Abstract syntax trees (ASTs)

The AST eliminates the scaffolding introduced to render the grammar unambiguous. Items such as temporary variables can be introduced into the AST to simplify subsequent activity (optimization, code generation).
Creating an AST

We can easily add actions to the grammar to create AST nodes and properly link these nodes to form the AST.

\[
\begin{align*}
S & \rightarrow \ E \ \$ \\
E & \rightarrow \ E \ + \ T \\
& \rightarrow \ T \\
T & \rightarrow \ T \ * \ F \\
& \rightarrow \ F \\
F & \rightarrow \ ( \ E ) \\
& \rightarrow \ \text{const} \\
& \rightarrow \ \text{id}
\end{align*}
\]

\{
$$ = \text{MakeBinTree}($$+)$$,1$$,3);$$
\}
\{
$$ = 1;$$
\}
\{
$$ = \text{MakeBinTree}($$*)$$,1$$,3);$$
\}
\{
$$ = 1;$$
\}
\{
$$ = \text{MakeConst}($$1);$$
\}
\{
$$ = \text{MakeSymb}($$1);$$
\}

Free of clutter, the resulting tree can then be traversed to instantiate symbol tables, perform type checking, optimize the program, and generate code.
typedef struct _TreeNode {
    struct {
        int linenumber;
        int colnumber;
    } sourceinfo;
    NodeInfo info;
    struct _TreeNode *child;
    struct _TreeNode *sibling;
    struct _TreeNode *head;
    struct _TreeNode *parent;
    struct _TreeNode *leftsib;
} TreeNode;

NodeInfo is a union of tree node information: symbol table pointers, integer values, operator types, etc.

MakeFamily(parent, sibs): adopts sibs into the parent’s family, returning the parent.

MakeSiblings(c1,c2): units siblings c1 and c2, returning the end of the resulting list (shown below).

MakeOperatorNode(opnum): creates an operator node, where opnum is the “name” of a “token”.

MakeIntegerNode(intval): creates an integer node with value intval.

MakeStringNode(str): creates a string node with value str.

MakeSymbolNode(sym): creates a symbol reference node to sym.
Using the AST routines

\[
\text{Num} \rightarrow \text{D } \$
\]

\[
\text{D} \rightarrow \text{D } \text{d}
\]

\[
\text{B} \rightarrow \text{x } \text{d}
\]

\[
\{\$ = \$1 \rightarrow \text{head;}
\}
\]

\[
\{\$ = \text{MakeSiblings($1, MakeIntegerNode($2))};\}
\]

\[
\mid \space \text{B}
\]

\[
\{\$ = \text{MakeIntegerNode($1);}\}
\]

\[
\mid \lambda
\]

\[
\{\$ = 10;\}
\]

The above list is created by the actions shown to the left. The first number in the list is the base, and the subsequent numbers are the digits as parsed from left to right.
Example AST

```c
int a1;
extern int a2;

int factorial(X)
int X;
{
    int Y;
    Y = X;
    if (Y > 0) { Y*factorial(X-1); }
    else [1];
}

void main() {
    int i;
    a1 = factorial(i=5);
    a2 = factorial(3);
}
```

The AST is shown to the right, with indentation reflecting tree depth.

Note the regular structure:
- functions and inline procedures are represented similarly.
- an if-then structure is represented as an if-then-else with trivial "else" code.