

Short Tutorial on Generative Grammars

- A GENERATIVE GRAMMAR is given by:
 - {NTE, TE, S, P} where
 - NTE : nonterminal symbols (usually capital letters)
 - TE : terminal symbols (usually lowercase underscored strings)
 - S : Start symbol (usually S)
 - P : set of productions like
$$X \rightarrow Y \mid A B C \mid Z$$
which means symbol X can be replaced by the string "Y" or "A B C" or "Z" in the current sentence that is being generated.
You can use any of these 3 options for replacement.
- A grammar like this can be used to generate a variety of sentences starting with S, and repeatedly using any of the productions which are applicable.

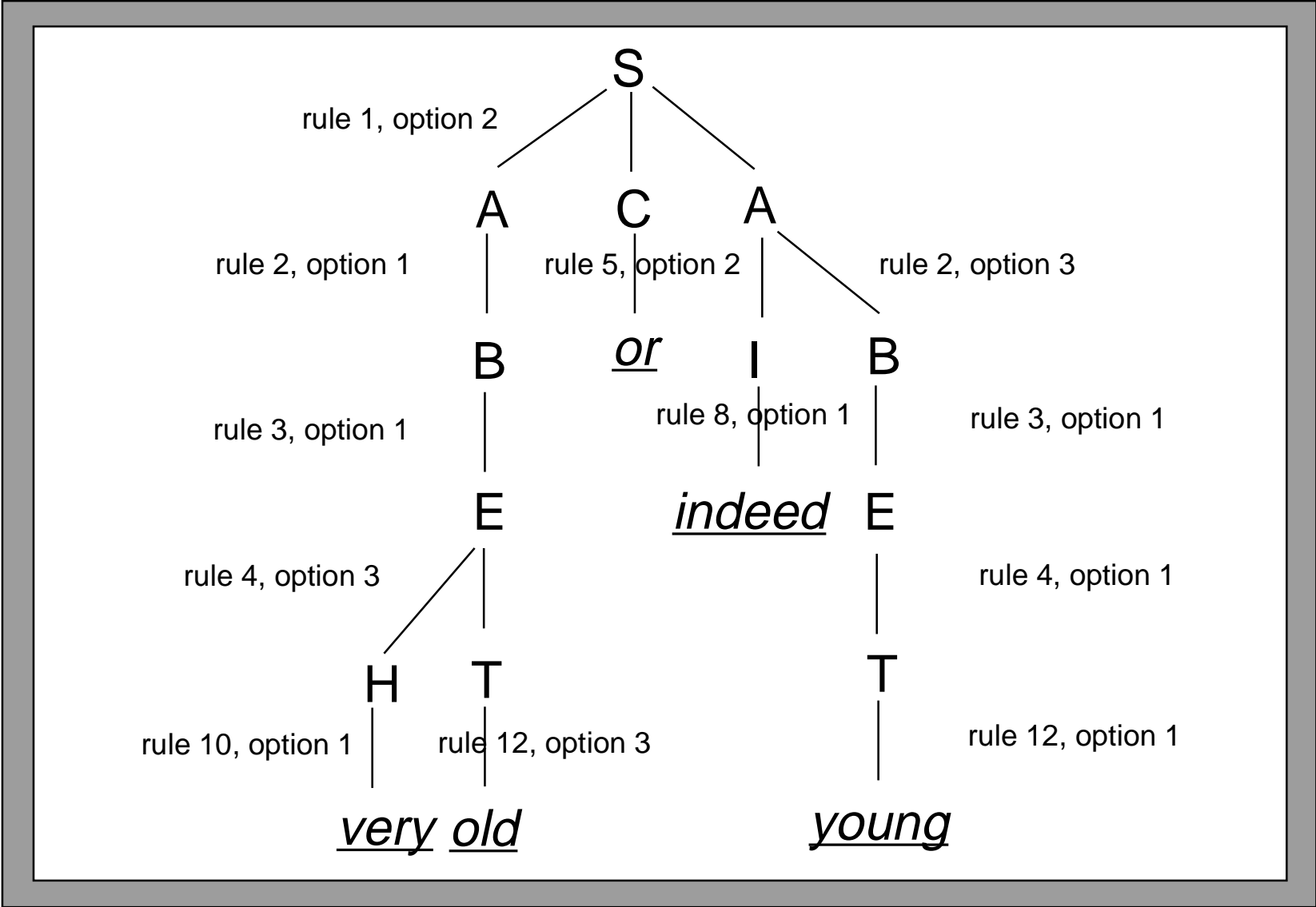
Grammar

- (Rule 1): S-> A | A C A | U
- (Rule 2): A-> B | R(B)(B) | I B
- (Rule 3): B-> E | N E | D E
- (Rule 4): E-> T | H T | V T
- (Rule 5): C-> and | or
- (Rule 6): R-> in-between | from-to
- (Rule 7): N-> not
- (Rule 8): I -> indeed | hardly
- (Rule 9): D-> younger-than | older-than
- (Rule 10): H-> very | more-or-less
- (Rule 11): V-> extremely
- (Rule 12): T-> young | middle-aged | old
- (Rule 13): U-> any-age | no-age | unknown-age

Short Tutorial on Generative Grammars

- You initialize the current string as start symbol S.
- Replace any nonterminal symbol in this string by using a production rule that has that symbol on its left hand side.
- For instance,
 - S gets replaced by A C A.
 - Then A gets replaced by B, C by or, and the second A by I B. The I then gets replaced by indeed, the second B by E, which is replaced by T, and the T by "young".
 - The first B gets replaced by E, which gets replaced by H T; H gets replaced by very, and the T gets replaced by old.
- At each stage the process looks like the following production tree:

Production Tree



Short Tutorial on Generative Grammars

- S start symbol
- A C A apply rule 1, option 2
- B C A rule 2, option 1
- B or A rule 4, option 2
- B or I B rule 2, option 3
- B or indeed B rule 8, option 1
- B or indeed E rule 3, option 1
- B or indeed T rule 5, option 1
- B or indeed young rule 12, option 1
- E or indeed young rule 3, option 1
- HT or indeed young rule 4, option 2
- very T or indeed young rule 10, option 1
- very old or indeed young rule 12, option 3

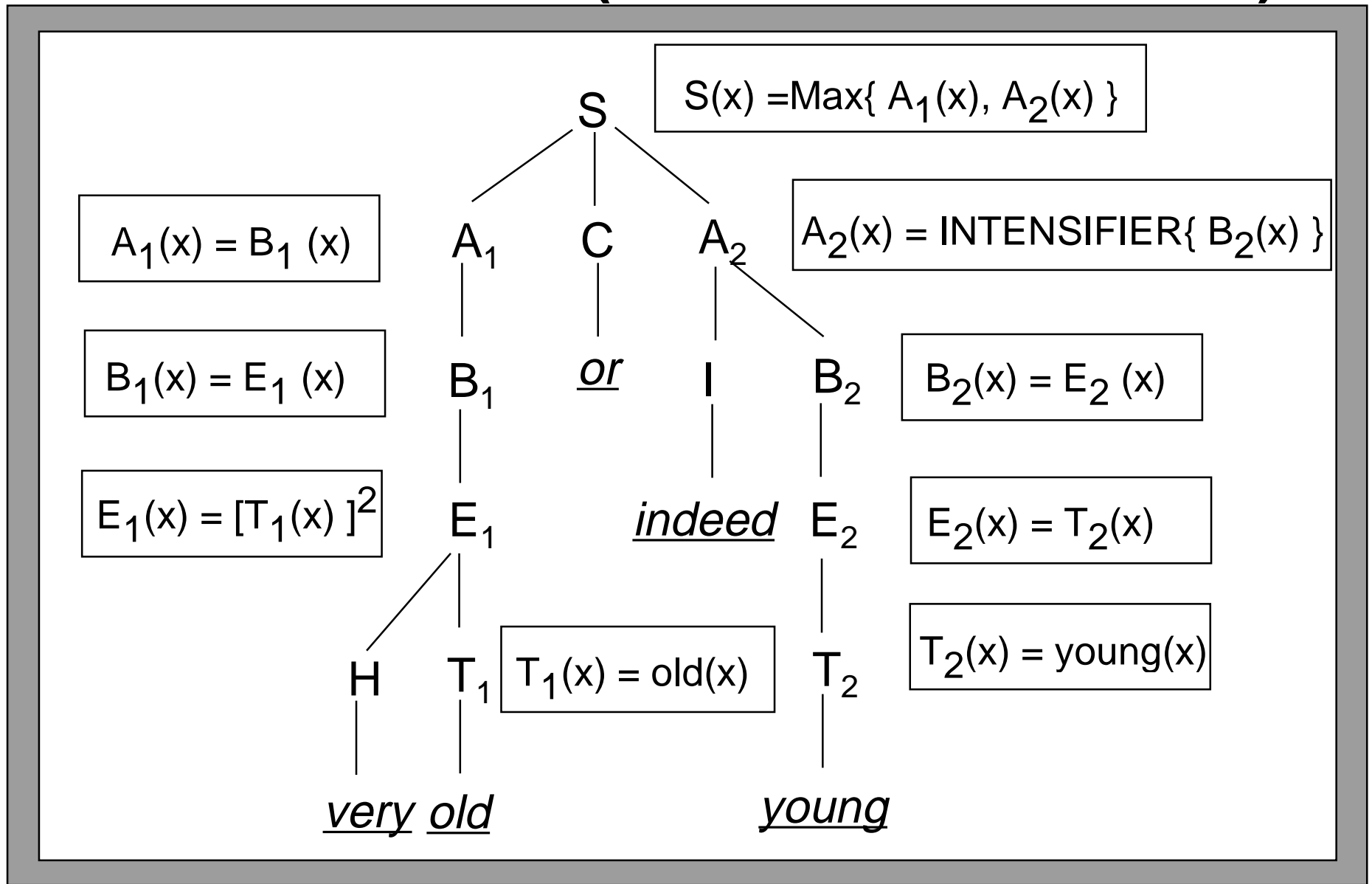
Short Tutorial on Generative Grammars

- This process also needs to keep track of which rules were applied where, and you can do that by the derivation tree, which I talked about in class. So that is what the sentence generator does.
- Everytime a production rule is applied, there is also an implicit meaning transformation.
 - For instance, when I applied $A \rightarrow I B \rightarrow \dots \rightarrow \textit{indeed young}$ above, I can figure out that the meaning of A.
 - In this case, $\mu A(x)$ derives from $T_I(\mu \textit{young})$, which is obtained by applying the transformation associated with "indeed" (T_I) to the fuzzy set represented by "young"
 $\mu \textit{young}(x)$.

Short Tutorial on Generative Grammars

- This sort of reasoning can be automated if you have the derivation tree. Each leaf of the derivation tree is a terminal string. It is either a primary term or a linguistic hedge or an operator.
- In all these cases, it is easy to replace it by the appropriate functional transformation on the membership functions.
- Bubbling this process up leads to a $\mu S()$ function which is the meaning of the sentence. E.g.:
 - B C A got changed to B or A above, so
 $\mu (B C A)() = \max \{ \mu B() , \mu A() \}$ is the transformation applied at the node where C is replaced by "or".

Production Tree (Semantic Constraints)



Short Tutorial on Generative Grammars

- That's the semantic interpretation part.
- Try to work out some of this procedure by hand to make sure you understand it, then automate it by writing a program to do it.
- If you want to avoid the nitty-gritty of implementation, you could try using lex and yacc instead (UNIX tools which generate code for you).
- My purpose is not to get you to hack compilers but to understand how to easily manipulate fuzzy sentences.