Incremental Input Stream Segmentation for Real-time NLP Applications

Mahsa Yarmohammadi

Streaming NLP for Big Data Class
SBU – Computer Science Department

9/29/2016

Outline

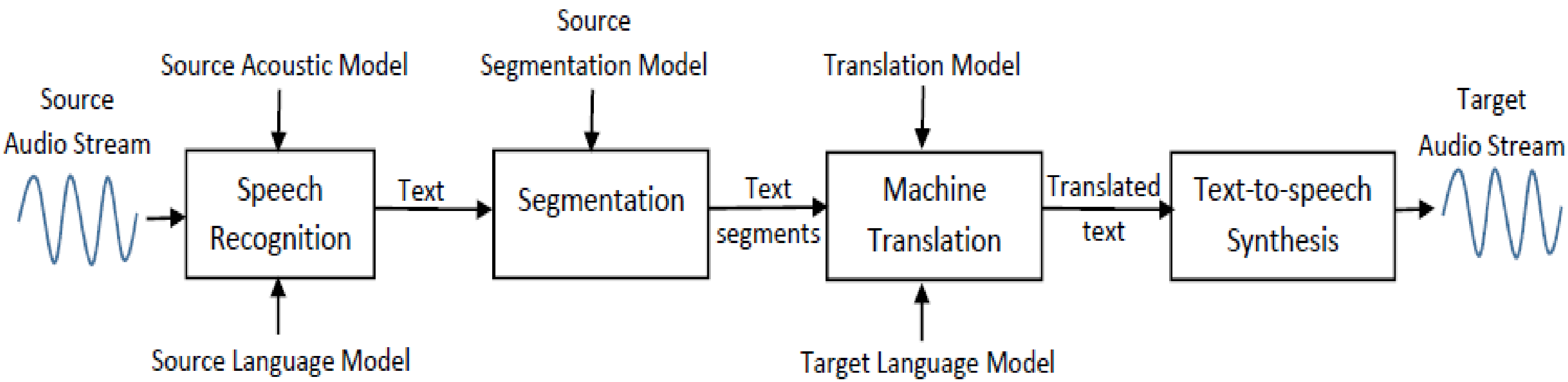
- Introduction
  - Simultaneous speech-to-speech translation (SST) as an example of a real-time NLP application
- Current input stream segmentation strategies
- Incorporating syntax into input stream segmentation
- A novel partial parsing approach
  - Hedge parsing
- Impact of hedge parsing in MT and SST

Introduction

- Simultaneous speech-to-speech translation (SST):
  Listening to source language speech, and *at the same time*, producing target language speech.

- Challenges of SST, and real-time systems:
  - no later revisions of mistakes
  - little latency in delivering the output after receiving the input
  - process parts of the input, even before it has been completed
  - s*egment* continuous stream input data to the appropriate units

SST Pipeline



Input Segmentation

- SST requires segments of the stream input that:
  - are separated at appropriate positions
  - are non-overlapping
  - could be processed sequentially

- Granularity of segments impacts translation latency/acc.
  - shorter segments are typically delivered more quickly
  - shorter segments are typically processed more quickly
  - shorter segments will likely result in inferior translation accuracy

Input Segmentation

- Previous work on SST mainly focused on
  - Pauses in the speech
  - The location of comma or period in the transcribed text
  - Combined punctuation-based and length-based methods
  - Joint segmentation and translation optimization

Input Segmentation

- Fügen et al. (2007)
  - Baseline: sentence boundaries
    - 36.6% BLEU score by translating ASR reference transcripts, 33.4% by translating ASR hypotheses
    - avg sentence length: 30 words
  - Automatically predicted punctuation marks
    - similar BLEU scores as above, avg segment length: 9 words
  - Every n words
    - n=7, 30.1% BLEU for ASR reference, 27.5% BLEU for ASR hypothesis
    - can destroy semantic context
  - Non-speech duration of 0.3 seconds
    - 32.6% BLEU score for ASR hypotheses
    - + lexical features 32.9% BLEU score, avg segment length: 9 words

Input Segmentation

- Rangarajan Sridhar et al. (2013)
  - Non-linguistic and linguistic segmentation strategies

  - Every n words
    - larger n values: good translation accuracy, but high latency
  - Optimal word alignment occurs only within segments
    - poor translation due to short segments (2-4 words)

  - Sentences, or comma-separated segments
    - automatically predicted by an SVM classifier
    - performs the best, but the classifier introduces a significant delay
  - Four segment types of noun, verb, particle, and adverbial
    - poor translation, mainly due to short segments

Input Segmentation

- Matusov et al. (2007)
  - automatic sentence boundary and sub-sentence punctuation prediction
  - the best translation achieved when boundary detection algorithms were directly optimized for translation quality

- Cettolo and Federico (2006)
  - punctuation-based, length-based, and combined text segmentation criteria
  - the best performance achieved by combining both linguistic and input length constraints

Syntax-based Segmentation and Annotation

- Human interpreters depend on info. of a *structural nature*
  - the input segmentation follows mainly *syntactical* principles

- Syntactic *annotations* in the input segments could potentially improve the performance of SST

  - syntactic annotations can be helpful in regular (non-incremental) translation

    (Mi et al., 2008;Liu et al., 2011;Zhang et al., 2011;Tamura et al., 2013)

Incremental Syntactic Analysis

- Applying syntactic info in real-time scenarios is challenging

- Conventional full syntactic parsing:
  - is not directly applicable to sub-sentential segments
  - builds fully connected structures over the *entire string*
  - is generally computationally expensive

- A fast partial syntactic parsing of the input should be considered

A Novel Partial Parsing Approach

Propose a novel partial parsing method for fast and incremental syntactic analysis of the input that:

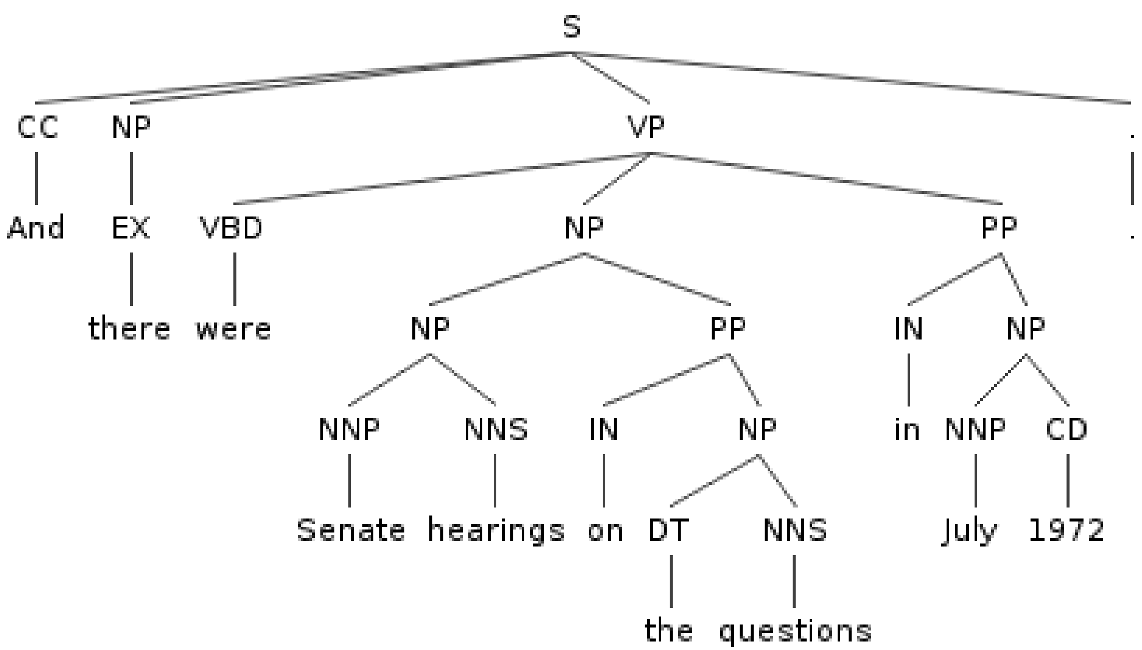
1) less computationally demanding than a full parser but more effective than a shallow parser

2) allows for syntax-based segmentation, and

3) incorporates some degree of syntax without requiring the entire sentence

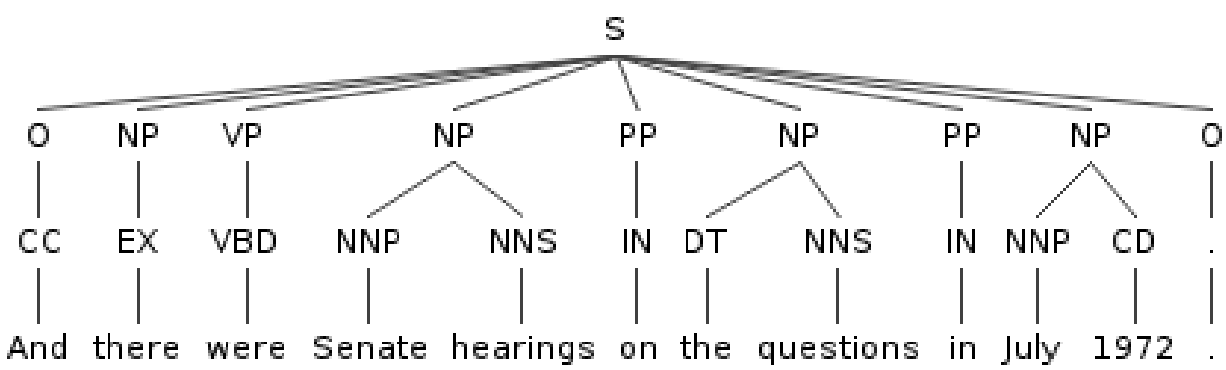
Full Syntactic Parsing

- *Full parsing* gives a complex complete parse tree of the sentence
  - hierarchically embedded structures, recursive phrase construction
  - great expressive power but computationally expensive



Partial Syntactic Parsing

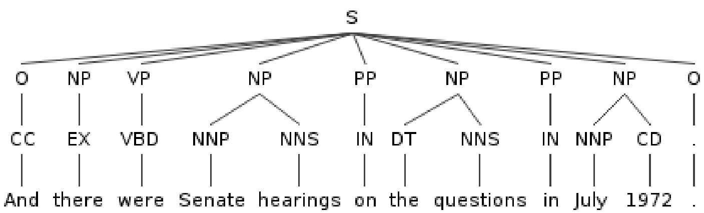
- *Shallow parsing (chunking)* identifies flat, non-overlapping constituents
  - the chunks lack hierarchical structures
  - very fast and efficient, but not powerful enough to define recursive phrases

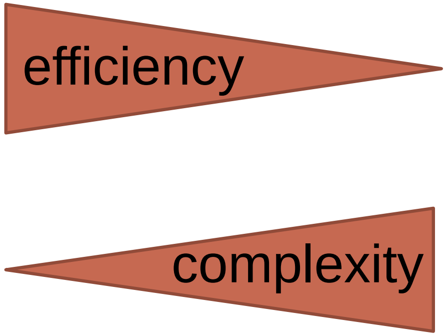


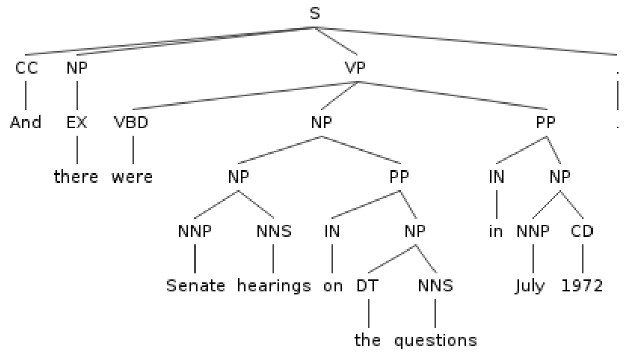
Syntactic Parsing

Shallow parsing

Full parsing







flat bracketing structures

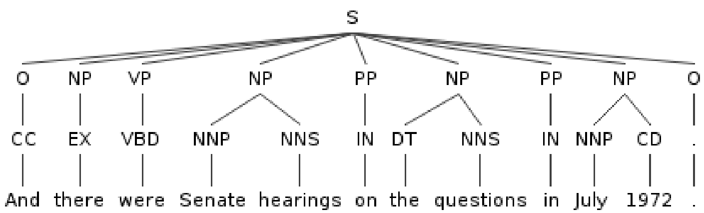
fully recursive structures

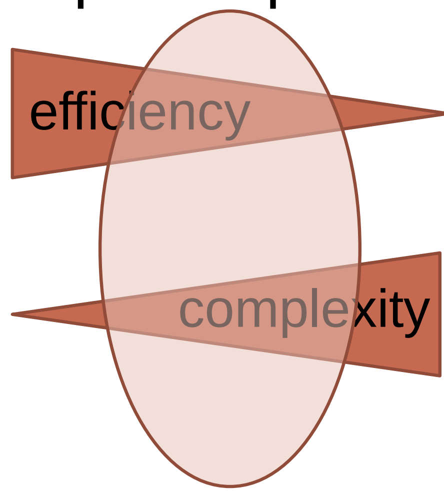
Syntactic Parsing

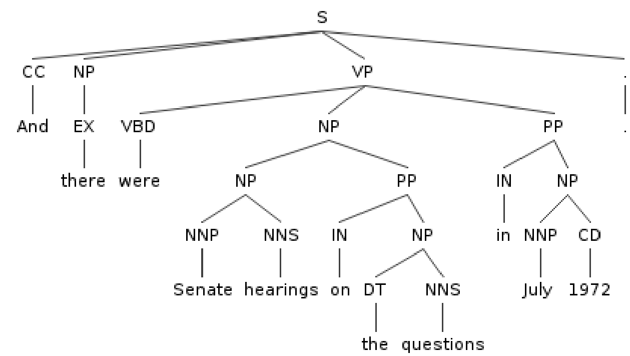
Shallow parsing

Some partial parsing?

Full parsing







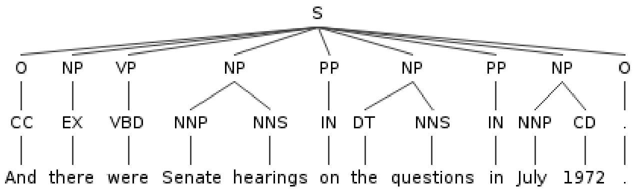
flat bracketing structures

fully recursive structures

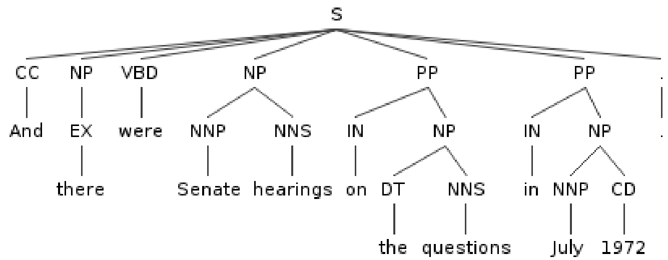
portions of recursive structures

Syntactic Parsing

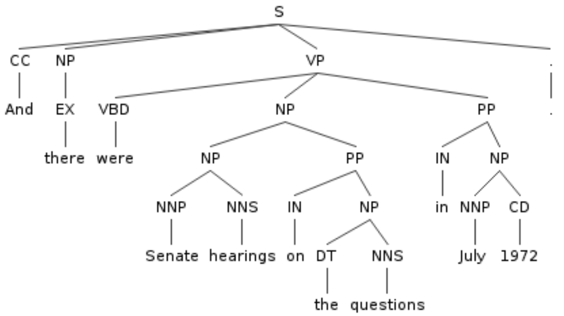
Shallow parsing



Hedge parsing



Full parsing



flat bracketing structures

fully recursive structures

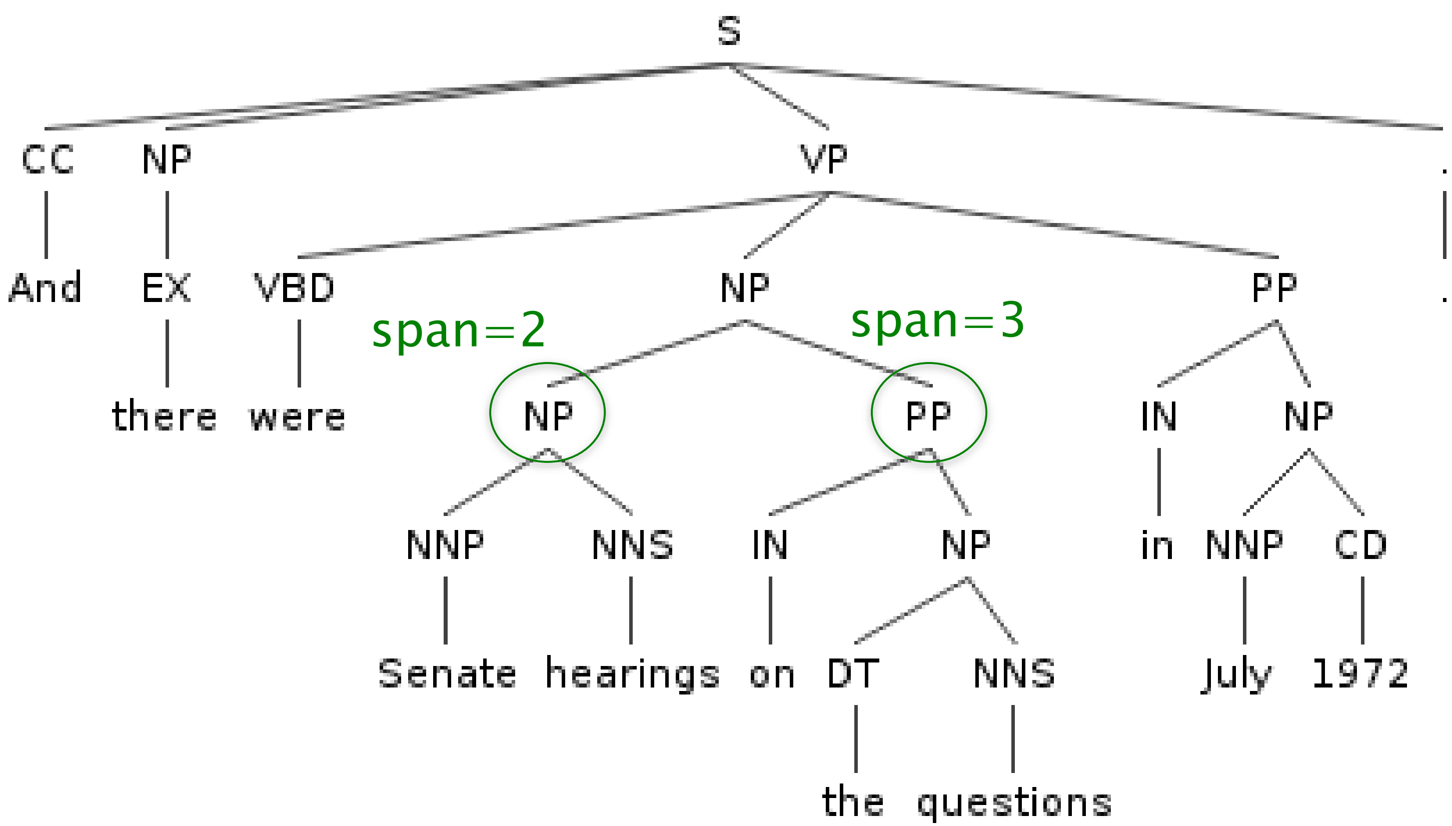
fully recursive structures for
constituents covering < L words

Hedge Transform

- Preserving every constituent of length up to some span *L*

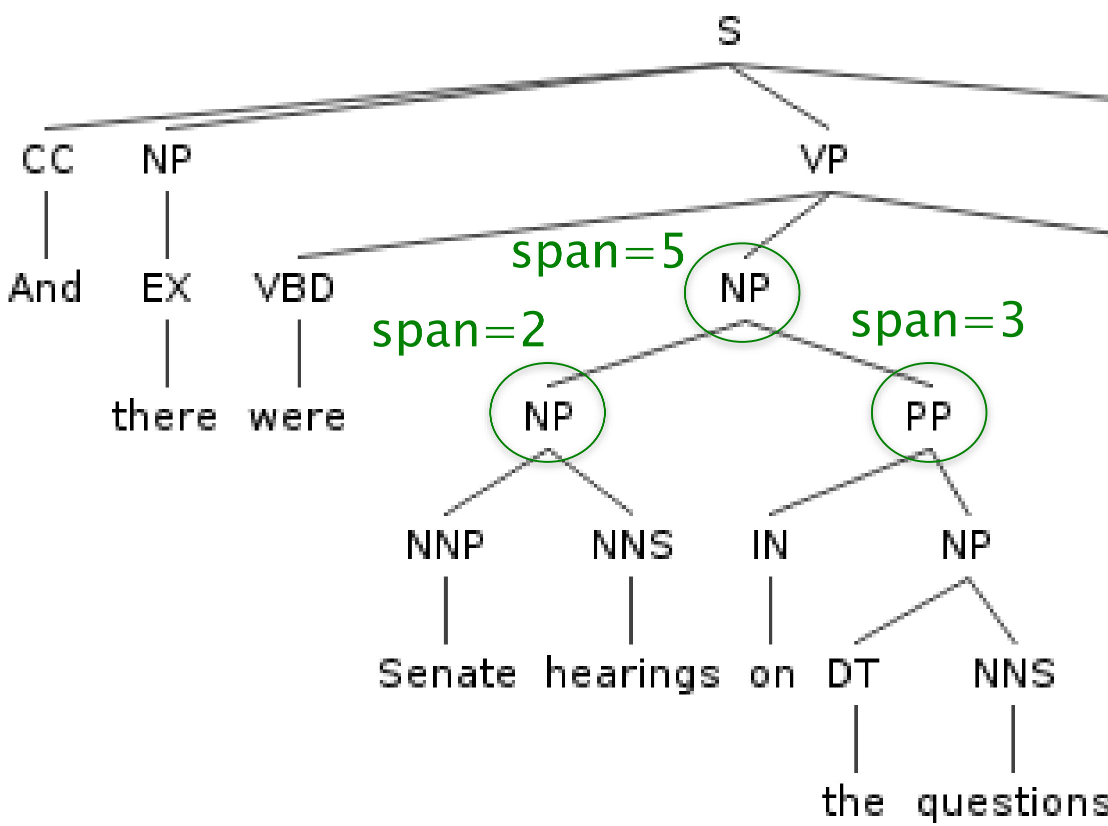
Hedge Transform

- Preserving every constituent of length up to some span *L*



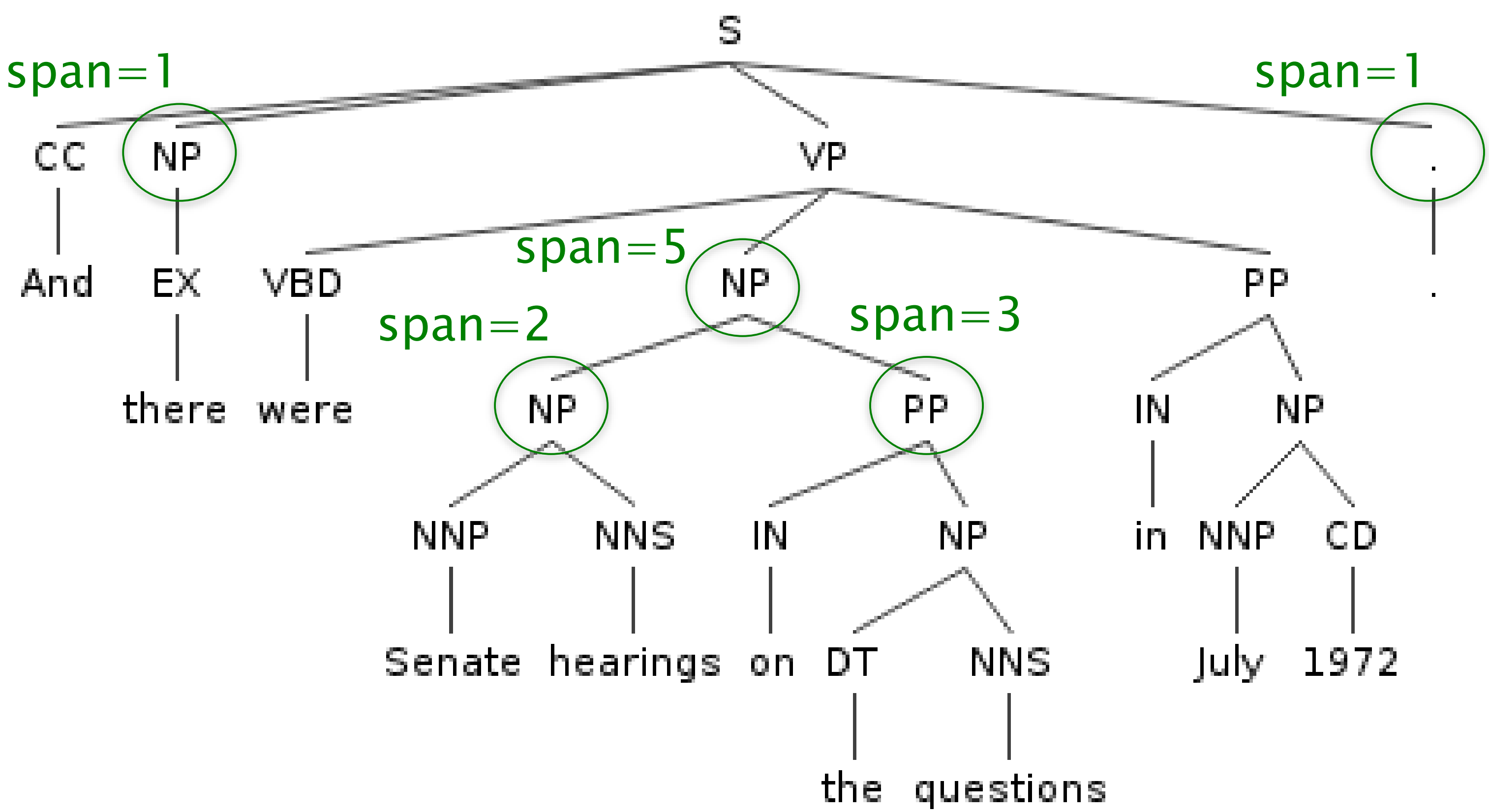
Hedge Transform

- Preserving every constituent of length up to some span *L*



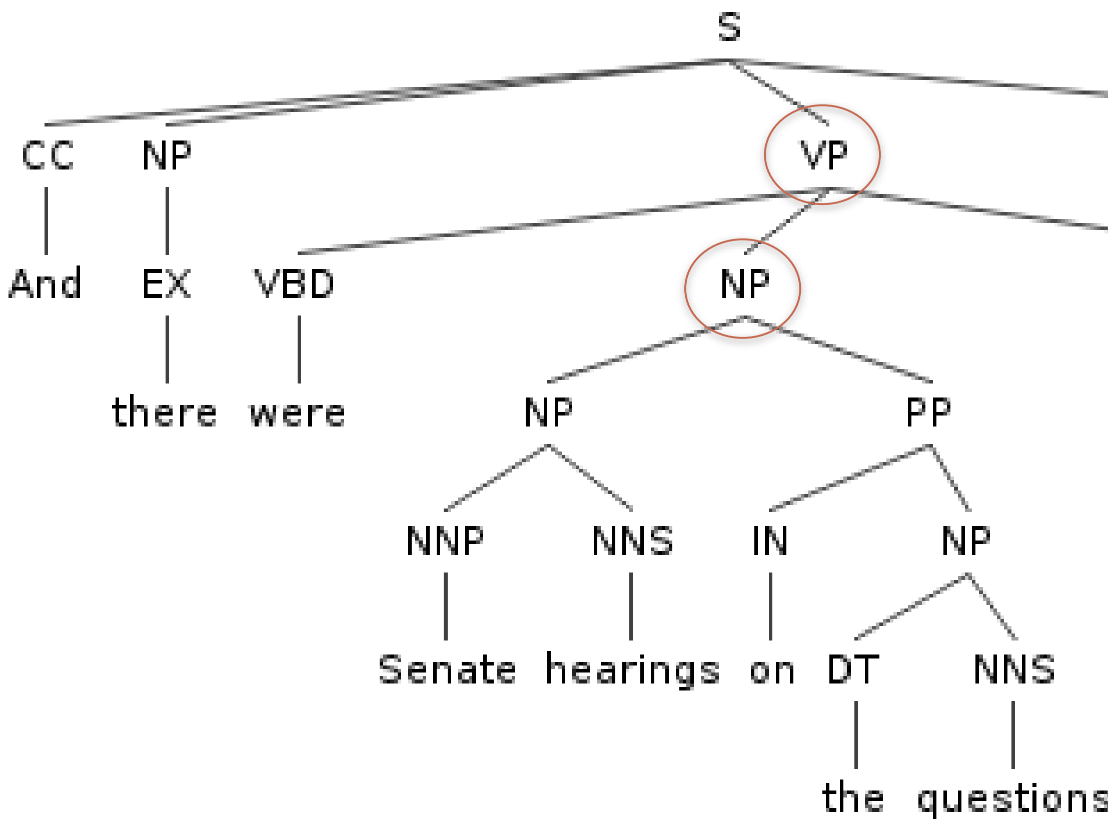
Hedge Transform

- Preserving every constituent of length up to some span *L*



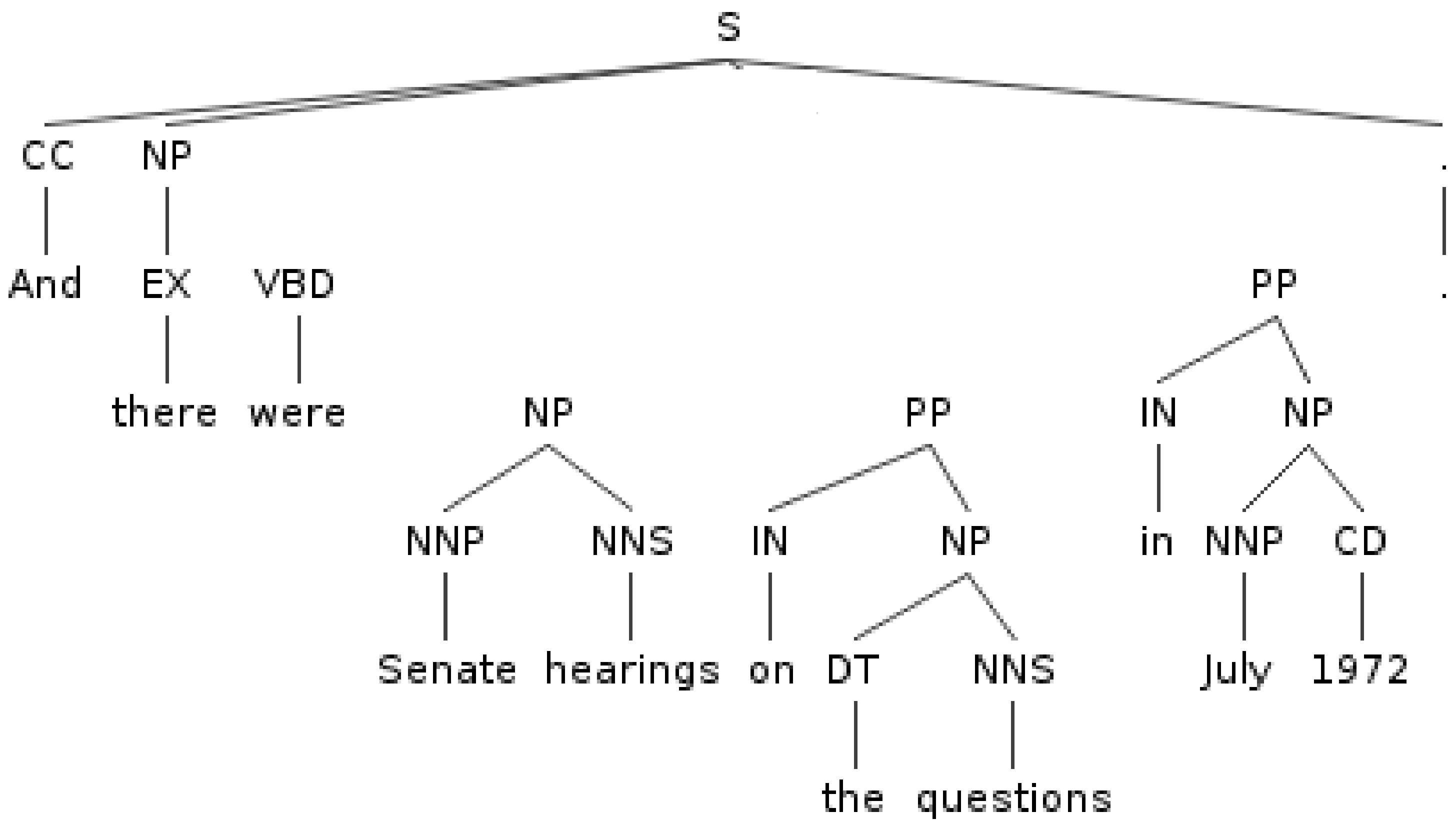
Hedge Transform

- Constituents of span > *L* are recursively removed, children are attached to the parent
  - example: *L=4*



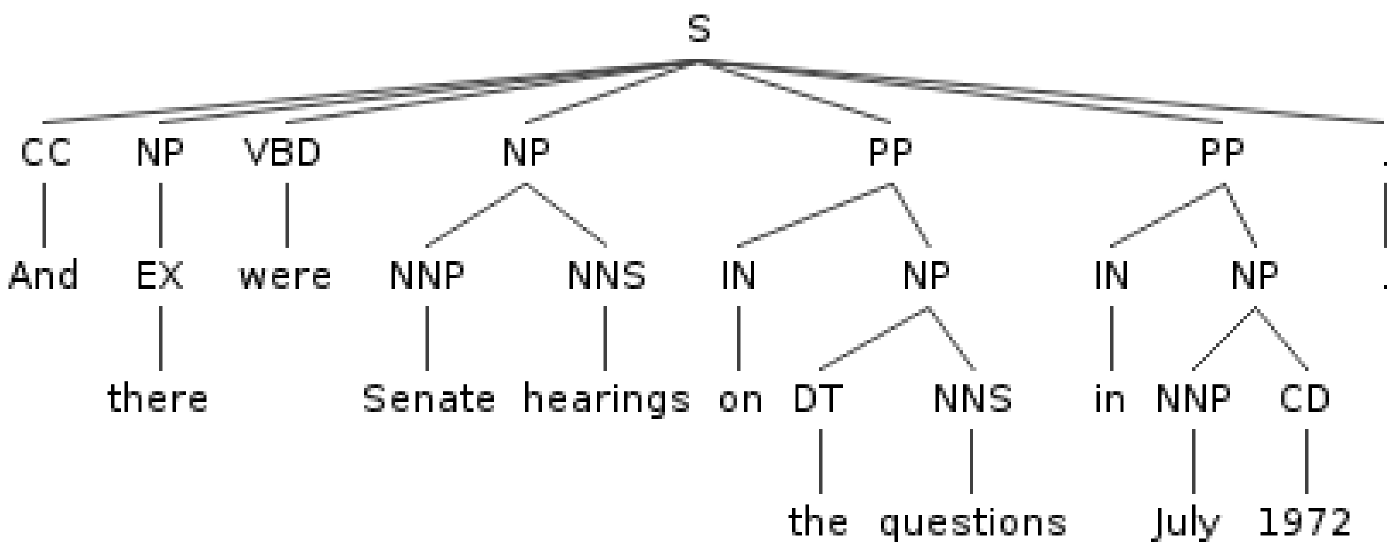
Hedge Transform

- Constituents of span > *L* are recursively removed, children are attached to the parent
  - example: *L=4*



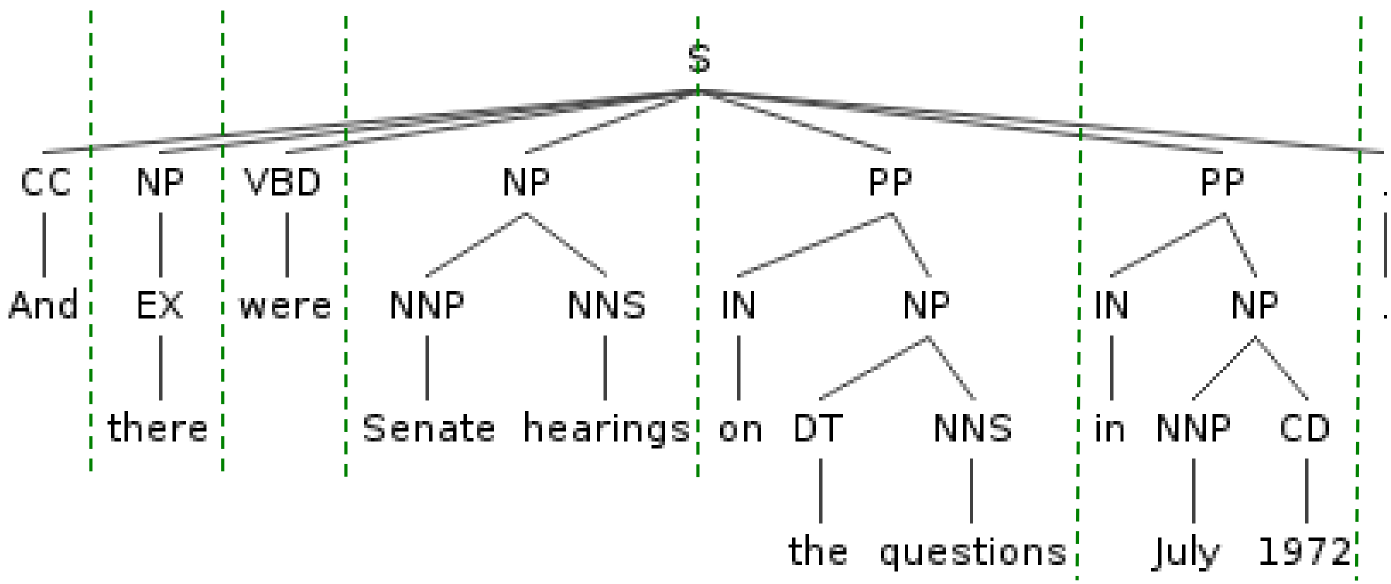
Hedge Transform

- Constituents of span > *L* are recursively removed, children are attached to the parent
  - example: *L=4*



Hedge Transform

- Hedges are sequentially connected to the top-most node, allowing for sentence segmentation before parsing



Hedge Parsing in MT

- Impact of hedge parsing in machine translation (MT):

(1) How does augmenting a translation model with hedge syntax affect a regular (non-incremental) translation?

compared to

- no syntax
- shallow syntax
- full syntax

(2) How does hedge segmentation of the input affect the latency/acc trade-off in an incremental translation?

compared to

- raw segments
- non-linguistic syntax
- shallow syntax

Hedge Parsing in MT

- In summary, the results show:
  - significant improvement in translation quality by using hedge-syntax on the target side of the translation model compared to shallow- or no-syntax
  - comparable to the performance of a full-syntax model
  - hedge-syntax on the source side of the translation model falls behind full syntax although again outperforms shallow syntax
  - hedge parsing of the inputs resulted in an acceptable accuracy/ latency trade-off in simultaneous translation, notably outperforming shallow syntax

Thank You!

Questions?

SST Pipeline