Ambiguity in Real Languages

In actual programming languages ambiguity occurs most often in expressions involving operators (where it can be fixed hierarchically) and in if-then-else constructs.

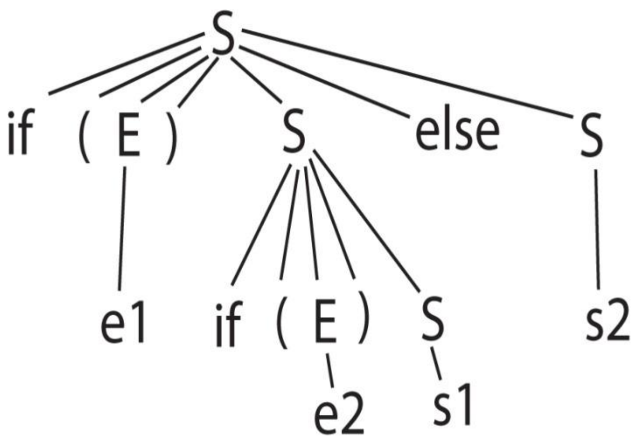
Most languages, including BPL, have a grammar rule similar to

```
      S ::= if (E) S else S | if (E) S | T
      (T refers to other forms of statements)
```

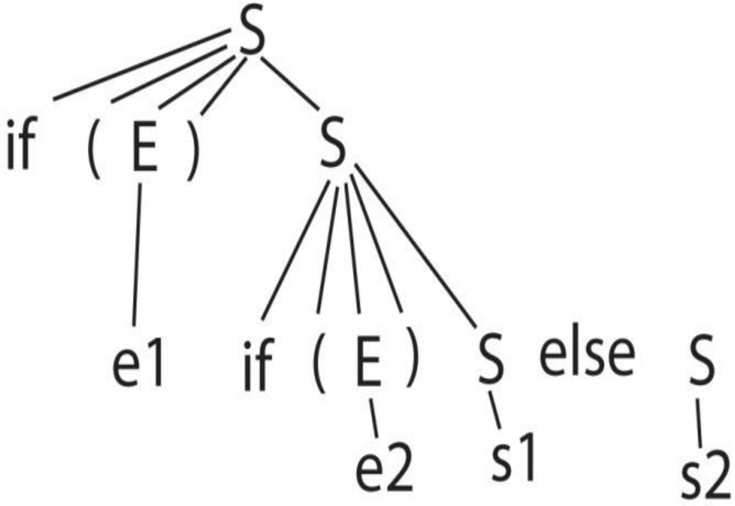
With this grammar the sentence

```
            if (e1) if (e2) s1 else s2
```

can be parsed two ways:



```
if (e1)
    if (e2)
        s1
else
    s2
```



```
if (e1)
    if (e2)
        s1
    else
        s2
```

There is no simple rewrite of the grammar (that I know of) that fixes this problem. Most languages leave the rule ambiguous, disambiguate it internally by designing their parsers to always choose the second of the parse trees:

```
if (e1)
    if (e2)
        s1
    else
        s2
```

and tell programmers that a dangling else always goes with the nearest "unelsed" if. Of course, most languages allow compound statements (statements grouped together with braces), and using these the programmer can indicate which grouping is desired.

Here is a simple grammatical way to fix the ambiguity:

S ::= if (E) s else S fi | if (E) S fi | T

If we want to group "else s2" with the inner if we can only write

```
if (e1)
    if (e2)
        s1
    else
        s2
    fi
fi
```

To link "else s2" with the outer if we must write

```
if (e1)
    if (e2)
        s1
    fi
else
    s2
fi
```

Each of these expressions leads to only one parse tree.

Sadly, this never caught on so our grammars remain ambiguous.