Tree Languages and Context Free Grammars

Lecture Note 2 for COM S 474
Mats Rooth

Formal tree languages

In the conception adopted in formal language theory, a language is a set of structured objects, such as strings. For instance, the elements of the language described by the regular expression *ab** are those character strings which have length at least one and have *a* in the first position, and *b* in every other position.

Example The following strings are elements of the language *ab**.

a, ab, abb, abbb, abbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

The following strings are not elements of that language.

b, ba, abc

Example Let N be the set {covers,doors,books,tables,carpets,floors,windows},and let P be the set {by,near,on,with,under}. Let A be the set of strings which have elements of N in all odd positions (starting with the first position), elements of P in all odd positions, and an odd length. Then the following strings are elements of A.

books
books in boxes
books in boxes under tables in boxes

Languages corresponding to regular expressions are called *regular languages*. It is possible to describe some interesting fragments of English syntax with regular expressions. In applications and research, one often wants to scan large bodies of text for structures of some specific kind, for instance corporation names or names of chemical compounds. This can be done efficiently with regular expressions defining the phrases one is interested in, and regular expression matching algorithms which find the instances.

Example Let P be the language defined by the regular epression [books | tables | boxes] [[on | in | under] [books | tables | boxes]]*. The following strings are elemtnst of P.

boxes
books in boxes under tables
books in boxes under tables on books

In this example, an element of the language P is a string of words. An alternative approach to the same data is to take the elements of the language to be labeled trees. A language like this is called a *tree language*.

Example Let Q be the set of all finite ordered, rooted, labeld trees which satisfy the following properties.

(i) Terminal vertices have labels in the set {books, windows, tables, on, near, under}.

(ii) Non-terminal vertices have labels in the set {PP,IN,NNS}.

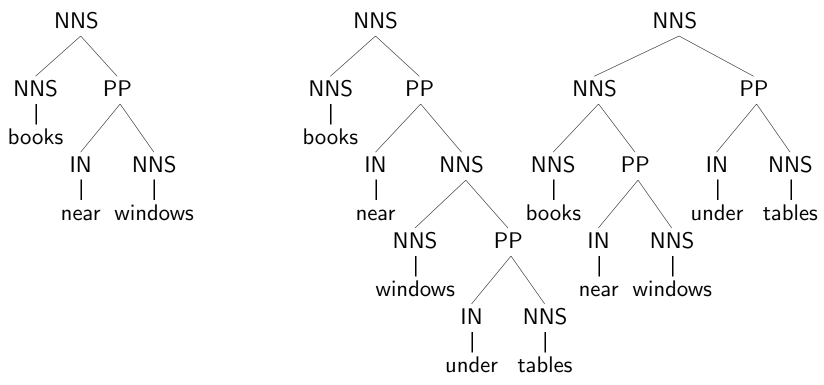
(iii) If v is a vertex with label PP, then v has two children, the first of which has label IN, and the second of which has label NNS.

(iv) If v is a vertex with with label NNS, then either v has one child, which has a label in the set {books, windows,tables}, or it has two children, which have (in order) labels NNS and PP.

(v) If v is a vertex with label IN, then v has one child, which has a label in the set {on, near, under}.

(vi) The root has label NNS.

Then the following trees are elements of Q.



This example illustrates the idea of working with tree languages consisting of syntactic trees for phrases of English or another natural language. The radical extension of this is to collect the syntactic trees for all possible English sentences into a set E

It is clear that E is a large set. Our vocabularies consist of tens of thousands of words, which combine combinatorially, resulting in an exponential growth in the number of sentences, as sentence length increases. And people often use sentences which have never been used before---I would guess that 90% of the sentences in this lecture have never been used before, even by me.

.

.

If we use the Treebank convention for English syntax, then the following tree is an element of E. Arguably, E is countably infinite. Patterns such as the following can be extended seemingly without bound.

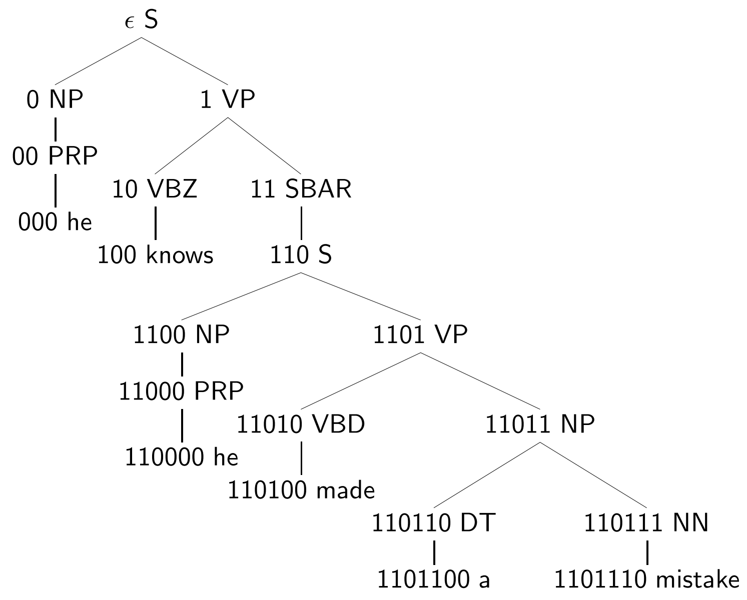
This is the dog that worried the cat that chased the rat that ate the carrot that lay in the house that Jack built.

Even long sentences of this pattern are comprehensible. It seems arbitrary to say that, at some point, sentences of this pattern are no longer English sentences. If we conclude that accepts is no such point, then English contains a countably infinite set of sentences of this pattern.

Tree Domains

In reasoning about tree languages and natural language syntax, it is useful to work with a concrete construction of rooted, ordered, labeled trees. We start with a construction which captures the shape of trees, and then add the labels.

The *address* of a vertex in a tree is a path specifying the route from the root to the vertex. Here is a syntactic tree drawn with addresses, in addition to the labels.



An address is a finite sequence of non-negative integers. The address of the root is the empty sequence ε . If a is an non-terminal address in the tree, then the address of the first child of *a* is *a*0. The address of the second child (if there is one) is a1, and the address of the *i*th child (counting from 0) is *ai*.

A tree domain is a tree (without labels) constructed as a set of addresses, by imposing two closure properties. In the definition, *N* is the set of natural numbers including 0, *N** is the set of finite sequences of natural numbers, including the empty sequence, the variables i and k range over N, and the variable α ranges over N*.

Definition A tree domain X is a subset of N^* satisfying the following conditions.

(i) If $\alpha k \in X$ and $0 \leq i < k$ then $\alpha i \in X$.

(ii) If $\alpha i \in X$ then $\alpha \in X$.

The first condition ensures that local addresses range upward from 0. The second condition ensures that an address is in the domain only if it is an extension (i.e. a child) of an address which is also in the domain.

Labels are represented as using using a function on the tree domain. Since the domain is recoverable from the function, the labeled tree can be identified with that function.

Definition A labeled tree t is a function such that *Dom(t)* is a tree domain.

Example The tree drawn above is the finite function t defined by the following table of arguments and values.

TABLE 1.

x	*t(x)*	*x*	*t(x)*	*x*	*t(x)*	*x*	*t(x)*
ε	S	10	VBZ	11000	PRP	11011	NP
0	NP	100	knows	110000	he	110110	DT
00	PRP	11	SBAR	1101	VP	1101100	a
000	he	110	S	11010	VBD	110111	NN
1	VP	1100	NP	110100	made	110111	mistake

Note that, if t is a labeled tree constructed in this way as a function, then the corresponding unlabeled tree is the domain of the funtion t. This justifies the terminology "tree domain".

Context Free Grammars

Given that languages such as E are countably infinite sets, or if finite are very large sets, one is faced with the problem of defining the sets in some reasonably compact way. This is usually done with a compuational or discrete-mathematical *grammar formalism* in which particular tree languages are defined. Grammar formalisms are a medium for scientific investigation of natural languages, and a basis for manipulating natural languages compuationally. In computational research, these aims have to be balanced: one wants to work with a formalism which is expressive enough to capture a lot of linguistic reality, but which also has the advantages of computational tractability. We will look at several grammar formalisms in these lectures, starting in this lecture with context free grammars.

The core part of context free grammar is a set of *productions* or *local trees*. Here is the production which describes the most common shape for and English prepositional phrase:

PP ---> IN NP

A production consists of a left-hand side which is a single symbol, and a right hand side which is a finite sequence of symbols. Where u is the production above, we write

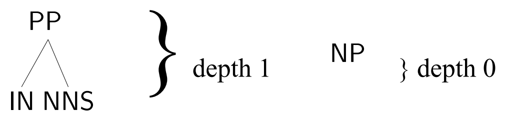
\

$$lhs(u) = PP \qquad rhs(u) = \langle IN, PP \rangle .$$

Productions with empty right hand side are allowed, for instance the production *v* described by

$$lhs(v) = NP \qquad rhs(v) = \varepsilon .$$

An alternative way of drawing the same production is as a tree of depth one, or zero in the case of a production whose right hand side is the empty string:



Here *lhs(u)* is the single parent, and *rhs(u)* is the ordered sequence of children in the tree. Or saying it in the formalism of tree domains, the production *u* on the left is the finite function with domain $\{\varepsilon, 0, 1\}$ such that $u(\varepsilon) = PP$, $u(0) = IN$, and $u(1) = NP$. The production v on the right is the finite function with domain $\{\varepsilon\}$ such that $u(\varepsilon) = NP$.

Definition 1. A *production* is a tree whose domain consists of addresses with length one or zero.

In the formal definition of context free grammars given below, the set of productions of the grammar is represented in a set P.

Definition 2. A context free grammar is a tuple $\langle N, \Sigma, P, R, L \rangle$, where

(i) N is a finite set (the non-terminal symbols)

(ii) Σ is a set (the set of terminal symbols, not necessarily finite)

(iii) P finite set of productions, with labels in N.

(iv) R is a subset of N (the set of root or start symbols), and

(v) L is a relation between Σ and N.

Definition 2 differs from the standard definition of context free grammars in a couple of ways which are motivated by the natural language application. First, it allows for several root or start symbols. This is because it is useful to allow for several distinct root symbols in natural language trees. Second, there are no productions in P introduce elements of Σ. The motivation for this some natural languages arguably have countably infinite terminal vocabularies. The definition allows for an infinite Σ and L, while still keeping the set of non-terminal productions P finite.

Example Let G be the tuple $\langle N, \Sigma, P, R, L\rangle$, where

N = {PP,IN,NNS}.

Σ = {books, windows tables, in, near, under}

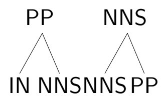
L = {<books,NNS>,<windows,NNS>,<tables,NNS>,<in,IN>,
<near,IN>,<under,IN>}

R = {NNS}

$P = \{u,v\}$ where $Dom(u) = Dom(v) = \{\varepsilon, 0, 1\}$, Dom(u) = Dom(v) = {e,0,1},
u(ε)=PP, u(0)=IN,u(1)=NNS, v(ε)=NNS, v(0)=NNS, and v(1)=PP.

Then G is a context free grammar.

The line describing *P* is a recondite way of saying that *P* consists of these two local trees:



Lab Note

File format for a CFG corresponds directly to Definition 2. P is given in a file with extension *gram*. Here is pp.gram:

```
PP IN NNS
NNS NNS PP
```

R is given in a file with extension *start*. Here is pp.start:

```
NNS
```

Finally, L is captured in a file which lists a word form on the left of each line, and the possible parts of speech to the right on the same line. Here is pp.lex:

```
books NNS
windows NNS
tables NNS
in IN
near IN
under IN
```

(Though it doesn't look like it, the words are separated from the tags by a tab. Subsequent tags are separated by a space, so that if *under* was ambiguous between a preposition and an plural noun, the last line would be *under-tab-IN-space-NNS*.)