Parse Trees

Beefed-up calculator language

\[
\begin{align*}
\text{run} & \rightarrow \text{stmt run} \mid \text{stmt} \\
\text{stmt} & \rightarrow \text{ares STOP} \\
\text{ares} & \rightarrow \text{VAR ASN bres} \mid \text{bres} \\
\text{bres} & \rightarrow \text{bres BOP res} \mid \text{res} \\
\text{res} & \rightarrow \text{res COMP exp} \mid \text{exp} \\
\text{exp} & \rightarrow \text{exp OPA term} \mid \text{term} \\
\text{term} & \rightarrow \text{term OPM factor} \mid \text{factor} \\
\text{factor} & \rightarrow \text{NUM} \mid \text{VAR} \mid \text{LP bres RP}
\end{align*}
\]

Download today’s tarball and run make to get a parse tree for some string in this language.

We notice that the parse tree is large and unwieldy with many unnecessary nodes.

Abstract Syntax Tree

Consider the program \( x := (5 + 3) \times 2; x - 7; \). What should the AST for this look like?

AST Properties

Remember, ASTs are not about the syntax! They remove syntactic details from the program, leaving only the semantics.

Typically, we show ordering (e.g. of \text{ares}'s in the previous example) by nesting: the last child of a statement is the next statement, or null.

Are ASTs language independent?
Static type checking

Consider the string \((7 > 2) + 3\);. This is an error. But where should this error be identified?

Each node in the AST has a type, possibly "void".

---

Static type checking with variables

What about the string \(x = 6 > 3; x * 12;\)?

We have to know the type of the variable \(x\). Otherwise, there is no way to detect this error at compile-time.

Only \textit{statically-typed languages} allow this sort of checking. Remember, in this class \textit{errors are a good thing}!

---

Unit outcomes

You should know:

- What an AST is, and why we need them.
- The relationship between language, parse tree, and AST.
- How static type-checking works, at a basic level.

You should be able to:

- Draw a parse tree for a given string, given the grammar.
- Determine the AST from the parse tree. Note that there is some flexibility here!