SLR(1)

SLR(1) parsers handle conflicts by using one token of look-ahead:
- If the next token is an outgoing edge label of that state, shift and move on.
- If the next token is in the follow set of a non-terminal that we can reduce to, then do that reduction.

Of course, there may still be conflicts, in which case the grammar is not SLR(1). More look-ahead may be needed.

LALR parsers are similar, but they use more specialized FOLLOW sets rather than the “global” follow sets that we have seen.

Parse Trees

Beefed-up calculator language

```
run → ares STOP run | ares STOP
ares → VAR ASN bres | bres
bres → bres BOP res | res
res → res COMP exp | exp
exp → exp OPA term | term
term → term OPM factor | factor
factor → NUM | VAR | LP bres RP
```

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 "ars" 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 \( x = 6 > 3; \ x \ast 12; \)?

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

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

Class 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!