Parsing Techniques for Lexicalized Context-Free Grammars*

Summary

- Part I: Lexicalized Context-Free Grammars
  - motivations and definition
  - relation with other formalisms
- Part II: standard parsing
  - TD techniques
  - BU techniques
- Part III: novel algorithms
  - BU enhanced
  - TD enhanced

Lexicalized grammars

- each rule specialized for one or more lexical items
- advantages over non-lexicalized formalisms:
  - express syntactic preferences that are sensitive to lexical words
  - control word selection

Syntactic preferences

- adjuncts

  Workers [dumped sacks] into a bin

  *Workers dumped [sacks into a bin]

- N-N compound

  [hydrogen ion] exchange

  *hydrogen [ion exchange]

Word selection

- lexical

  Nora convened the meeting

  ?Nora convened the party

- semantics

  Peggy solved two puzzles

  ?Peggy solved two goats

- world knowledge

  Mary shelved some books

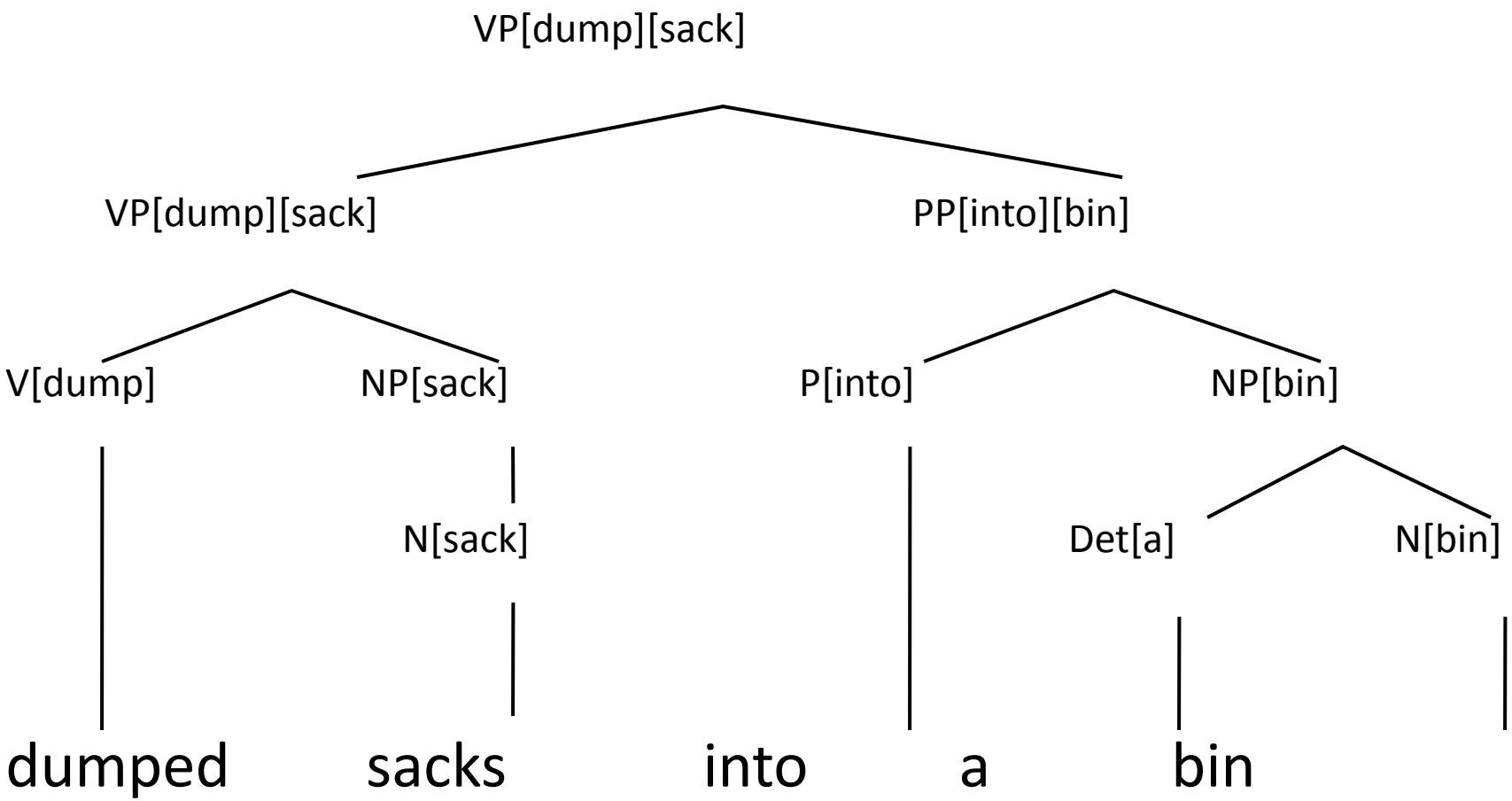
  ?Mary shelved some cooks

Lexicalized CFG

Motivations :

- study computational properties common to generative formalisms used in state-of-the-art real-world parsers
- develop parsing algorithm that can be directly applied to these formalisms

Lexicalized CFG



Lexicalized CFG

Context-free grammars with :

- alphabet V_T:
  - dumped, sacks, into, ...
- delexicalized nonterminals V_D:
  - NP, VP, ...
- nonterminals V_N:
  - NP[sack], VP[dump][sack], ...

Lexicalized CFG

Delexicalized nonterminals encode :

- word sense

  N, V, ...
- grammatical features

  number, tense, ...
- structural information

  bar level, subcategorization state, ...
- other constraints

  distribution, contextual features, ...

Lexicalized CFG

- productions have two forms :
  - V[dump] → dumped
  - VP[dump][sack]
    → VP[dump][sack] PP[into][bin]
- lexical elements in lhs inherited from rhs

Lexicalized CFG

- production is k-lexical :
  k occurrences of lexical elements in rhs
  - NP[bin] → Det[a] N[bin]
    is 2-lexical
  - VP[dump][sack]
    → VP[dump][sack] PP[into][bin]
    is 4-lexical

LCFG at work

- 2-lexical CFG
  - Alshawi 1996 : Head Automata
  - Eisner 1996 : Dependency Grammars
  - Charniak 1997 : CFG
  - Collins 1997 : generative model

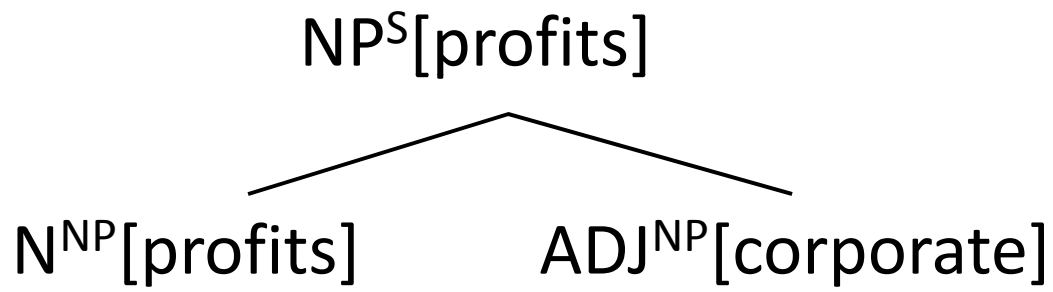
LCFG at work

Probabilistic LCFG *G* is strongly equivalent to probabilistic grammar *G'* iff

- 1-2-1 mapping between derivations
- each direction is a homomorphism
- derivation probabilities are preserved

LCFG at work

From Charniak 1997 to 2-lex CFG :



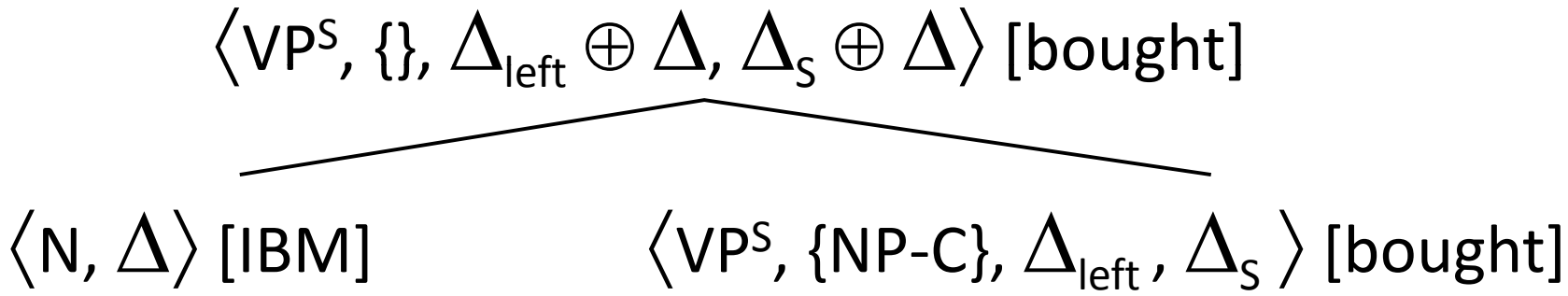
Pr_1 (corporate | ADJ, NP, profits)

Pr_1 (profits | N, NP, profits)

Pr_2 (NP $\rightarrow$ ADJ N | NP, S, profits)

LCFG at work

From Collins 1997 (Model #2) to 2-lex CFG :

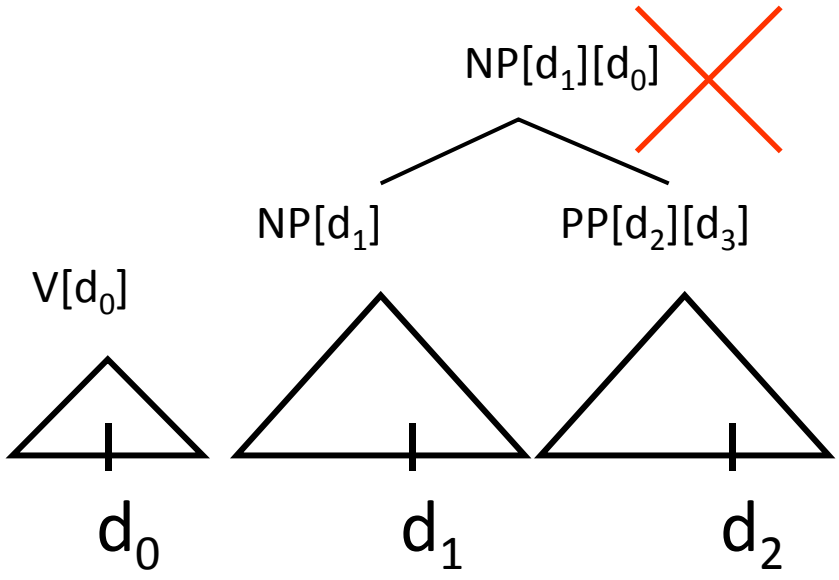


Pr_{left} (NP, IBM | VP, S, bought, Δ_{left} , {NP-C})

LCFG at work

Major Limitation :

Cannot capture relations involving lexical items outside actual constituent (cfr. history based models)



cannot look at d_0 when computing PP attachment

LCFG at work

- lexicalized context-free parsers that are not LCFG :
  - Magerman 1995 : Shift-Reduce+
  - Ratnaparkhi 1997 : Shift-Reduce+
  - Chelba & Jelinek 1998 : Shift-Reduce+
  - Hermjakob & Mooney 1997 : LR

Standard Parsing

- standard parsing algorithms (CKY, Earley, LC, …) run on LCFG in time $O(|G| \times |w|^3)$
- for 2-lex CFG (simplest case) $|G|$ grows with $|V_D|^3 \times |V_T|^2$!!

Goal :

Get rid of $|V_T|$ factors

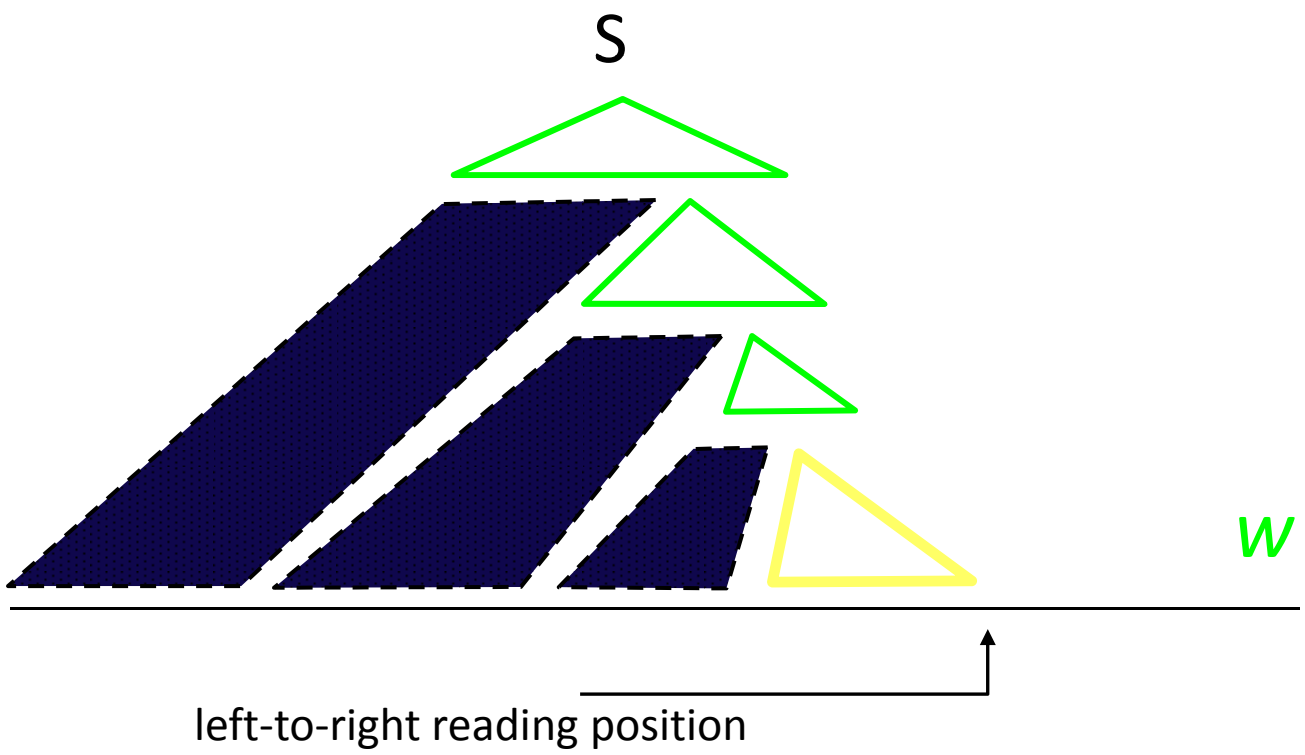
Standard Parsing: TD

Result (to be refined) :

Algorithms satisfying the correct-prefix property are “unlikely” to run on LCFG in time independent of $|V_T|$

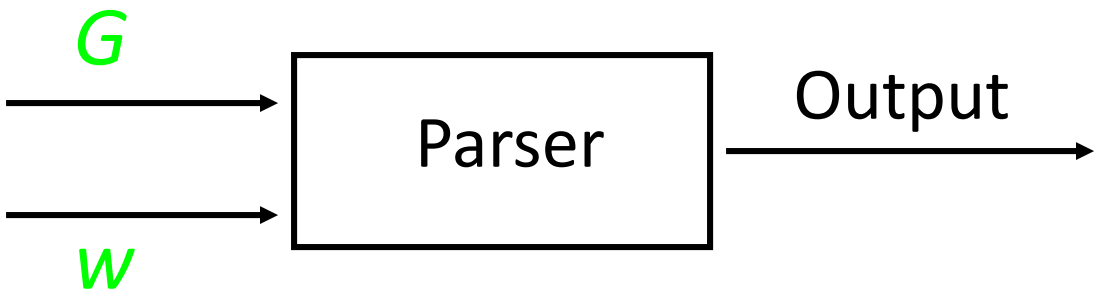
Correct-prefix property

Earley, Left-Corner, GLR, ... :



On-line parsing

No grammar precompilation (Earley) :



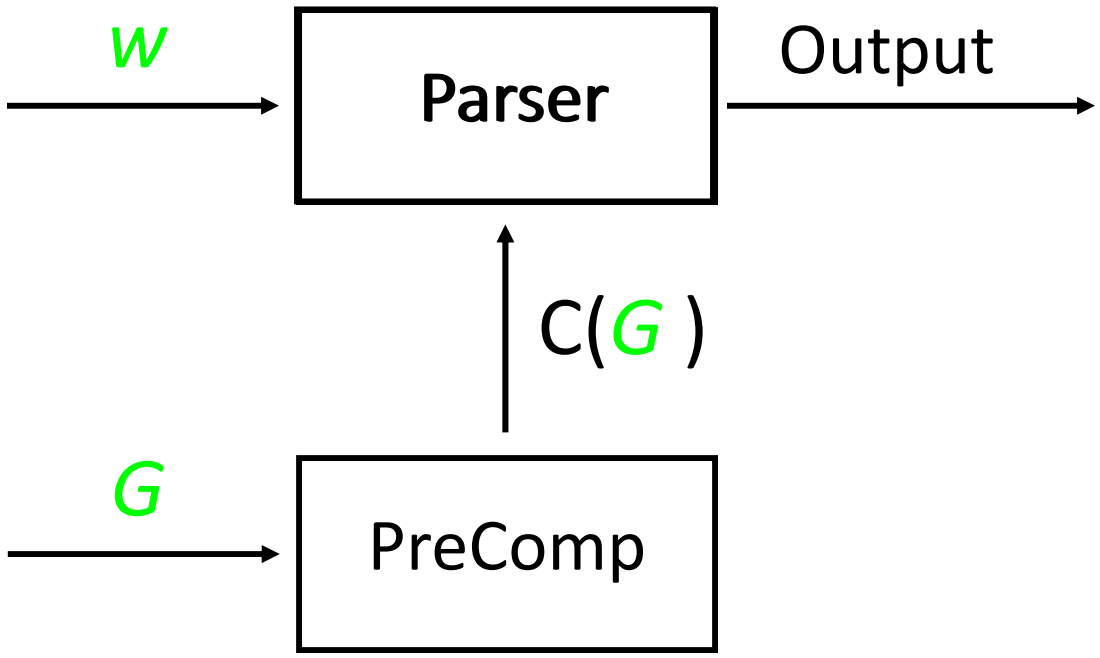
Standard Parsing: TD

Result :

On-line parsers with correct-prefix property cannot run in time O (f(| V_D |, | w |)), for any function f

Off-line parsing

Grammar is precompiled (Left-Corner, LR) :



Standard Parsing: TD

Fact :

We can simulate a nondeterministic FA M on w in time $O(|M| \times |w|)$

Conjecture :

Fix a polynomial p.

We cannot simulate M on w in time $p(|w|)$ unless we spend exponential time in precompiling M

Standard Parsing: TD

Assume our conjecture holds true

Result :

Off-line parsers with correct-prefix property cannot run in time O (p(| V_D |, | w |)), for any polynomial p, unless we spend exponential time in precompiling G

Standard Parsing: BU

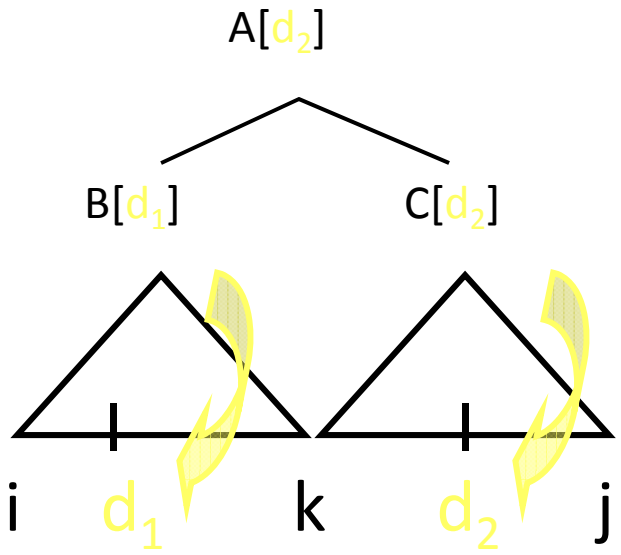
Common practice in lexicalized grammar parsing :

- select productions that are lexically grounded in w
- parse BU with selected subset of G

Problem :

Algorithm removes $|V_T|$ factors but introduces new $|w|$ factors !!

Standard Parsing: BU

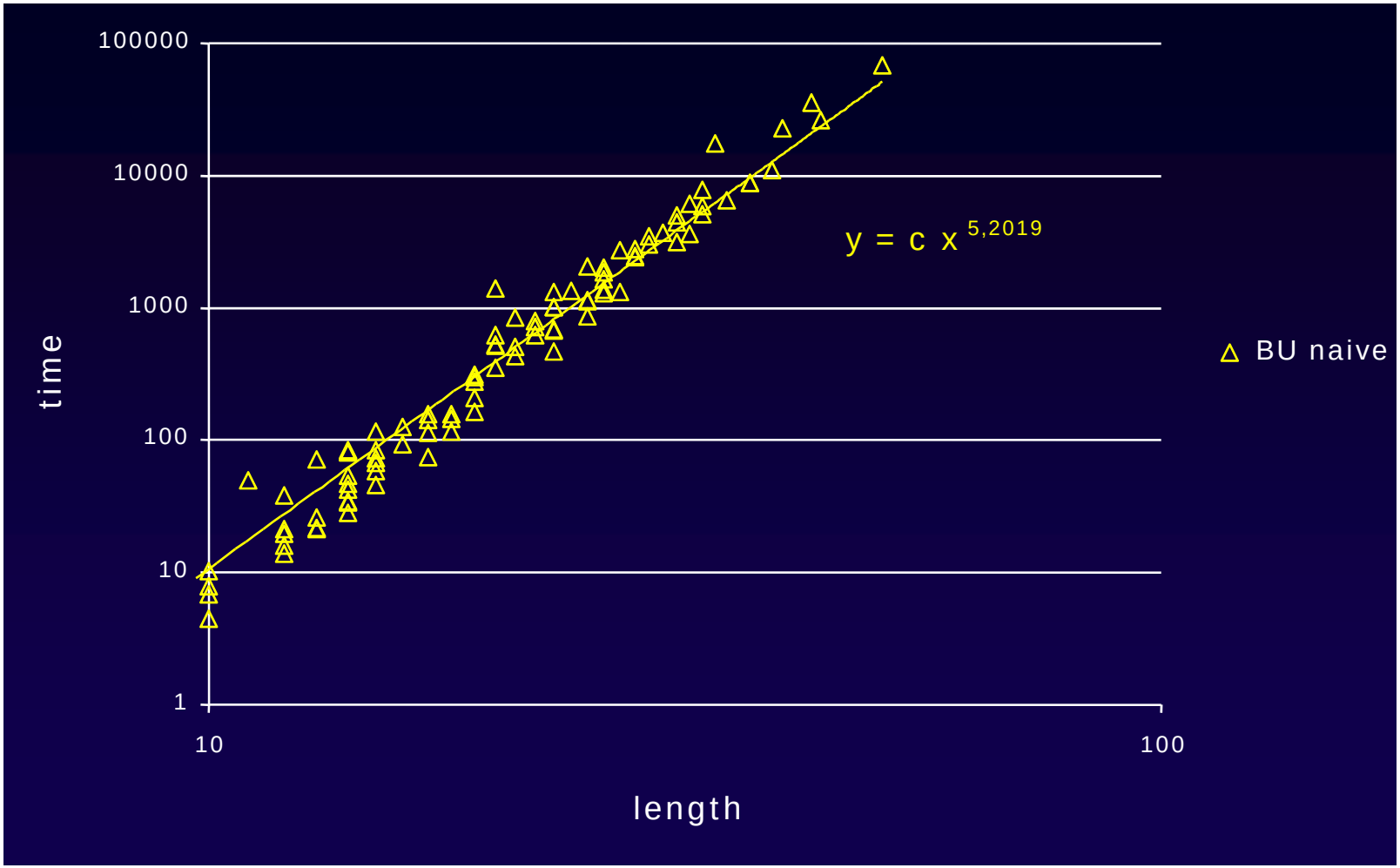


Time charged :

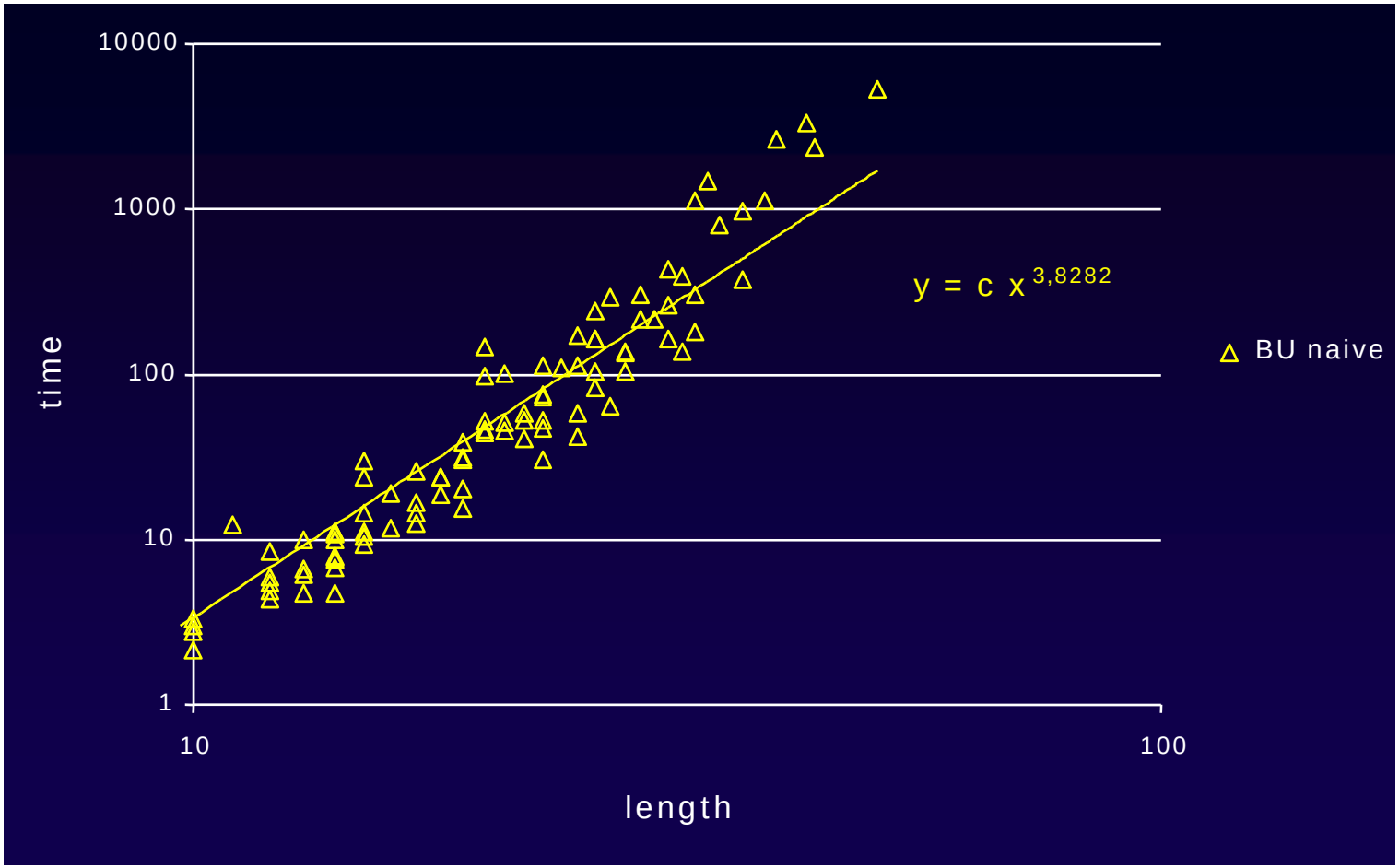
- i, k, j $\Rightarrow$ $|w|^3$
- A, B, C $\Rightarrow$ $|V_D|^3$
- d_1, d_2 $\Rightarrow$ $|w|^2$

Running time is $O(|V_D|^3 \times |w|^5)$!!

Standard BU : Exhaustive



Standard BU : Pruning



novel algorithms

BU enhanced

Result :

Parsing with 2-lex CFG in time

$O(|V_D|^3 \times |w|^4)$

Remark :

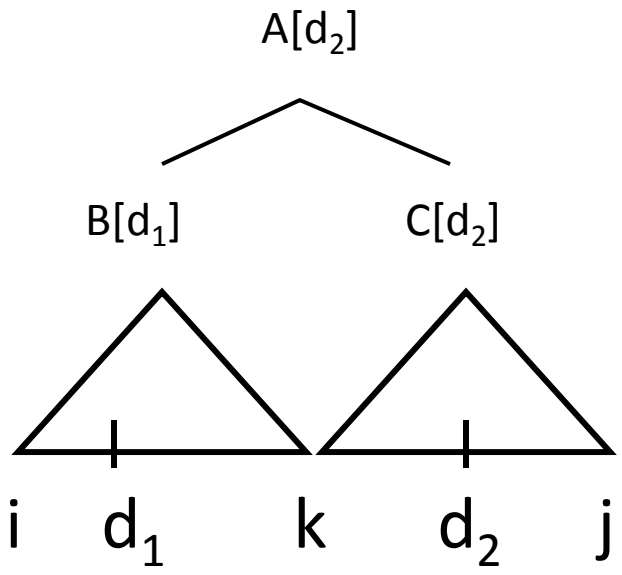
Result transfers to models in Alshawi 1996, Eisner 1996, Charniak 1997, Collins 1997

Remark :

Technique extends to improve parsing of Lexicalized-Tree Adjoining Grammars

Algorithm #1

Basic step in naive BU :

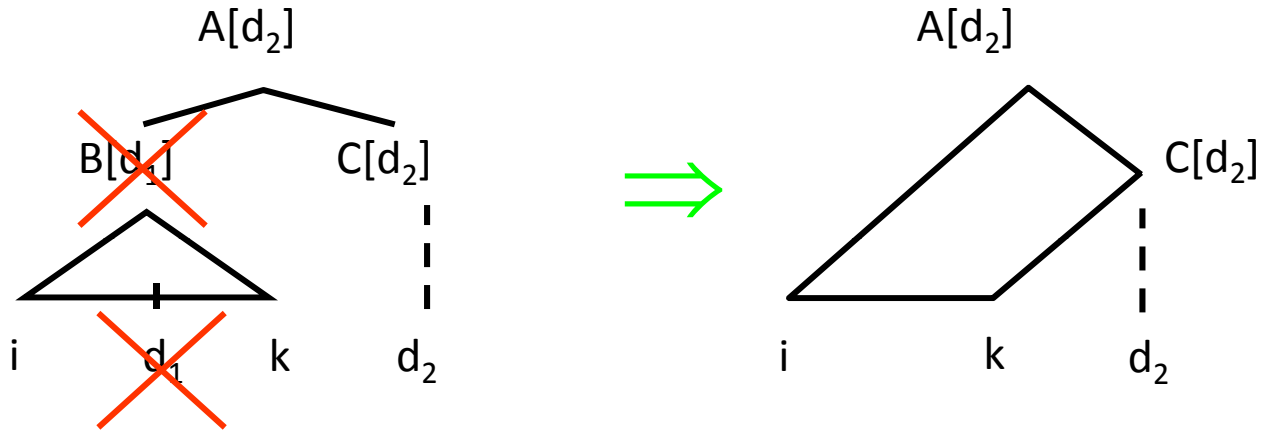


Idea :

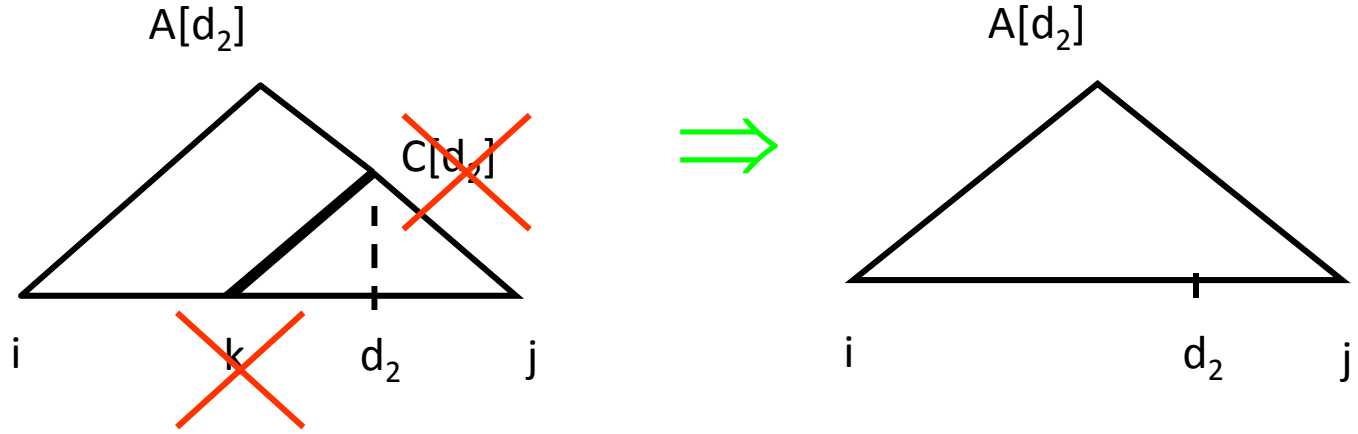
Indices d_1 and j can be processed independently

Algorithm #1

- Step 1



- Step 2



BU enhanced

Upper bound provided by Algorithm #1 :

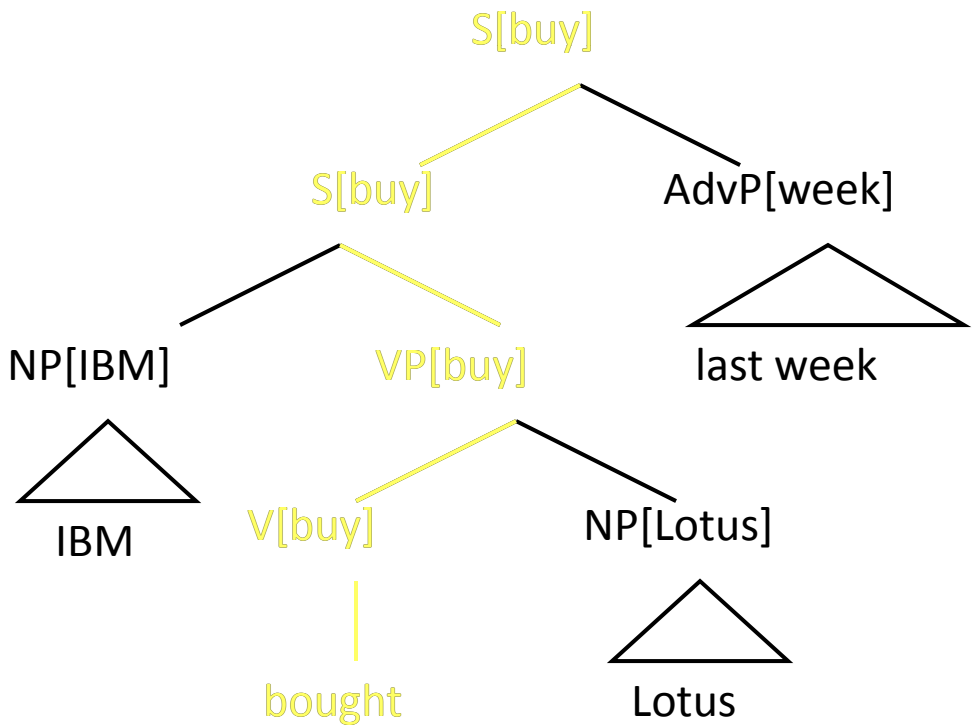
$O(|w|^4)$

Goal :

Can we go down to $O(|w|^3)$?

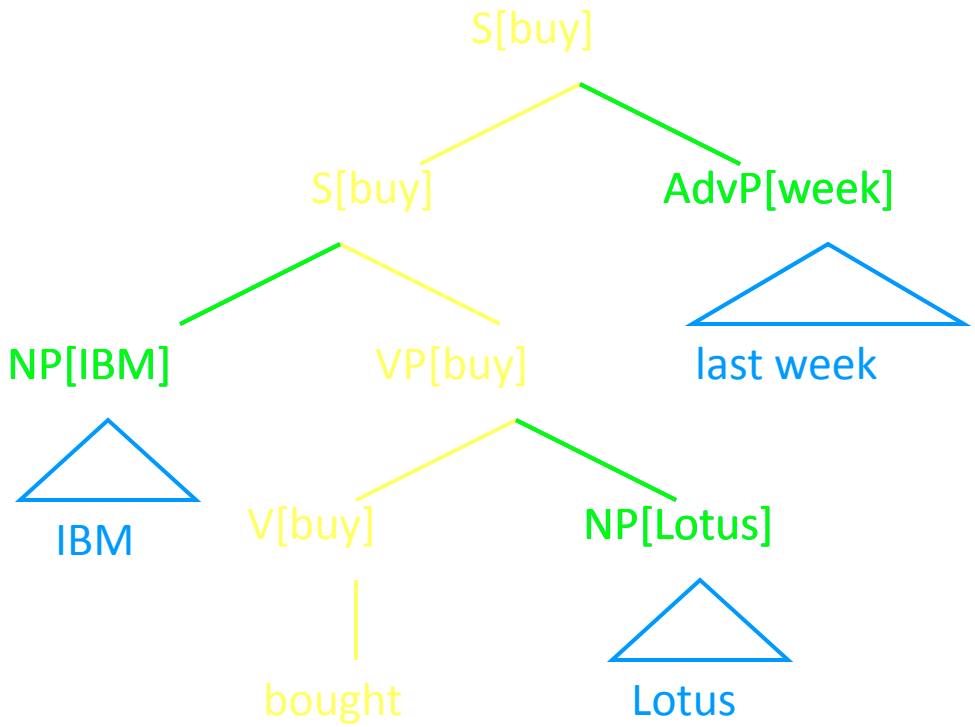
Spine

The spine of a parse tree is the path from the root to the root's head



Spine projection

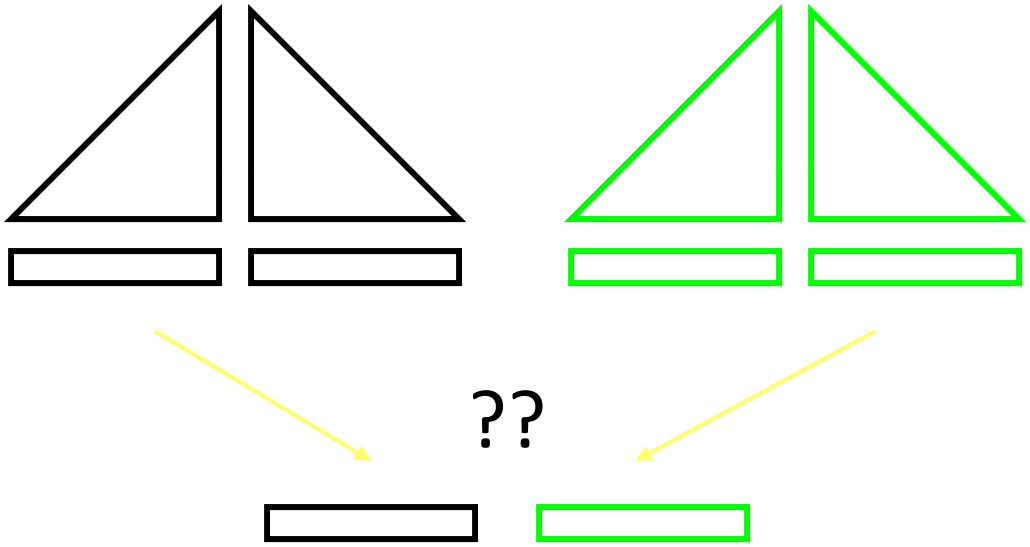
The spine projection is the yield of the sub-tree composed by the spine and all its sibling nodes



NP[IBM] bought NP[Lotus] AdvP[week]

Split Grammars

Split spine projections at head :



Problem :

how much information do we need to store in order to construct new grammatical spine projections from splits ?

Split Grammars

Fact :

Set of spine projections is a linear context-free language

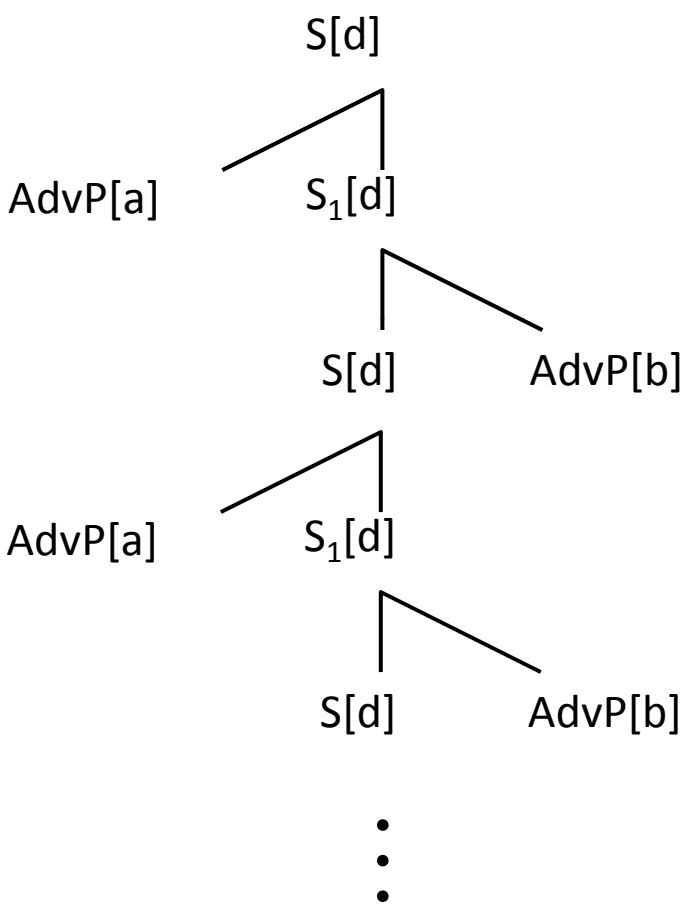
Definition :

2-lex CFG is split if set of spine projections is a regular language

Remark :

For split grammars, we can recombine splits using finite information

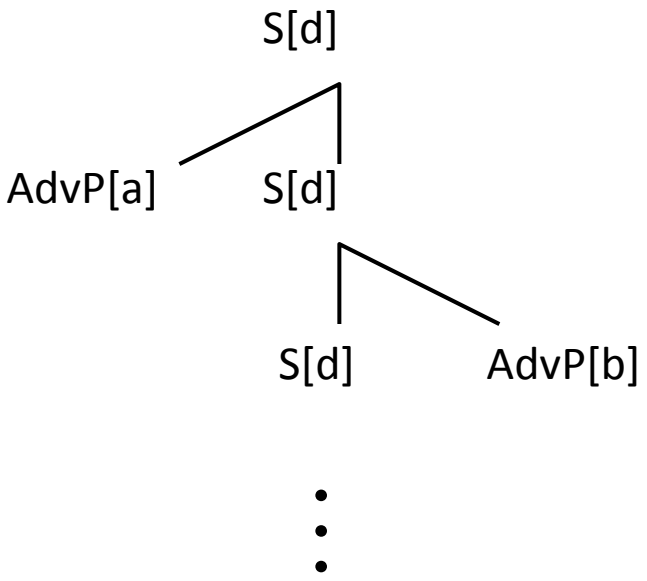
Split Grammars



Non-split grammar :

- unbounded # of dependencies between left and right dependents of head
- linguistically unattested and unlikely

Split Grammars

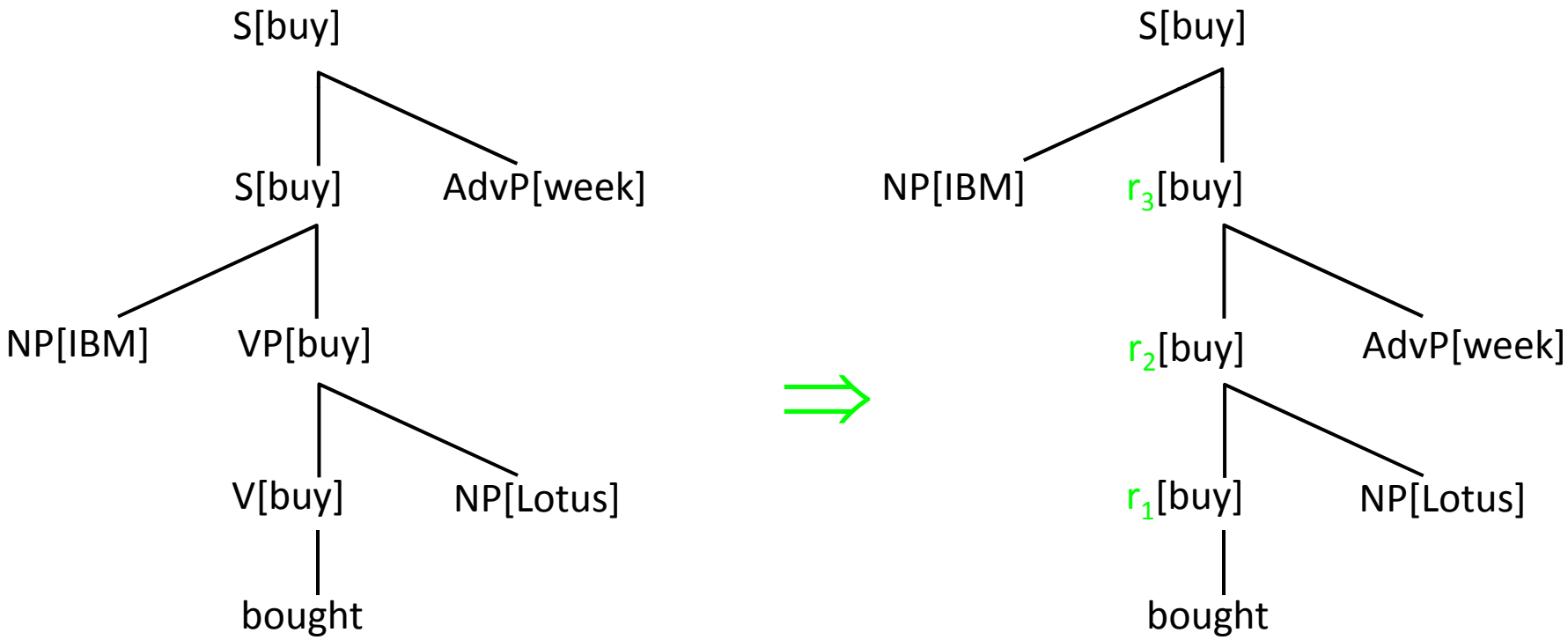


Split grammar :

finite # of dependencies between left and right dependents of lexical head

Split Grammars

Precompile grammar such that splits are derived separately



r_3[buy] is a split symbol

Split Grammars

- t : max # of states per spine automaton
- g : max # of split symbols per spine automaton ($g < t$)
- m : # of delexicalized nonterminals thare are maximal projections

BU enhanced

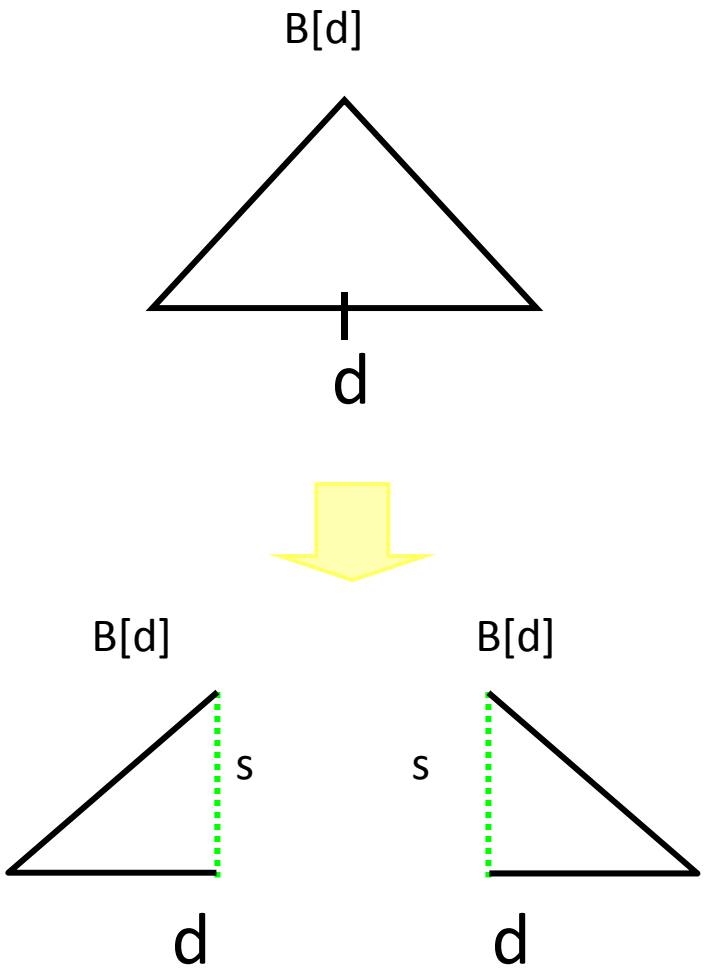
Result :

Parsing with split 2-lexical CFG in time $O(t^2 g^2 m^2 |w|^3)$

Remark:

Models in Alshawi 1996, Charniak 1997 and Collins 1997 are not split

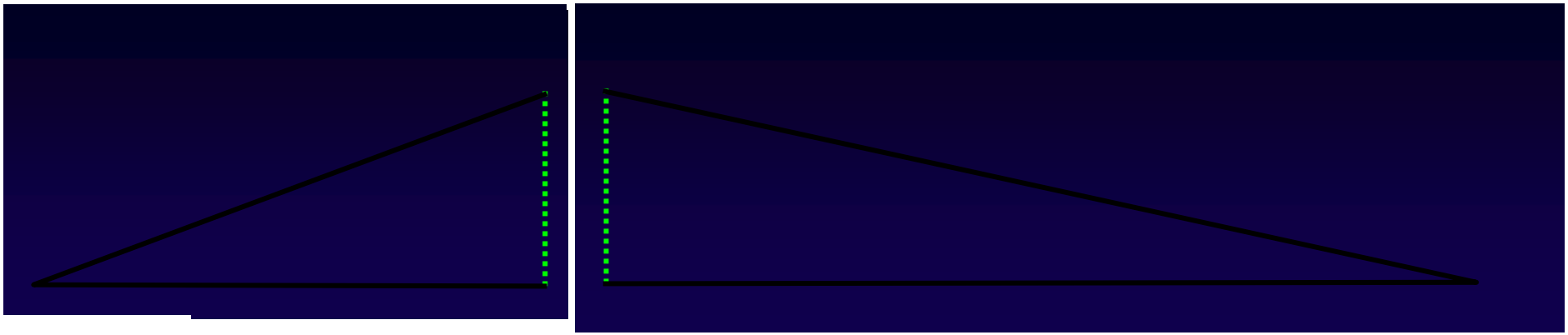
Algorithm #2



Idea :

- recognize left and right splits separately
- collect head dependents one split at a time

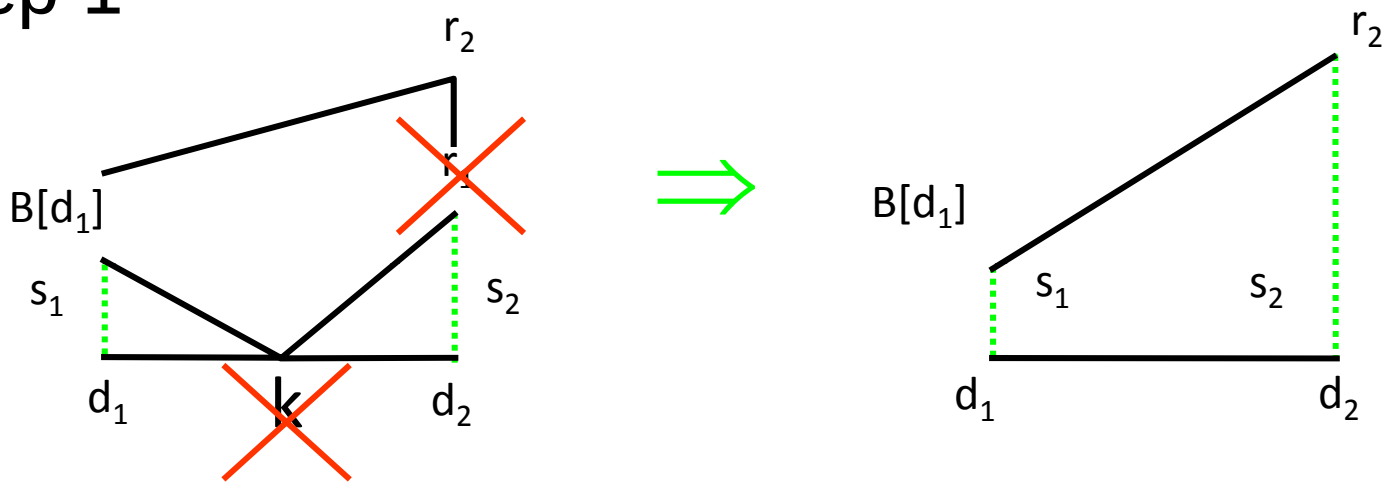
Algorithm #2



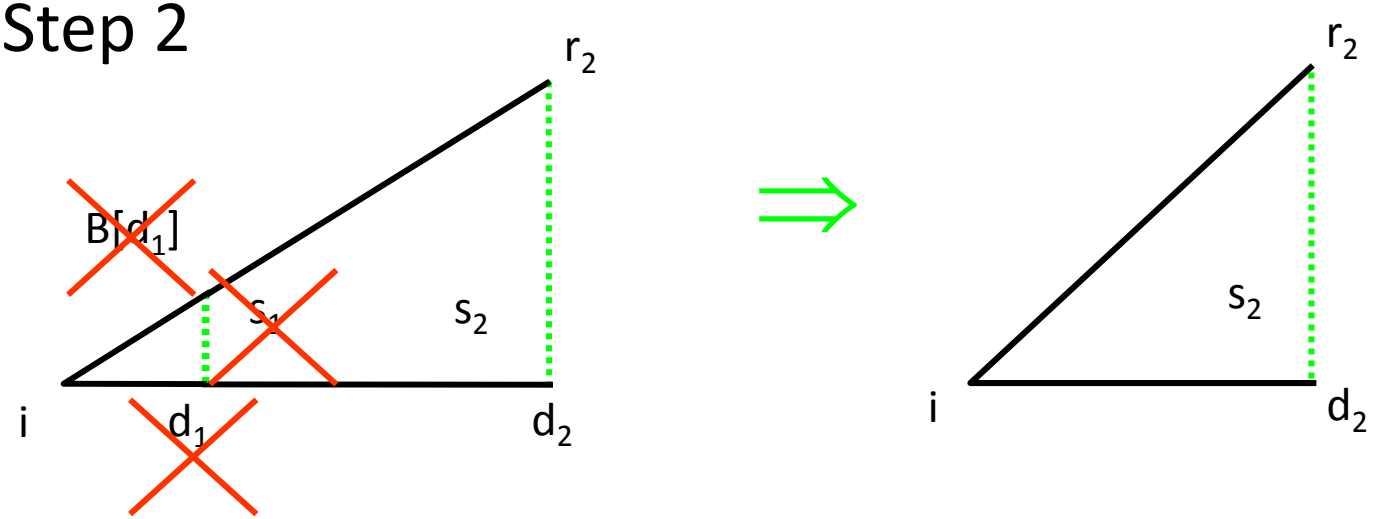
NP[IBM] bought NP[Lotus] AdvP[week]

Algorithm #2

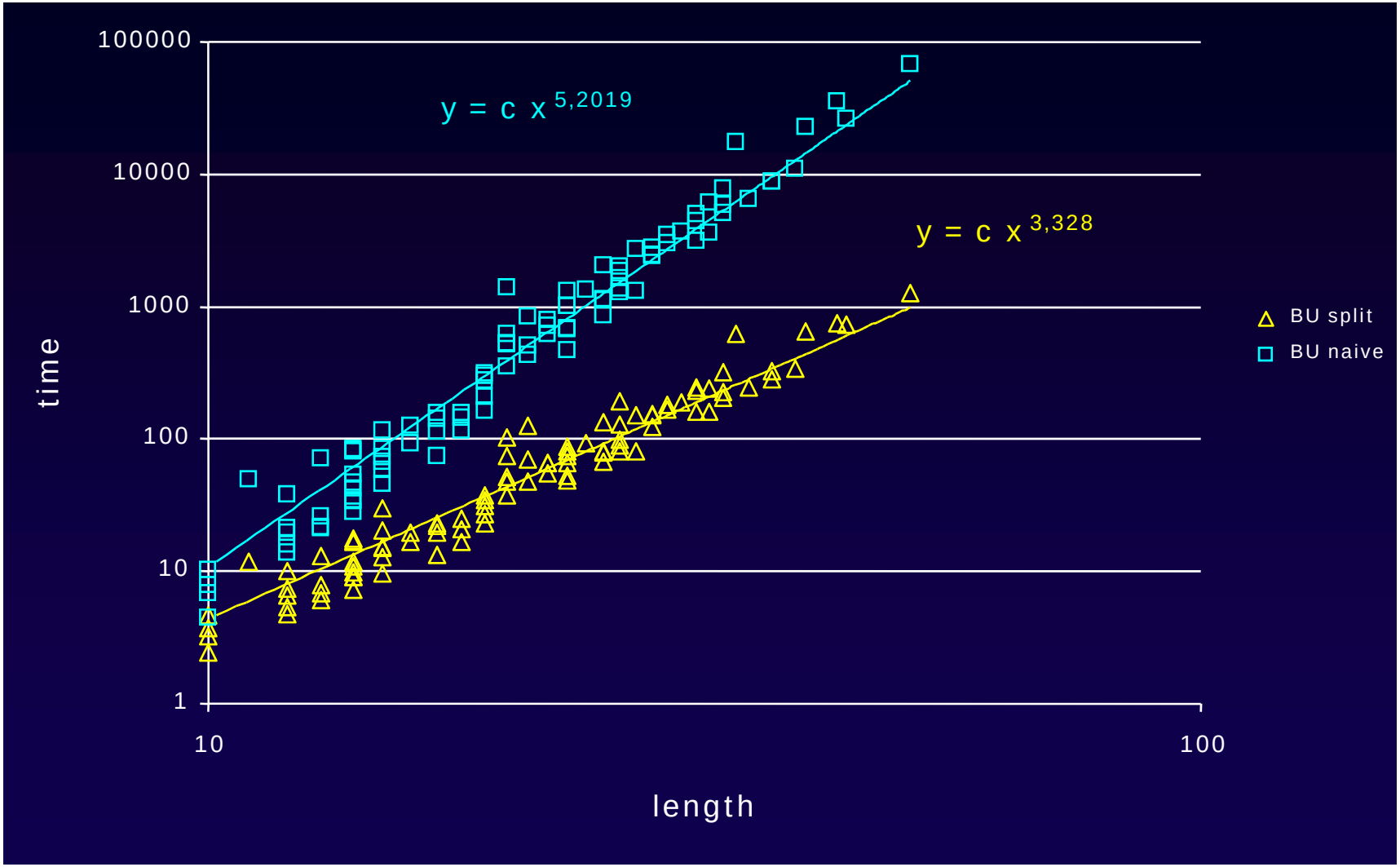
- Step 1



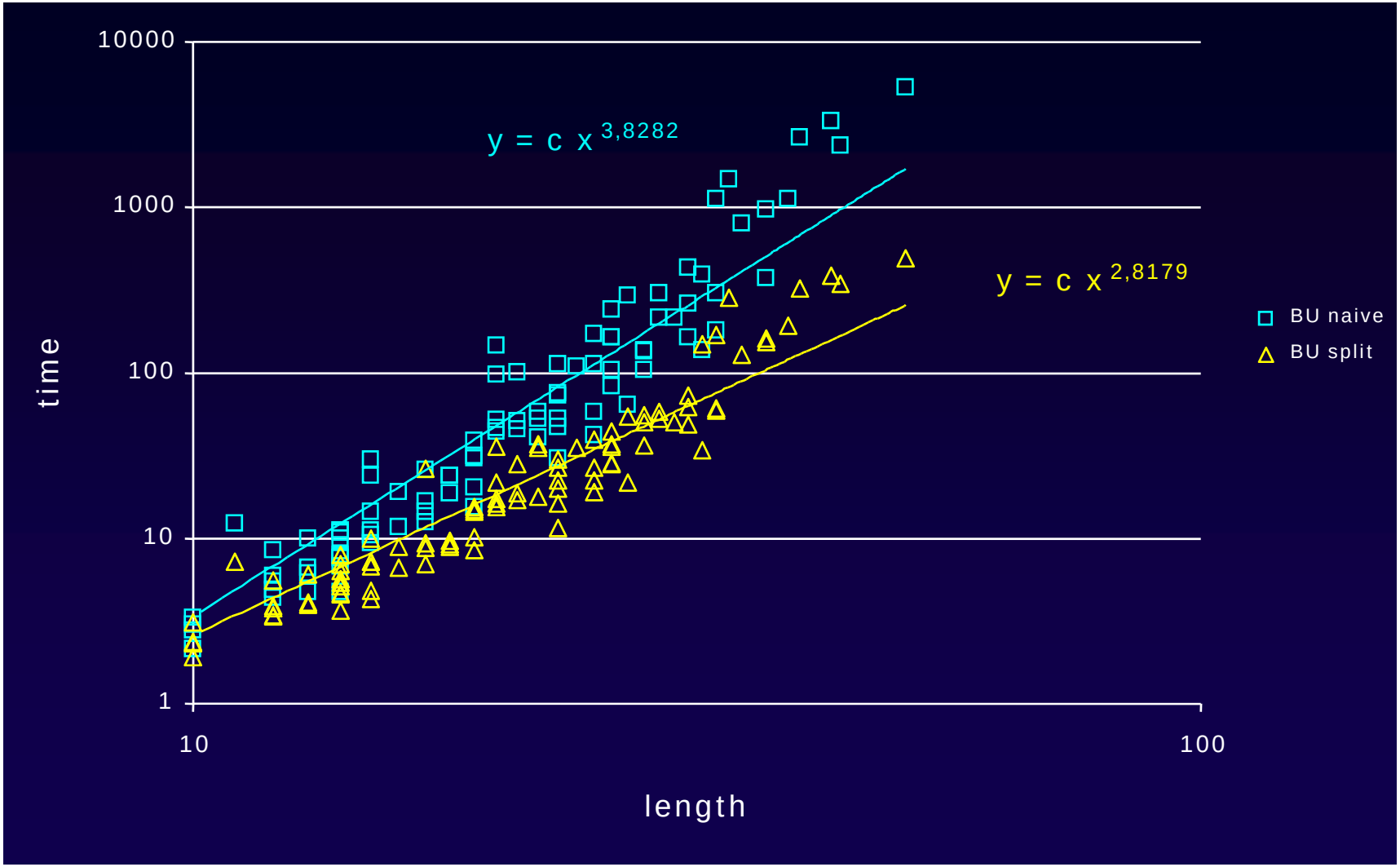
- Step 2



Algorithm #2 : Exhaustive



Algorithm #2 : Pruning



TD enhanced

Goal :

Introduce TD prediction for 2-lexical CFG parsing, without $|V_T|$ factors

Remark :

Must relax left-to-right parsing (because of previous results)

TD enhanced

Result :

TD parsing with 2-lex CFG in time

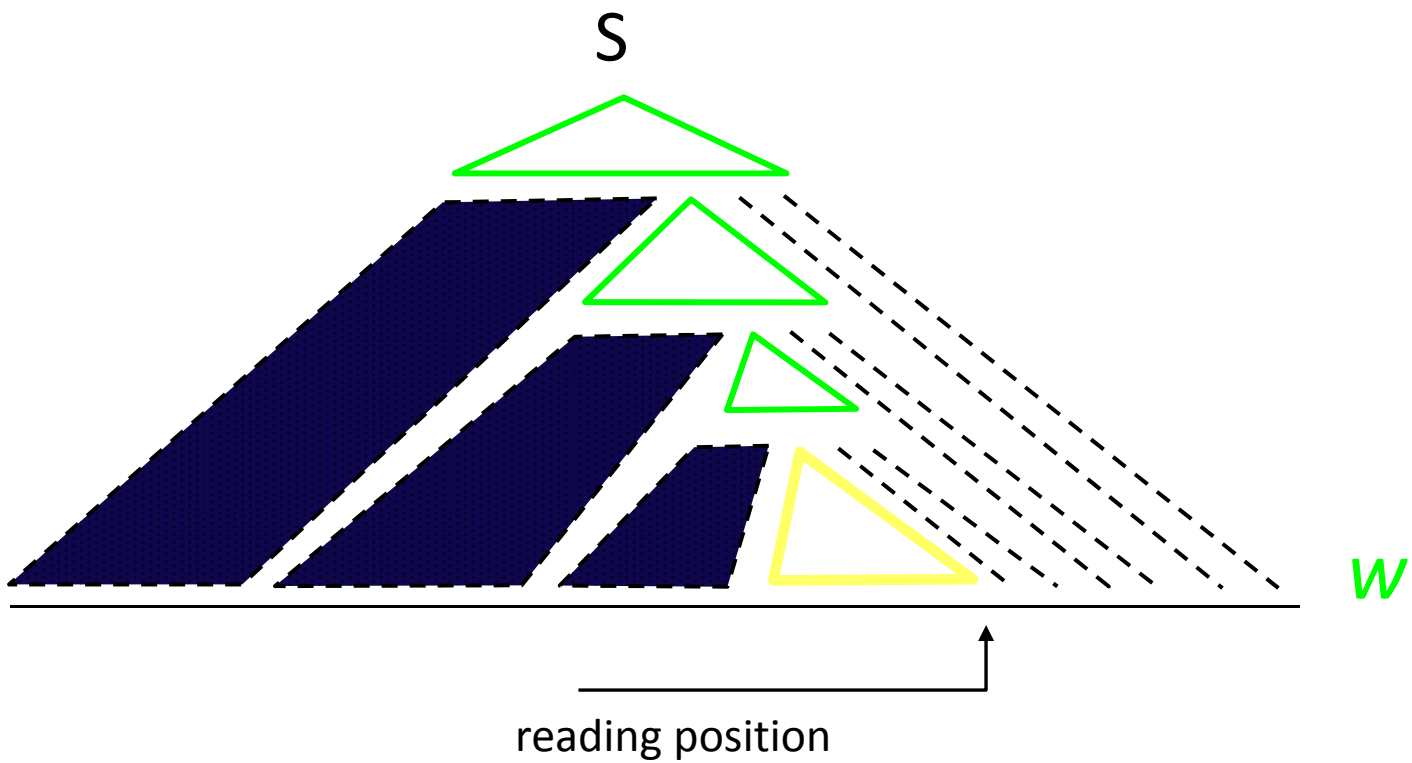
$O(|V_D|^3 \times |w|^4)$

Open :

$O(|w|^3)$ extension to split grammars

TD enhanced

Strongest version of correct-prefix property :

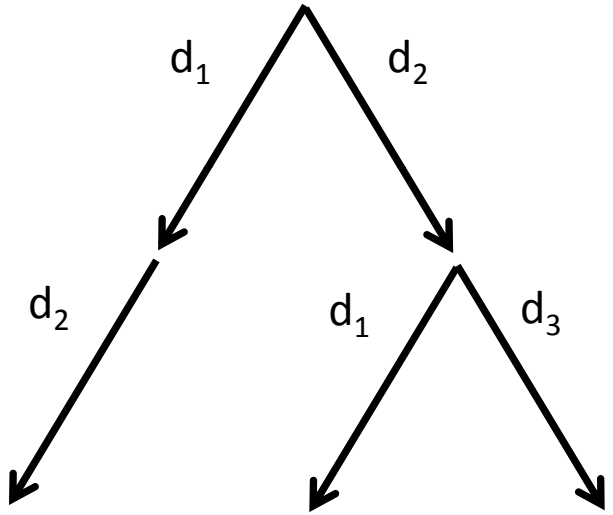


Data Structures

Prods with lhs A[d] :

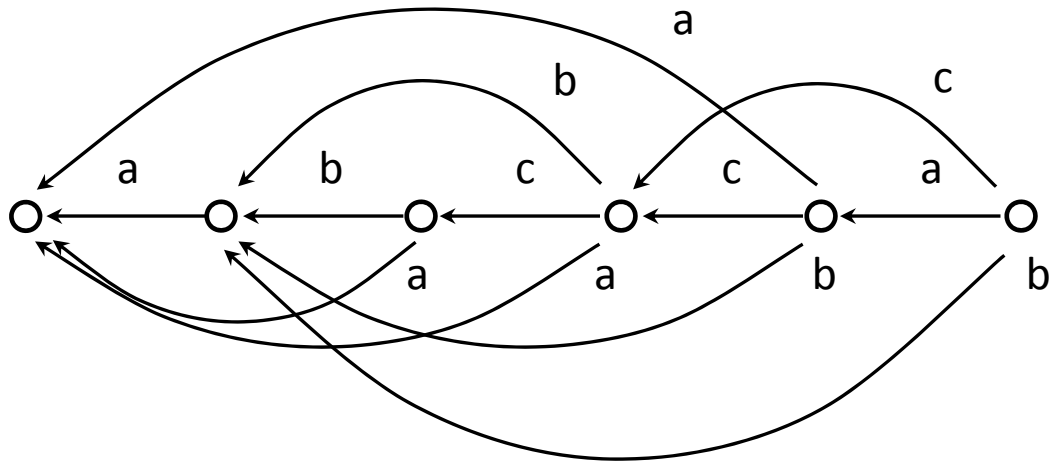
- $A[d] \rightarrow X_1[d_1]\ X_2[d_2]$
- $A[d] \rightarrow Y_1[d_3]\ Y_2[d_2]$
- $A[d] \rightarrow Z_1[d_2]\ Z_2[d_1]$

Trie for A[d] :

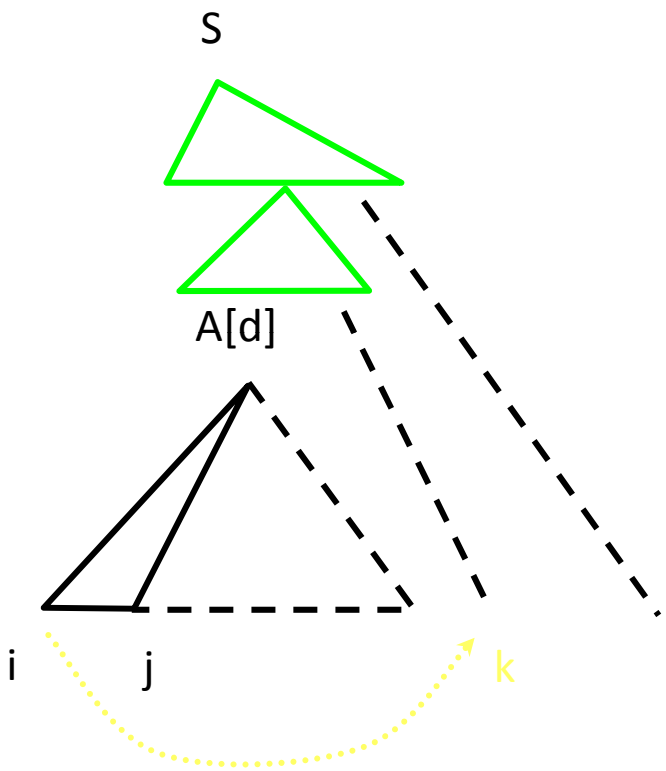


Data Structures

Rightmost subsequence recognition by precompiling input w into a deterministic FA :



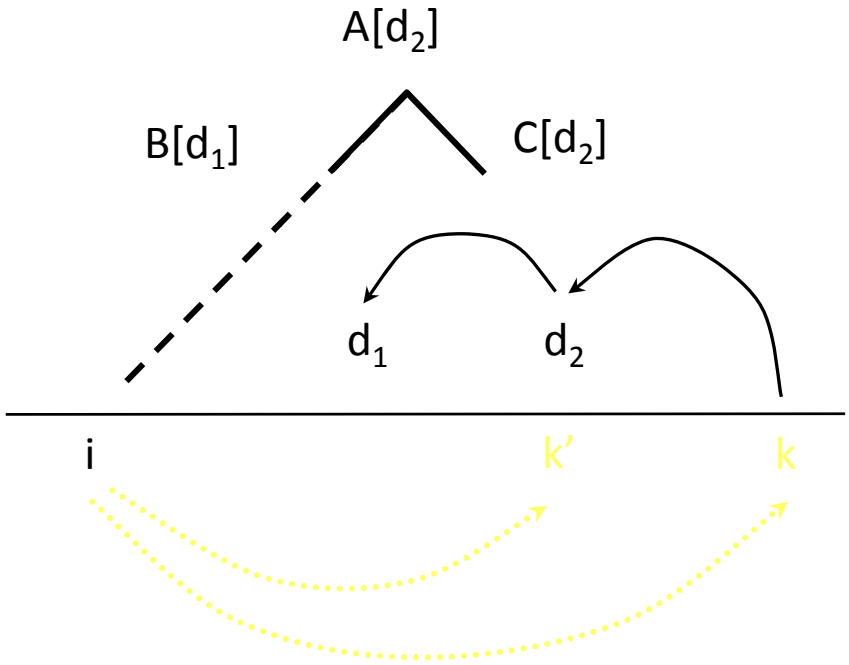
Algorithm #3



Item representation :

- i, j indicate extension of A[d] partial analysis
- k indicates rightmost possible position for completion of A[d] analysis

Algorithm #3 : Prediction



- Step 1 :
  find rightmost subsequence before k for some $A[d_2]$ production
- Step 2 :
  make Earley prediction

Conclusions

- standard parsing techniques are not suitable for processing lexicalized grammars
- novel algorithms have been introduced using enhanced dynamic programming
- work to be done :
  extension to history-based models