Programming Assignment 5 and last: Compositional semantics

Assigned: November 21
Due: December 10.

Overview

In this assignment, you will write a program that takes as input a parsed sentence and produces a symbolic representation.

Grammar

The context-free grammar is as follows:

Grammar
S → NP VP
NP → Name
NP → Det Adj* Noun
VP → Verb NP PP*
VP → Verb NP NP PP*
PP → Prep NP

Lexicon:
Adj: { big, nice, old, pretty, small }
Det: { a, an, the }
Name: { Amy, Bob, Carol, David }
Noun: { book, boy, flowers, girl, man, woman, store }
Prep: { for, from, to }
Verb: { gave, got, bought, sold }

The asterisks in Adj* and PP* means that there can be 0, 1, 2 ... Adjs/PPs. (This is known as the Kleene star, pronounced "Clay-ny".)

Representation

The representation that you will output will be as a set of atomic, ground formulas in the predicate calculus (that is, there are no quantifiers and no Boolean operators). When there are entities that are unnamed or implicit in the natural language text, you create a new constant symbol to represent them. Other than these new constant symbols, the non-logical primitives of the language are as follows:

Unary Predicates:
`Big(x), Nice(x), Old(x), Pretty(x), Small(x), Book(x), Boy(x) Flowers(x), Girl(x), Store(x), Man(x), Woman(x), Money(x), Transfer(e)`

Binary Predicates:
`Source(e,x), Destination(e,x), Object(e,x)`

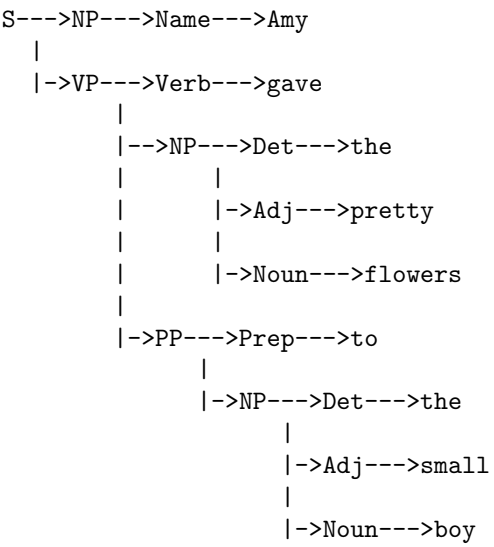
The verbs "gave" and "got" denote a single transfer event. The verbs "bought" and "sold" denote two transfer events: a transfer of merchandise and a transfer of money. The object of a transfer is the thing whose ownership is changing; the source is the original owner; and the destination is the final owner.

If X buys Y for Z from W, or X buys Z Y from W, then there are three transfer events: X transfer money to W, W transfers Y to X, and then X presumably will transfer Y to Z.

Input/Output format

The input format is a parenthesized form of the sentence, either read from input or passed as an argument in an interpreter. A subtree whose label at the root is L is represented by the expression L(subtree, ... subtreeK).

For instance the parse tree



is input as the string
`S(NP(Name(Amy)),VP(Verb(gave),NP(Det(the),Adj(pretty),Noun(flowers)),`
`PP(Prep(to),NP(Det(the),Adj(small),Noun(boy))))).`

If a person is not mentioned, then no constant need be created for it. For instance, in representing the sentence "David sold a book," you do not have to create a constant for the person he sold it to. However, with sentences involving "bought" or "sold", you do need a constant for the money, since that is what distinguishes "bought" and "sold" from "got" and "gave".

Ignore upper and lower case.

The grammar of course includes sentences that are semantically, pragmatically anomalous and even grammatically wrong e.g. "The flowers sold the big small big big boy to a old book." I don't care what happens with these; the program can do whatever is easiest for you. You may assume that the program will be tested on reasonable inputs. If you have questions about borderline cases, feel free to ask, but the answer will probably be, "I don't care."

The output is a set of atomic ground formulas, one per line. The order doesn't matter.

Examples

Input:
```
S(NP(Name(Amy)),VP(Verb(gave),NP(Det(the),Adj(pretty),Noun(flowers)),
PP(Prep(to),NP(Det(the),Adj(small),Noun(boy))))).
```

Output:
```
Transfer(X1).
Pretty(X2).
Flowers(X2).
Small(X3).
Boy(X3).
Source(X1,Amy).
Destination(X1,X3).
Object(X1,X2).
```

Here `X1, X2, X3` are constant symbols that the program creates to denote the event, the flowers, and the boy, respectively.

Input:
```
S(NP(Det(The),Noun(boy)),VP(Verb(bought),NP(Det(the),Adj(old),Noun(book)),
PP(Prep(from),NP(Det(the),Adj(old),Noun(man)))))
```

Output:
```
Transfer(X1).
Transfer(X2).
Money(X3).
Boy(X4).
Book(X5).
Old(X5).
Man(X6).
Old(X6).
Source(X1,X4).
Destination(X1,X6)
Object(X1,X3).
Source(X2,X6).
Destination(X2,X4).
Object(X2,X5).
```

Input:
```
S(NP(Name(David)),VP(Verb(bought),NP(Name(Amy)),NP(Det(a),Adj(small), Adj(nice),Noun(book))))
```

Output:
```
Transfer(X1).
Transfer(X2).
Transfer(X3).
Money(X4).
Book(X5).
Small(X5).
Nice(X5).
Source(X1,David).
Object(X1,X4).
Destination(X2,David).
Object(X2,X5).
Source(X3,David).
Destination(X3,Amy).
Object(X3,X5).
```