\mathsf{Concrete}(b_1) \quad \mathsf{Concrete}(b_2)}{\mathsf{Concrete}(b_1(b_2))}\ \mathsf{App}$$

$$\frac{}{\mathsf{Concrete}(\mathsf{T})}\ \mathsf{T} \qquad \frac{\mathsf{Concrete}(b)}{\mathsf{Concrete}(\mathsf{fun}\ x\ \{b\})}\ \mathsf{Fun} \qquad \frac{\mathsf{Concrete}(b)}{\mathsf{Concrete}((b))}\ \mathsf{Paren}$$

$$\frac{}{\mathsf{Concrete}(\mathsf{F})}\ \mathsf{F}$$

$\mathsf{fun}\ x\ \{\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{F}\ \mathsf{else}\ \mathsf{T}\}$

Recall that we define concrete syntax, the user-interface to a programming language, using a grammar, but this grammar can equally be defined using an inductive definition. Here I define the grammar for our boolean language using a judgment called Concrete, and inference rules stating that which syntax is valid concrete syntax. Finally, we have an example of concrete syntax which is the compliment function we defined during class. Concrete syntax is an external facing interface, but it is not super convenient to use to write algorithms or write proofs. One thing that gets tedious is keep track of variable scope...

Concrete Syntax

Core Design Concepts:

$$\text{fun}\, x\, \{(\text{fun}\, y\, \{b_1\})(b_2)\}$$

$$b_1 \wedge b_2 \vee b_3$$

Consider this example. We, as humans, can see that x can be used in both b_1 and b_2, but y can only be use in b_1 and not in b_2. Another example is order of operations, in the concrete syntax this is hard to see. For example, the second program implicitly uses order of operations. Keeping track of all this in a program or proof is very tedious. For this reason, we translate concrete syntax into a new representation called abstract syntax. The goal of abstract syntax is...

Abstract Syntax

Core Design Concepts:

Make writing algorithms and proofs easier by abstracting away extraneous details.

make it easier to write algorithms and proofs by abstracting away details in the concrete syntax that are difficult to process.

Abstract Syntax

Core Design Concepts:

May be harder to read and write by humans.

Even though abstract syntax is easier to process, the representation is designed for machines, and may be both harder to read and harder to write by humans. Let's consider an example...

Core Design Concepts:

Abstract Syntax

$$b \rightarrow \mathsf{T} \mid \mathsf{F} \mid \mathsf{if}(b_1, b_2, b_3)$$

$$\frac{}{\mathsf{Abstract}(\mathsf{T})}\,\mathsf{T} \qquad \frac{\mathsf{Abstract}(b_1) \quad \mathsf{Abstract}(b_2) \quad \mathsf{Abstract}(b_3)}{\mathsf{Abstract}(\mathsf{if}(b_1, b_2, b_3))}\,\mathsf{If}$$

$$\frac{}{\mathsf{Abstract}(\mathsf{F})}\,\mathsf{F}$$

Here we have a grammar and judgment for abstract syntax of our boolean language. We will add functions soon. The major difference is that if becomes a prefix operator. This may seem very small, but it has one big advantage, we no longer need parens as part of our syntactic category. So we have gotten simpler. So how do we add functions?

Abstract Syntax

Core Design Concepts:

$$b \rightarrow \mathsf{T} \mid \mathsf{F} \mid \mathsf{if}(b_1, b_2, b_3) \mid \mathsf{fun}(x, b) \mid \mathsf{app}(b_1, b_2)$$

$$\frac{}{\mathsf{Abstract}(x)}\ \mathsf{Var} \qquad \frac{\mathsf{Abstract}(b_1) \quad \mathsf{Abstract}(b_2) \quad \mathsf{Abstract}(b_3)}{\mathsf{Abstract}(\mathsf{if}(b_1, b_2, b_3))}\ \mathsf{If} \qquad \frac{\mathsf{Abstract}(b_1) \quad \mathsf{Abstract}(b_2)}{\mathsf{Abstract}(\mathsf{app}(b_1, b_2))}\ \mathsf{App}$$

$$\frac{}{\mathsf{Abstract}(\mathsf{T})}\ \mathsf{T} \qquad \frac{\mathsf{Abstract}(b)}{\mathsf{Abstract}(\mathsf{fun}(x, b))}\ \mathsf{Fun}$$

$$\frac{}{\mathsf{Abstract}(\mathsf{F})}\ \mathsf{F}$$

We add functions by adding two new terms for functions and application which are both prefix operators. Functions have two parameters: the variable being bound by the function and the body of the function. However, the scoping issue we saw before is less complex, because of the style of abstract syntax....

Core Design Concepts:

Syntax

Abstract Syntax

$$b \rightarrow \mathsf{T} \mid \mathsf{F} \mid \mathsf{if}(b_1, b_2, b_3) \mid \mathsf{fun}(x, b) \mid \mathsf{app}(b_1, b_2)$$

$$\frac{}{\mathsf{Abstract}(x)}\ \mathsf{Var} \qquad \frac{\mathsf{Abstract}(b_1) \quad \mathsf{Abstract}(b_2) \quad \mathsf{Abstract}(b_3)}{\mathsf{Abstract}(\mathsf{if}(b_1, b_2, b_3))}\ \mathsf{If} \qquad \frac{\mathsf{Abstract}(b_1) \quad \mathsf{Abstract}(b_2)}{\mathsf{Abstract}(\mathsf{app}(b_1, b_2))}\ \mathsf{App}$$

$$\frac{}{\mathsf{Abstract}(\mathsf{T})}\ \mathsf{T} \qquad \frac{\mathsf{Abstract}(b)}{\mathsf{Abstract}(\mathsf{fun}(x, b))}\ \mathsf{Fun}$$

$$\frac{}{\mathsf{Abstract}(\mathsf{F})}\ \mathsf{F}$$

$$\mathsf{fun}\, x\, \{(\mathsf{fun}\, y\, \{b_1\})(b_2)\} \rightarrow \mathsf{fun}(x, \mathsf{app}(\mathsf{fun}(y, b_1), b_2)$$

We can see that the scope of x is exactly the entire application, but the score of y is exactly b_1 and nothing more. Prefix operators make it very easy to understand the structure of a program. How about order of operations?

Core Design Concepts:

Syntax

Abstract Syntax

$$b \rightarrow \mathsf{T} \mid \mathsf{F} \mid \mathsf{if}(b_1, b_2, b_3) \mid \mathsf{fun}(x, b) \mid \mathsf{app}(b_1, b_2)$$

$$\frac{}{\mathsf{Abstract}(x)}\ \mathsf{Var} \qquad \frac{\mathsf{Abstract}(b_1) \quad \mathsf{Abstract}(b_2) \quad \mathsf{Abstract}(b_3)}{\mathsf{Abstract}(\mathsf{if}(b_1, b_2, b_3))}\ \mathsf{If} \qquad \frac{\mathsf{Abstract}(b_1) \quad \mathsf{Abstract}(b_2)}{\mathsf{Abstract}(\mathsf{app}(b_1, b_2))}\ \mathsf{App}$$

$$\frac{}{\mathsf{Abstract}(\mathsf{T})}\ \mathsf{T} \qquad \frac{\mathsf{Abstract}(b)}{\mathsf{Abstract}(\mathsf{fun}(x, b))}\ \mathsf{Fun}$$

$$\frac{}{\mathsf{Abstract}(\mathsf{F})}\ \mathsf{F}$$

$$b_1 \wedge b_2 \vee b_3 \rightarrow \mathsf{Or}(\mathsf{And}(b_1, b_2), b_3)$$

The abstract syntax translations removes the ambiguity, because prefix operators are fully parenthesized. Every programming language translates their concrete syntax into an abstract syntax similar to what we have shown here. Now we can fully define what parsing, pretty printing or unparsing is....

Static Semantics

Core Design Concepts:

- Parsing: Translating concrete syntax into abstract syntax.
- Pretty printing (unparsing): Translating abstract syntax into concrete syntax.

parsing is the translation of concrete syntax into abstract syntax. One property that must be true is that for every concrete program there is exactly one abstract program. That is, the grammar for the concrete syntax cannot be ambiguous. However, for any abstract program there are LOTs of concrete programs. This is because, we can take a lot of liberties with formatting our concrete programs. Thus, this translation is indeed ambiguous.