Homework 4: Parsing

Chuan Lin, Kai Zhang

October 21, 2024

1.a

For this question, I choose the Stanford Parser. See Figure 4

A very interesting thing about the style of the tree is that the parser takes a verb phrase with infinite tense as a sentence. We can clearly see that in the parse tree of "My mother have called me to go back", the root of the subtree for "to go back" is "S".

1.b

For this question, I choose the Stanford Parser. See Figure 2

One hard thing that is hard to manage and makes parser wrong is ambiguous attachment. In the parse tree for "I see the man with a telescope", the attachment "with telescope" is used to describe the vert "see" instead of the noun "man". However, we can clearly see that the parser fails to manage this point.

1.c

For this question, I choose the Stanford Parser. See Figure 3 One grammatical sentence that confuses the parser is the sentence that have multiple relative clauses in a row while these clauses describes different things.

For example, in the parse tree for "They discussed the dogs on the beach with their owners", the parser believes that both "on the beach" and "with their owners" describe the noun "the dogs". However, the truth is that "with their owners" is used to describe the verb "discussed".

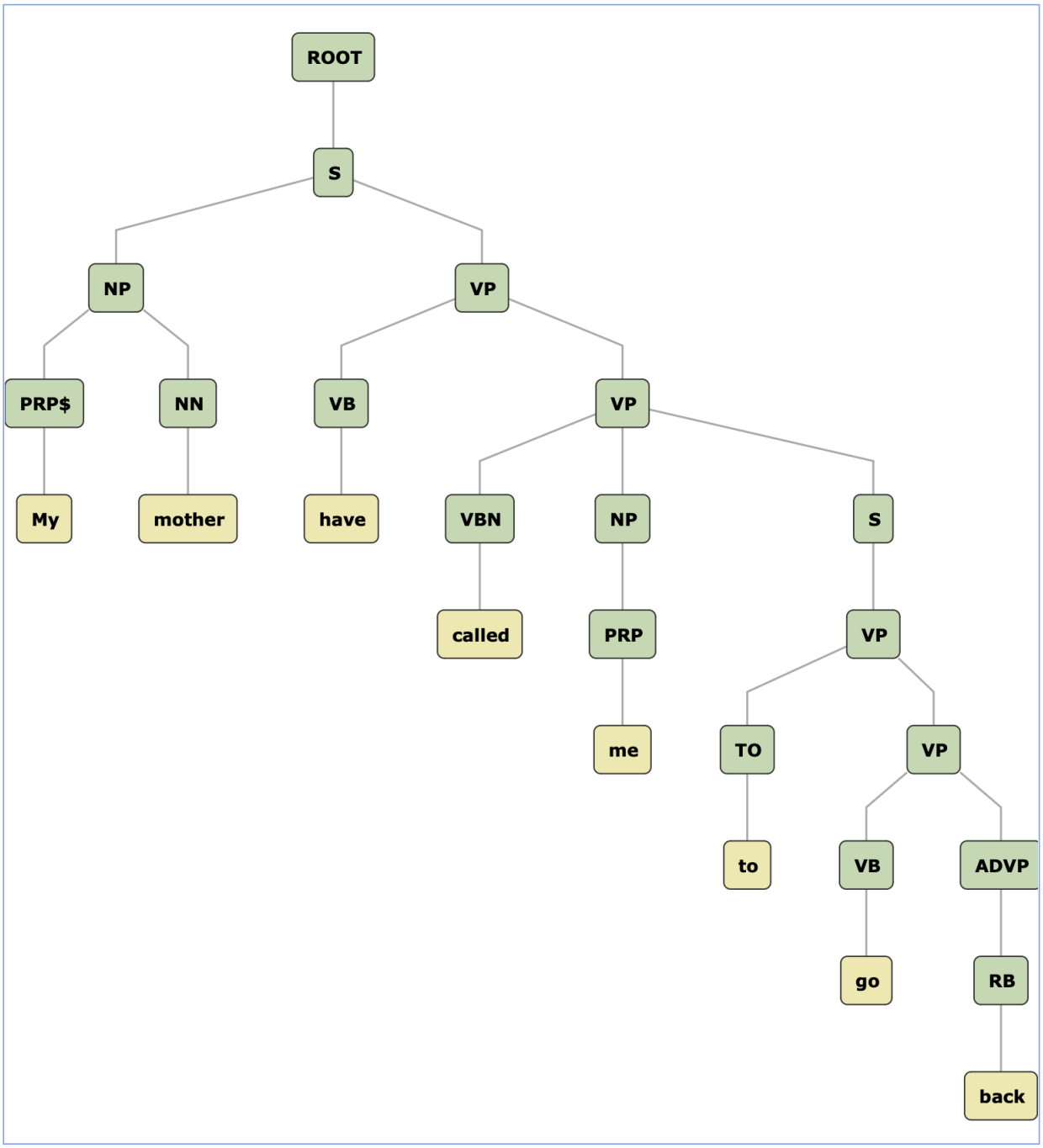
ROOT
S
NP
VP
PRP$
NN
VB
VP
My
mother
have
VBN
NP
S
called
PRP
VP
me
TO
VP
to
VB
ADVP
go
RB
back

Figure 1: 1-a

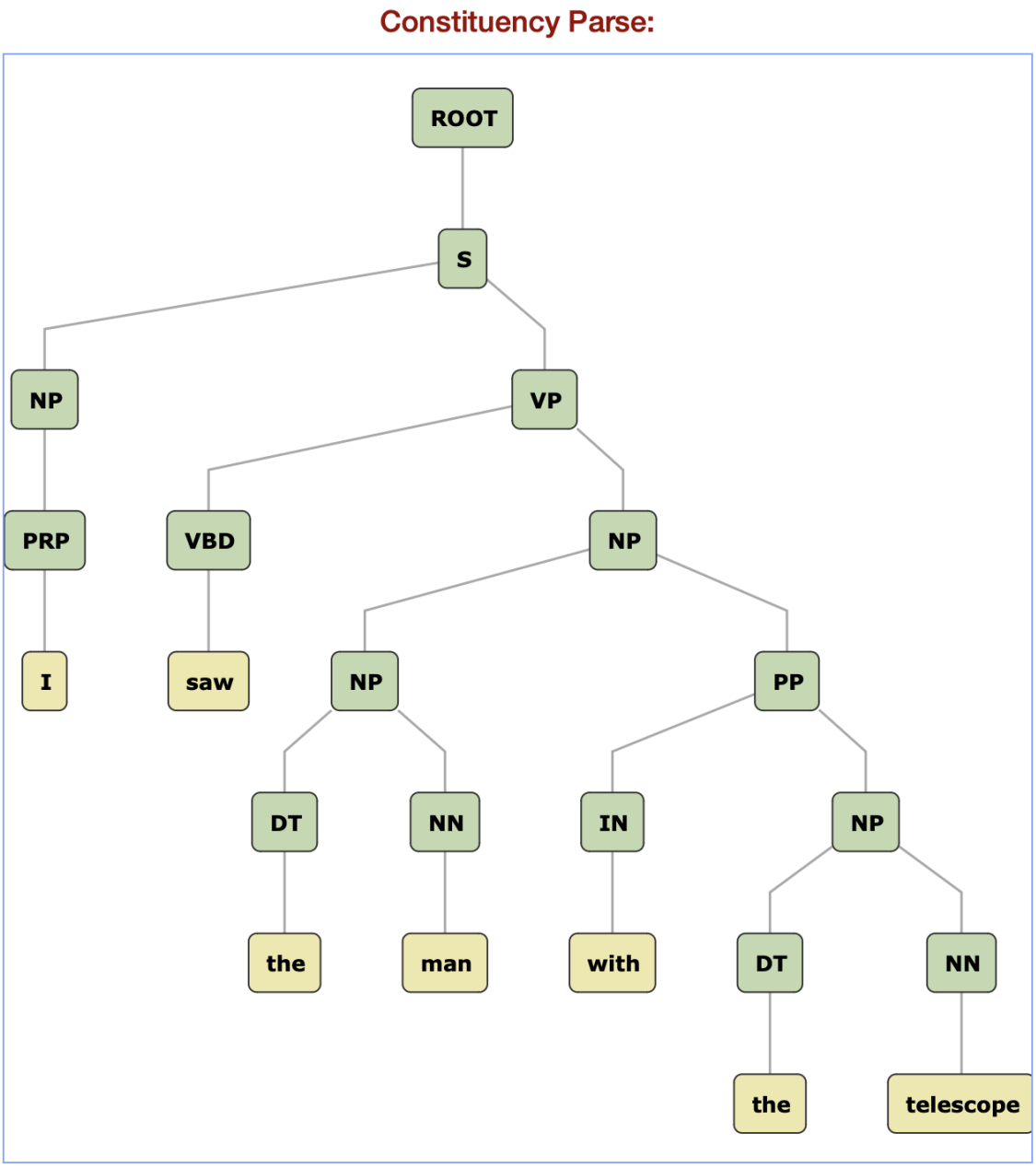
Constituency Parse:
ROOT
S
NP
VP
PRP
VBD
NP
I
saw
NP
PP
DT
NN
IN
NP
the
man
with
DT
NN
the
telescope

Figure 2: 1-b

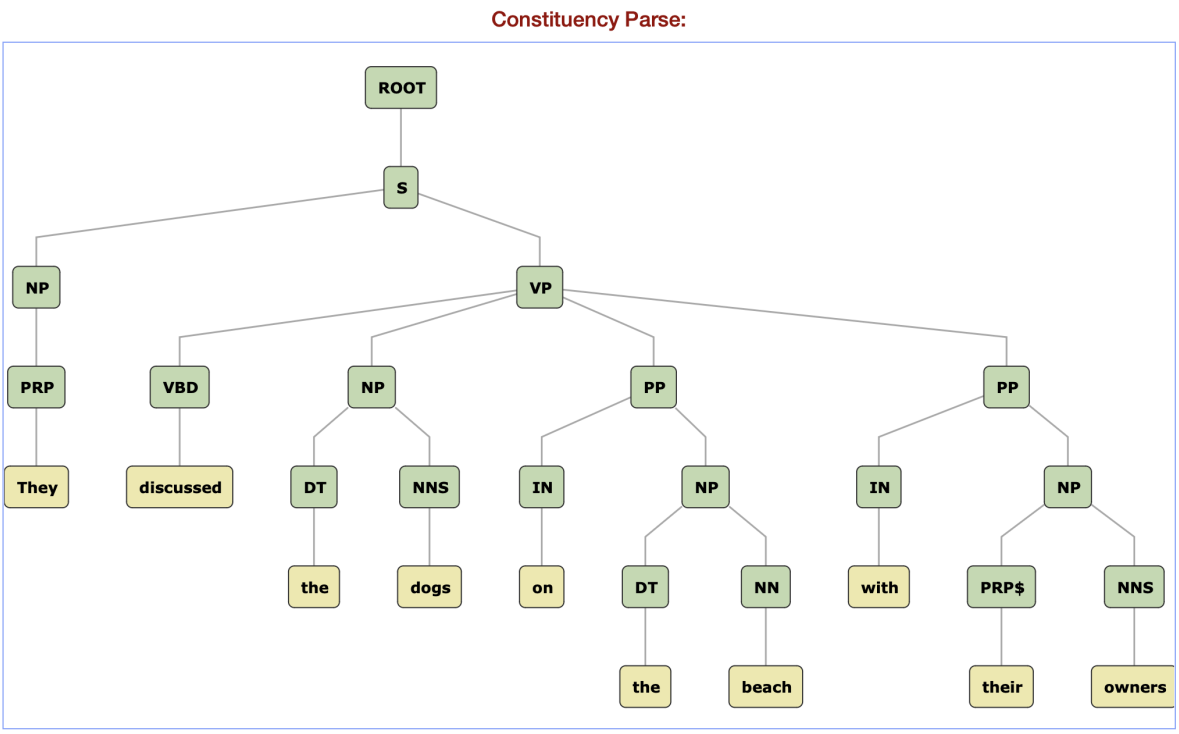
Constituency Parse:
ROOT
S
NP
VP
PRP
VBD
NP
PP
PP
They
discussed
DT
NNS
IN
NP
IN
NP
the
dogs
on
DT
NN
with
PRP$
NNS
the
beach
their
owners

Figure 3: 1-c

2.a

Let's compare the five parsers with the example sentence "The very biggest companies are not likely to go under .", which is the second sentence in wallstreet.sen:

1. Syntactic Structure

- Earley (CFG):
  Breaks the sentence into standard phrase structures: Noun Phrase (NP), Verb Phrase (VP), etc.

  $(S$
  $(NP\,(DT\,The)\,(ADJP\,(RB\,very)\,(JJS\,biggest))\,(NNS\,companies))$
  $(VP\,(VBP\,are)\,(RB\,not)\,(ADJP\,(RB\,likely))\,(VP\,(TO\,to)\,(VP\,(VB\,go)\,(PP\,(IN\,under))$
  $)$
  Example: "The very biggest companies" is parsed as an NP, and "are not likely to go under" as a VP.
- Dependency Grammar (spaCy):
  Focuses on relations between words rather than breaking the sentence into phrases.

  companies (head) ← biggest (amod) ← very (advmod)
  are (head) ← companies (nsubj)
  likely (head) ← not (neg) ← are (acomp)
  Example: "companies" is the head of the subject, with "very biggest" modifying it. "are" is the main verb, and "likely" is a complement of "are."
- Link Grammar (Link Parser):
  Connects words based on grammatical links without traditional phrase structures.

  $+$–Ds–$+$ $+$–Ss–$+$ $+$–MV–$+$ $+-Pa-+$
  The companies are not likely to go under
  | | | |
  $+$–Xd–$+$ $+-Oe-+$ $+$–R–$+$
  biggest very
  Example: "The" links to "companies" (determiner), "are" links to "companies" (subject-verb), and "likely" links to "go."
- HPSG (PET):
  Builds rich feature structures for each word, linking them through head-driven rules. • Example: • "companies" (head of NP), • "likely" is an adjective complement modifying "are", • The verb "go" takes "under" as a prepositional argument.

$$\text{The} \xrightarrow{\text{DET}} \text{companies} \quad \text{are} \xrightarrow{\text{AUX}} \text{likely} \quad \text{to} \xrightarrow{\text{INF}} \text{go} \xrightarrow{\text{PP}} \text{under}$$

2. Heads and Dependencies

   - Earley (CFG):
     No explicit head concept. The parser generates a tree based on grammar rules, without specific attention to which word is the "head."
     Example: The subject "companies" and the verb "are" are separate nodes in the tree.
   - Dependency Grammar (spaCy):
     Focuses on head-dependency relations. Each word depends on another word, with a single head for each dependent.
     Example: "companies" is the head of the subject, and "are" is the head of the sentence, linking to "not" and "likely."
   - Link Grammar (Link Parser):
     Does not explicitly identify heads. Links between words represent grammatical relations.
     Example: "are" is linked to both the subject "companies" and the complement "likely," but no explicit head-dependency structure.
   - HPSG (PET):
     Head-driven structure. Each phrase has a clear head (e.g., verb, noun) that drives the structure of the phrase.
     Example: "companies" is the head of the noun phrase, and "are" is the head of the verb phrase.

3. Handling Word Order and Flexibility

   - Earley (CFG):
     Relies on fixed rules for word order based on context-free grammar.
   - Dependency Grammar (spaCy):
     Flexible with word order because the dependencies focus on relations, not positions.
   - Link Grammar (Link Parser):
     Can handle non-standard word orders, as it focuses on word links.
   - HPSG (PET):
     Highly flexible due to unification-based parsing. It can handle word orders like questions or topicalization.

Summary of Comparison

1. **Earley Parser (CFG)**

   - **Syntactic Structure**: Phrase-structure grammar, outputs traditional NP/VP hierarchical trees.
   - **Heads and Dependencies**: No explicit head information, relies on phrase structure.

- **Word Order Flexibility**: Fixed word order is assumed based on predefined grammar rules.
- **Lexicalization**: Not lexicalized, focuses on syntactic categories (e.g., NP, VP).
- **Ambiguity Handling**: Generates multiple parse trees for different interpretations.
- **Semantics**: Purely syntactic, semantics need to be added separately.
- **Efficiency**: Can be slower, especially with complex sentences and grammars.

2. **Dependency Grammar (spaCy)**

   - **Syntactic Structure**: Head-dependent relations, outputs dependency trees.
   - **Heads and Dependencies**: Explicit head-dependent relationships, e.g., "companies" is the subject of "are."
   - **Word Order Flexibility**: Word order is flexible, and structure is determined by relations, not position.
   - **Lexicalization**: Partially lexicalized; word relations depend on their grammatical roles.
   - **Ambiguity Handling**: Handles ambiguity with multiple dependency structures if necessary.
   - **Semantics**: Captures semantic relationships through the dependency structure.
   - **Efficiency**: Very fast, optimized for real-time applications.

3. **Link Grammar (Link Parser)**

   - **Syntactic Structure**: Uses links between words instead of phrases, no explicit tree structure.
   - **Heads and Dependencies**: No explicit heads, structures rely on linking patterns between words.
   - **Word Order Flexibility**: Flexible word order, as long as valid link patterns exist between words.
   - **Lexicalization**: Highly lexicalized; parsing is driven by word-pair link rules.
   - **Ambiguity Handling**: Can generate multiple link structures to resolve ambiguities.
   - **Semantics**: Implies semantic relationships through links but does not explicitly represent them.
   - **Efficiency**: Generally efficient, but complex sentences with many link possibilities can slow it down.

4. **Head-driven Phrase-Structure Grammar (HPSG, PET)**

   - **Syntactic Structure**: Feature-based grammar, combines syntax and semantics using feature structures.

- **Heads and Dependencies**: Explicit head-driven structures, where heads control agreement and roles.
- **Word Order Flexibility**: Flexible, allows for various word orders using feature-based rules.
- **Lexicalization**: Fully lexicalized, with rich feature structures assigned to words.
- **Ambiguity Handling**: Handles ambiguity through multiple feature structures for words/phrases.
- **Semantics**: Rich semantic representation built directly into the feature structures.
- **Efficiency**: Can be slower due to the complexity of feature unification and detailed grammatical representation.

3.a Correctness

As we know, the Item class in recognize.py represents dotted rule (each Item instance is a dotted rule). To keep track of best derivation and its total weight, we build a class Tip. For each Item instance, we attach a Tip instance to it, which records the total weight and backpointers of the best derivation of this Item. If we find a better derivation for an Item instance, we create a new Tip instance and attach it to our Item instance. Besides, since this Item instance has a better Tip, which means that we need less weight to derive this Item instance, we need to reprocess this Item, which may cause some earlier Item instances to be attached with new Tip instances.

To ensure the correctness of finding the best derivation, we need to carefully design how to create these Tip instances. When a new item is created, its tip is created based on how we create this new item (namely, attach, predict, or scan.) When a existing item is pushed into the same agenda again, we need to check whether we should attach a new tip for it.

3.b Efficiency

In the following answer, we assume that we get lucky and never need to reprocess items. Besides, we assume that we have G dotted rules in total.

Firstly, we show why our algo runs in $O(n^2)$ space. As we know, in agenda, each item is a dotted rule. If we push an existing item to the agenda, nothing will happen for agenda. Thus, each agenda has at most $O(nG)$ items. Considering that we have $O(n)$ agendas, we have $O(n^2G)$ items at most. Besides, the space for each item is $O(1)$ considering that each item is only made up of three numbers (start position, dot position, rule number). This we need a space of $O(G * n * n) = O(n * n)$ for all items. Since the space for items dominates all the space that we need, the space of our algo is $O(n^2)$.

Secondly, we show why our algo runs in $O(n^3)$ time. As we know, except for the first item in our algo ($0\ \mathit{Root}\ \rightarrow\ .\ S$), each item is created and pushed into agenda by processing an exiting item. If the process is PREDICT, it takes O(1) time. If the process is SCAN, it also takes $O(1)$ time. If the process is ATTACH, it takes $O(nG)$ time because we need to check each item of the agenda at the start position of the processed item, which may has $O(nG)$ items. As a result, to process an item, we need $O(nG)$ time. As mentioned above, we have $O(n^2G)$ items and each item only needs only one process based on the assumption. The total time for processing is $O(nG * n^2G) = O(n^3)$. Since the time for processing the items dominate the time for our algo, the time for our algo is $O(n^3)$

4 Speedup Methods for parse2

For speeding up running time of parse2 on wallstreet.gr, we've implemented several methods described in Appendix E.
For running time comparison, due to the performance limit of our PCs, We'll use the first two sentences in **wallstreet.sen** for all the speedup evaluations. Firstly, the baseline algorithm takes about 100 seconds to parse the sentences as shown in Figure 4.

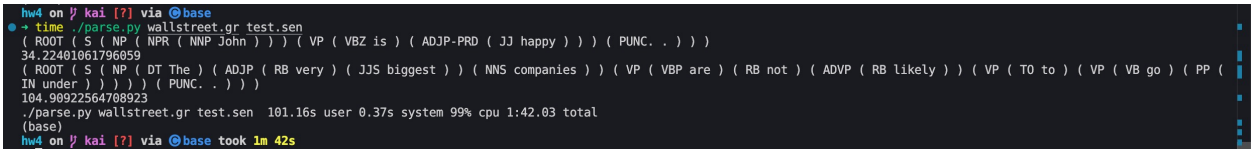

Figure 4: baseline efficiency of our parser

E.2: Vocabulary Specialization

We observed that there are tons of terminal-related rules in **wallstreet.gr**, which will cost too much time when parser tries to predict. So we initialize another dictionary in Grammar class to filter the related rules. The experiment shows that this speedup by itself can achieve 34 seconds, i.e. around 3x times faster, compared to the baseline.

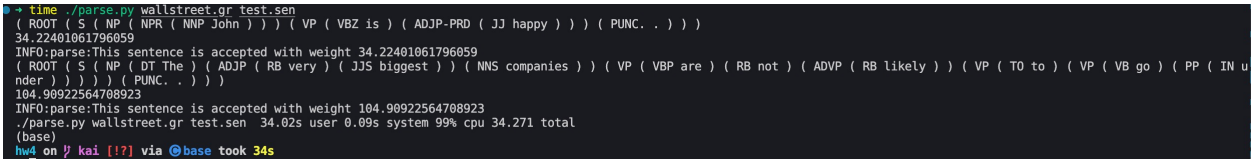

Figure 5: baseline+e2 efficiency of our parser

E.1: Batch Duplicate Check

This speedup is quite easy, we simply add a set for Earley algorithm to store the tuple of non-terminal and its position (i.e. the column index), and prevent the predict action when the tuple is found already in the set. The experiment shows that this speedup by itself can achieve 33 seconds, i.e. around 3x times faster too, compared to the baseline.

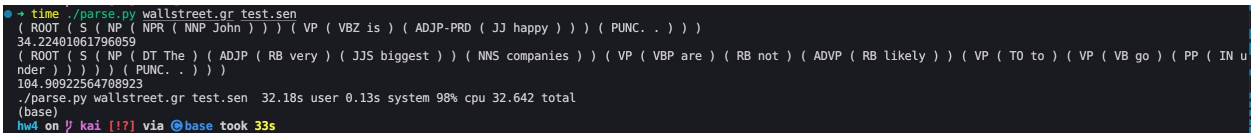

Figure 6: baseline+e1 efficiency of our parser

E.5: Indexing Customers

Actually, this speedup is quite interesting to implement, especially combined with E.1. We can add a dictionary to record the list of items in each Agenda with the key of the next symbol. The dictionary has the following multiple functions:

- when attaching, we can directly inquiry the dictionary of specific agenda with the LHS of attached rule, thus eliminating the original linear search of customers in the entire agenda
- when predicting, remember we want to eliminate all duplicate predict action for non-terminal, i.e. we only allow the first prediction action for all non-terminals. And if we maintain this dictionary properly, we'll find that the prediction will take place iff it is the second item is still valid, or the inquired list only contain one item.

The experiment shows that this speedup by itself can achieve 77 seconds, i.e. around 1.33x times faster too, compared to the baseline. The improvement of E.5 will be much more prominent when combined with other speedups, because the cost of recording these dictionaries will be much lower while its improvement still remains the same.

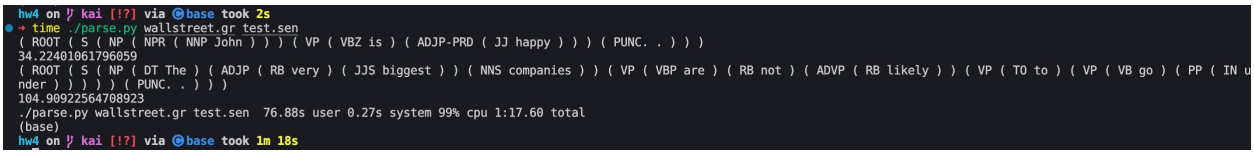

Figure 7: baseline+e5 efficiency of our parser

Overall Optimization

Finally, by combining all these speedups, our model achieves 4 seconds, i.e. around 24x times faster.
And when evaluated on **wallstreet.sen** and with Apple processor M1Pro, our model achieves around 25 minutes to finish all the sentences, as shown in Figure 8&9

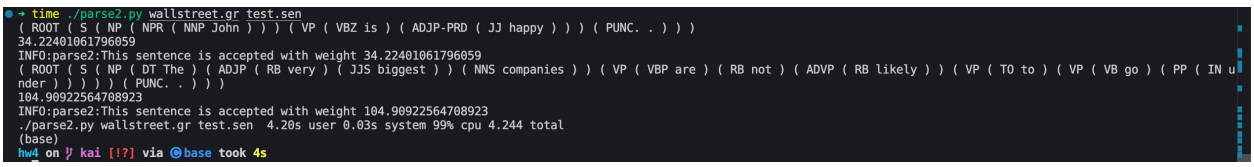

Figure 8: overall efficiency of our parser2

Figure 9: overall efficiency of our parser2