Derivation Time!

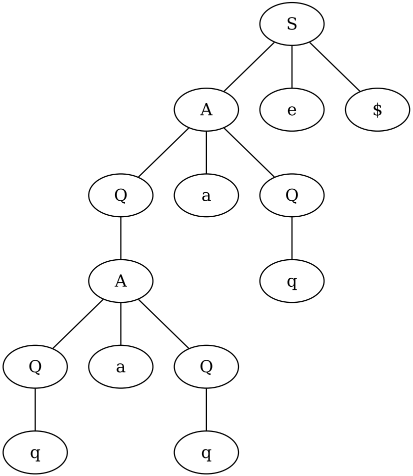
Find the parse tree for the following input: `q a q a q e`

$$
\begin{array}{rcl}
S & \rightarrow & A\ \$ \\
  & | & A\ e\ \$ \\
A & \rightarrow & Q\ a\ Q \\
Q & \rightarrow & q \\
  & | & A
\end{array}
$$

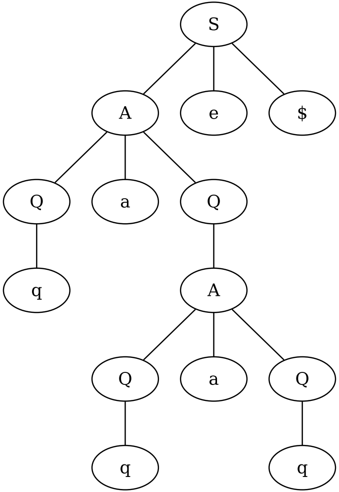
Derivation Time!

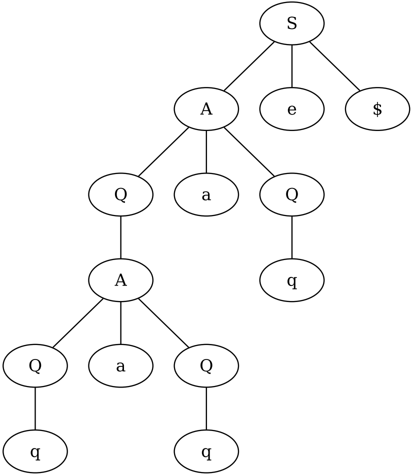
Find the parse tree for the following input: `q a q a q e`

$$
\begin{array}{rcl}
S & \rightarrow & A\ \$ \\
  & | & A\ e\ \$ \\
A & \rightarrow & Q\ a\ Q \\
Q & \rightarrow & q \\
  & | & A
\end{array}
$$



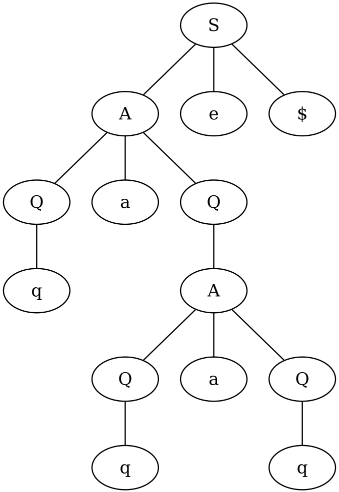
Which parse tree is correct for input: `q a q a q e`?

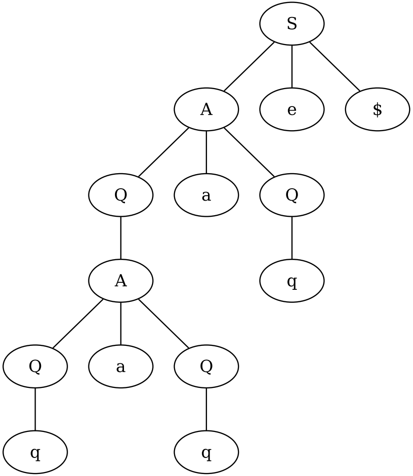




Which parse tree is correct for input: `q a q a q e`?

They both are, and **there's no humor in that!**





Ambiguity in Languages

Ambiguity When a grammar permits two different, **yet correct** parse trees for the same input.[1]

But we don't want ambiguity in our programming languages (imagine: the `MSVC++`, `g++` and `llvm` compilers all have different **runtime semantics** for [2]

```
++a - &--p->x.c % p->x.c++
```

[1] Ambiguity is usually fun in spoken languages, it is the basis for word-plays and puns.

[2] You'll end up launching the missiles when you just wanted to run off the coffee pot `:(`. You might as well be programming in Python~~2~~3.

Ambiguity in Languages

Ambiguity When a grammar permits two different, **yet correct** parse trees for the same input.[1]

But we don't want ambiguity in our programming languages (imagine: the `MSVC++`, `g++` and `llvm` compilers all have different **runtime semantics** for

```
++a - &--p->x.c % p->x.c++
```

Problematically: determining if an arbitrary grammar is ambiguous is an undecidable problem. You can't prove a grammar is ambiguous— why? Because there might be a parsing algorithm that corrects the ambiguity, you just don't know what it is.

Fortunately: this also means we can prove a language is NOT ambiguous. How? By demonstrating a parsing algorithm that permits only one interpretation of any language sentence.

[1] Ambiguity is usually fun in spoken languages, it is the basis for word-plays and puns.

Towards Automated Table-Driven Parsing Algorithms

And this is our motivation for pursuing *grammar analysis*,

we want to be confident that our programming language definitions are **unambiguous**,

along the way, we'll also benefit from a beautiful and pragmatic **separation of tasks**: the only thing linking the **grammar analysis** to the **parsing engine** will be a table of data.

The parsing engine will need to know the grammar's production rules P, but it won't need to perform contextual analysis of the grammar during the parse.