Syntax-based Statistical Machine Translation

Philip Williams and Philipp Koehn

29 October 2014

What Do We Mean by Syntax-based SMT?

- "Syntax-based" is a very inclusive term. It refers to a large family of approaches:
  - Hiero, syntax-directed MT, syntax-augmented MT, syntactified phrase-based MT, tree-to-string, string-to-dependency, dependency treelet-based, soft syntax, fuzzy tree-to-tree, tree-based, ...
- We mean that the translation model uses a tree-based representation of language.
  - We don't count syntax-based preordering or syntactic LMs.
- We will focus on four widely-used approaches:

1. Hierarchical phrase-based
2. Tree-to-string
3. String-to-tree
4. Tree-to-tree

Why Use Syntax?

- Many translation problems can be best explained by pointing to syntax
  - reordering, e.g., verb movement in German–English translation
  - long distance agreement (e.g., subject-verb) in output
- Encourage grammatically coherent output
- Important step towards more linguistically motivated models (semantics)
- State-of-the art for some language pairs
  - Chinese-English (NIST 2008)
  - English-German (WMT 2012)
  - German-English (WMT 2013)

Statistical Machine Translation

Given a source string, s, find the target string, t^*, with the highest probability according to a distribution $p(t|s)$:

$$t^* = \arg\max_t p(t|s)$$

1. Model a probability distribution $p(t|s)$
2. Learn the parameters for the model
3. Find or approximate the highest probability string t^*

Statistical Machine Translation

1. Model a probability distribution $p(t|s)$
   - How is syntax used in modelling?
2. Learn the parameters for the model
   - What are the parameters of a syntax-based model?
3. Find or approximate the highest probability string t^*
   - How do we decode with a syntax-based model?

Modelling $p(t|s)$

- Most SMT models use Och and Ney's (2002) log-linear formulation:

$$p(t|s) = \frac{\exp\left(\sum_{m=1}^{M} \lambda_m h_m(t,s)\right)}{\sum_{t'} \exp\left(\sum_{m=1}^{M} \lambda_m h_m(t',s)\right)}$$

$h_1, \ldots, h_M$ are real-valued functions and $\lambda_1, \ldots, \lambda_M$ are real-valued constants

- Denominator can be ignored during search:

$$\begin{aligned} t^* &= \arg\max_t p(t|s) \\ &= \arg\max_t \sum_{m=1}^{M} \lambda_m h_m(t,s) \end{aligned}$$

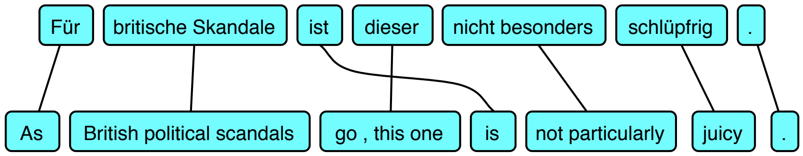
Modelling $p(t|s)$

$$t^* = \arg\max_t \sum_{m=1}^{M} \lambda_m h_m(t,s) \quad (1)$$

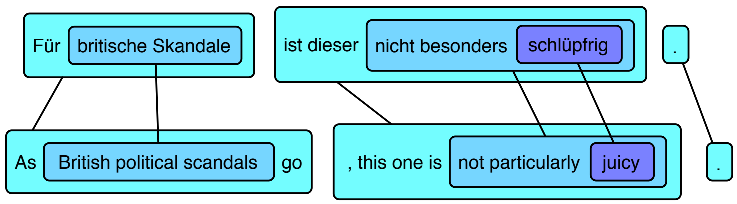
- In word-based models, s and t are modelled as sequences of words.
- In phrase-based models, s and t are modelled as sequences of phrases.
- So what about syntax-based models?

Hierarchical Phrase-based MT

Like phrase pairs. . .

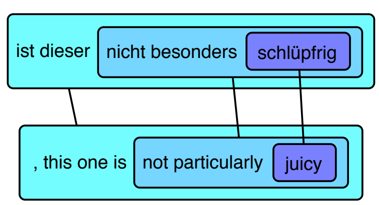


But with nesting:



Hierarchical Phrase-based MT

Hierarchical phrase pairs:



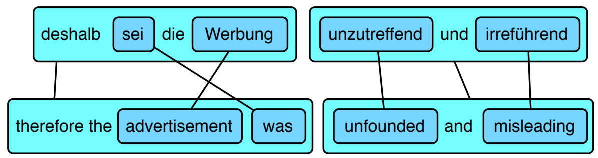
are modelled using Synchronous Context-Free Grammar (SCFG):

$$\mathrm{X} \rightarrow \textit{ist dieser}\ \mathrm{X}_1 \mid \textit{, this one is}\ \mathrm{X}_1$$

$$\mathrm{X} \rightarrow \textit{nicht besonders}\ \mathrm{X}_1 \mid \textit{not particularly}\ \mathrm{X}_1$$

$$\mathrm{X} \rightarrow \textit{schlüpfrig} \mid \textit{juicy}$$

Hierarchical Phrase-based MT



Rules can include up to two non-terminals:

$$\text{X} \rightarrow \textit{deshalb}\ \text{X}_1\ \textit{die}\ \text{X}_2 \mid \textit{therefore the}\ \text{X}_2\ \text{X}_1$$

$$\text{X} \rightarrow \text{X}_1\ \textit{und}\ \text{X}_2 \mid \text{X}_1\ \textit{and}\ \text{X}_2$$

Glue rules concatenate hierarchical phrases:

$$\text{S} \rightarrow \text{X}_1 \mid \text{X}_1$$

$$\text{S} \rightarrow \text{S}_1\ \text{X}_2 \mid \text{S}_1\ \text{X}_2$$

Hierarchical Phrase-based MT

- Synchronous Context-Free Grammar:
  - Rewrite rules of the form $\langle A, B \rangle \rightarrow \langle \alpha, \beta, \sim \rangle$
  - A and B are source and target non-terminals, respectively
  - α and β are strings of terminals and non-terminals for the source and target sides, respectively.
  - $\sim$ is a one-to-one correspondence between source and target non-terminals.

- Hiero grammars are a special case of SCFG:
  - One non-terminal type, X, on source side
  - Two non-terminal types, X and S, on target side
  - Various restrictions on rule form (see Chiang (2007))

SCFG Derivation

$S_1 \mid S_1$

- Derivation starts with pair of linked S symbols.

SCFG Derivation

$S_1 \mid S_1$

$\Rightarrow S_2\ X_3 \mid S_2\ X_3$

- $S \rightarrow S_1\ X_2 \mid S_1\ X_2$ (glue rule)

SCFG Derivation

$S_1 \mid S_1$

$\Rightarrow S_2\ X_3 \mid S_2\ X_3$

$\Rightarrow S_2\ X_4\ \textit{und}\ X_5 \mid S_2\ X_4\ \textit{and}\ X_5$

- $X \rightarrow X_1\ \textit{und}\ X_2 \mid X_1\ \textit{and}\ X_2$

SCFG Derivation

$S_1 \mid S_1$

$\Rightarrow S_2\ X_3 \mid S_2\ X_3$

$\Rightarrow S_2\ X_4\ \textit{und}\ X_5 \mid S_2\ X_4\ \textit{and}\ X_5$

$\Rightarrow S_2\ \textit{unzutreffend}\ \textit{und}\ X_5 \mid S_2\ \textit{unfounded}\ \textit{and}\ X_5$

- $X \rightarrow \textit{unzutreffend} \mid \textit{unfounded}$

SCFG Derivation

$\text{S}_1 \mid \text{S}_1$

$\Rightarrow \text{S}_2\ \text{X}_3 \mid \text{S}_2\ \text{X}_3$

$\Rightarrow \text{S}_2\ \text{X}_4$ *und* $\text{X}_5 \mid \text{S}_2\ \text{X}_4$ *and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und* $\text{X}_5 \mid \text{S}_2$ *unfounded and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und irreführend* $\mid \text{S}_2$ *unfounded and misleading*

- $\text{X} \rightarrow$ *irreführend* $\mid$ *misleading*

SCFG Derivation

$\text{S}_1 \mid \text{S}_1$

$\Rightarrow \text{S}_2\ \text{X}_3 \mid \text{S}_2\ \text{X}_3$

$\Rightarrow \text{S}_2\ \text{X}_4$ *und* $\text{X}_5 \mid \text{S}_2\ \text{X}_4$ *and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und* $\text{X}_5 \mid \text{S}_2$ *unfounded and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und irreführend* $\mid \text{S}_2$ *unfounded and misleading*

$\Rightarrow \text{X}_6$ *unzutreffend und irreführend* $\mid \text{X}_6$ *unfounded and misleading*

- $\text{S} \rightarrow \text{X}_1 \mid \text{X}_1$ (glue rule)

SCFG Derivation

$\text{S}_1 \mid \text{S}_1$

$\Rightarrow \text{S}_2\ \text{X}_3 \mid \text{S}_2\ \text{X}_3$

$\Rightarrow \text{S}_2\ \text{X}_4$ *und* $\text{X}_5 \mid \text{S}_2\ \text{X}_4$ *and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und* $\text{X}_5 \mid \text{S}_2$ *unfounded and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und irreführend* $\mid \text{S}_2$ *unfounded and misleading*

$\Rightarrow \text{X}_6$ *unzutreffend und irreführend* $\mid \text{X}_6$ *unfounded and misleading*

$\Rightarrow$ *deshalb* X_7 *die* X_8 *unzutreffend und irreführend*
$\mid$ *therefore the* $\text{X}_8\ \text{X}_7$ *unfounded and misleading*

- $\text{X} \rightarrow$ *deshalb* X_1 *die* $\text{X}_2 \mid$ *therefore the* $\text{X}_2\ \text{X}_1$ (non-terminal reordering)

SCFG Derivation

$\text{S}_1 \mid \text{S}_1$

$\Rightarrow \text{S}_2\ \text{X}_3 \mid \text{S}_2\ \text{X}_3$

$\Rightarrow \text{S}_2\ \text{X}_4$ *und* $\text{X}_5 \mid \text{S}_2\ \text{X}_4$ *and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und* $\text{X}_5 \mid \text{S}_2$ *unfounded and* X_5

$\Rightarrow \text{S}_2$ *unzutreffend und irreführend* $\mid \text{S}_2$ *unfounded and misleading*

$\Rightarrow \text{X}_6$ *unzutreffend und irreführend* $\mid \text{X}_6$ *unfounded and misleading*

$\Rightarrow$ *deshalb* X_7 *die* X_8 *unzutreffend und irreführend*
$\mid$ *therefore the* $\text{X}_8\ \text{X}_7$ *unfounded and misleading*

$\Rightarrow$ *deshalb sei die* X_8 *unzutreffend und irreführend*
$\mid$ *therefore the* X_8 *was unfounded and misleading*

- $\text{X} \rightarrow$ *sei* $\mid$ *was*

SCFG Derivation

$\text{S}_1 \mid \text{S}_1$

$\Rightarrow$ $\text{S}_2\ \text{X}_3 \mid \text{S}_2\ \text{X}_3$

$\Rightarrow$ $\text{S}_2\ \text{X}_4$ *und* X_5 | $\text{S}_2\ \text{X}_4$ *and* X_5

$\Rightarrow$ S_2 *unzutreffend und* X_5 | S_2 *unfounded and* X_5

$\Rightarrow$ S_2 *unzutreffend und irreführend* | S_2 *unfounded and misleading*

$\Rightarrow$ X_6 *unzutreffend und irreführend* | X_6 *unfounded and misleading*

$\Rightarrow$ *deshalb* X_7 *die* X_8 *unzutreffend und irreführend*
| *therefore the* $\text{X}_8\ \text{X}_7$ *unfounded and misleading*

$\Rightarrow$ *deshalb sei die* X_8 *unzutreffend und irreführend*
| *therefore the* X_8 *was unfounded and misleading*

$\Rightarrow$ *deshalb sei die Werbung unzutreffend und irreführend*
| *therefore the advertisement was unfounded and misleading*

- $\text{X} \rightarrow$ *Werbung* | *advertisement*

Hierarchical Phrase-based MT

- We can now define the search in terms of SCFG derivations

$$t^* = \arg\max_t \sum_{m=1}^{M} \lambda_m h_m(t,s) \quad (1)$$

$$= \arg\max_t \sum_d \sum_{m=1}^{M} \lambda_m h_m(t,s,d) \quad (2)$$

$d \in D$, the set of synchronous derivations with source s and yield t.

- In practice, approximated with search for single-best derivation:

$$d^* = \arg\max_d \sum_{m=1}^{M} \lambda_m h_m(t,s,d) \quad (3)$$

Hierarchical Phrase-based MT

- Search for single-best derivation:

$$d^* = \arg\max_d \sum_{m=1}^{M} \lambda_m h_m(t, s, d) \quad (3)$$

- Rule-local feature functions allow decomposition of derivation scores:

$$h_m(d) = \sum_{r_i} h_m(r_i)$$

- But n-gram language model can't be decomposed this way...

$$d^* = \arg\max_d \left(\lambda_1 \log p_{LM}(d) + \sum_{r_i} \sum_{m=2}^{M} \lambda_m h_m(r_i) \right) \quad (4)$$

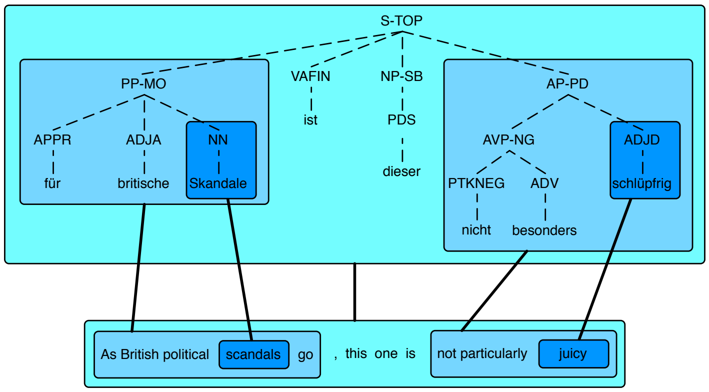
Hierarchical Phrase-based MT

- Summary so far:
  - Generalizes concept of phrase pair to allow nested phrases
  - Formalized using SCFG
  - No use of linguistic annotation: syntactic in a purely formal sense
  - Model uses standard SMT log-linear formulation
  - Search over derivations

- Later:
  - Rule extraction and scoring
  - Decoding (search for best derivation)
  - k-best extraction

Tree-to-String

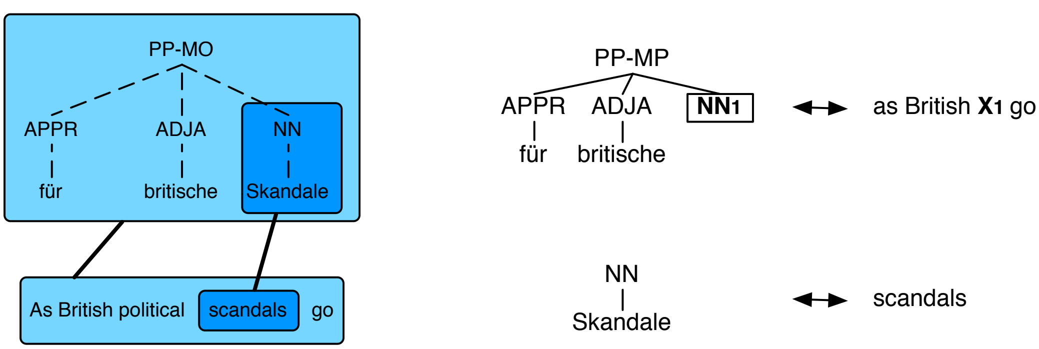
Hierarchical phrase pairs but with embedded tree fragments on the source side:



Each source subphrase is a complete subtree.

Tree-to-String

Formalized using Synchronous Tree-Substitution Grammar (STSG):



Tree-to-String

- Synchronous Tree Substitution Grammar (STSG):
  - Grammar rules have the form $\langle \pi, \gamma, \sim \rangle$
  - π is a tree with source terminal and non-terminal leaves
  - γ is a string[1] of target terminals and non-terminals
  - $\sim$ is a one-to-one correspondence between source and target non-terminals.
- Unlike Hiero:
  - Linguistic-annotation (on source-side)
  - No limit to number of substitution sites (non-terminals)
  - No reordering limit during decoding

[1] Technically, a 1-level tree formed by adding X as the root and the symbols from γ as children.

Tree-to-String

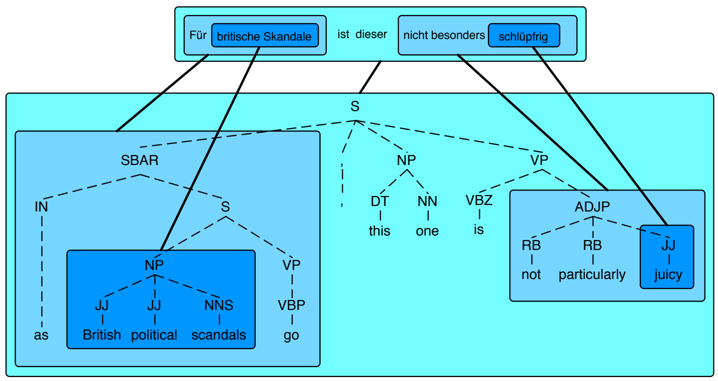
- Derivation involves synchronous rewrites (like SCFG)
- Tree fragments required to match input parse tree.
  - Motivation: tree provides context for rule selection ("syntax-directed")
- Efficient decoding algorithms available: source tree constrains rule options
- Search for single-best derivation:

$$d^* = \arg\max_d \left(\lambda_1 \log p_{LM}(d) + \sum_{r_i} \sum_{m=2}^{M} \lambda_m h_m(r_i) \right)$$

where source-side of d must match input tree

String-to-Tree

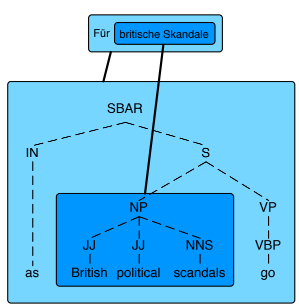
Hierarchical phrase pairs but with embedded tree fragments on the target side:

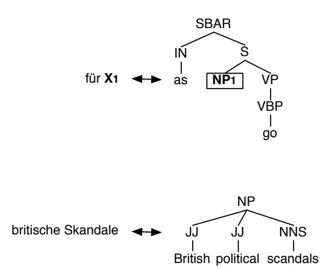


Each target subphrase is a complete subtree.

String-to-Tree

Formalized using STSG:





Or SCFG:

$$\text{SBAR} \rightarrow \textit{für } \text{X}_1 \mid \textit{as } \text{NP}_1 \textit{ go}$$

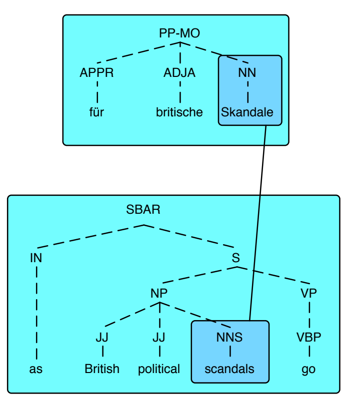
$$\text{NP} \rightarrow \textit{britische Skandale} \mid \textit{British political scandals}$$

String-to-Tree

- Derivation is a rewriting process, like hierachical phrase-based and tree-to-string
  - Rewrites only allowed if target labels match at substitution sites
  - Internal tree structure not used in derivation (hence frequent use of SCFG)
  - Motivation: constraints provided by target syntax lead to more fluent output
- Later:
  - Rule extraction and scoring
  - Decoding (Hiero will be special case of S2T)
  - k-best extraction (likewise)

Tree-to-Tree

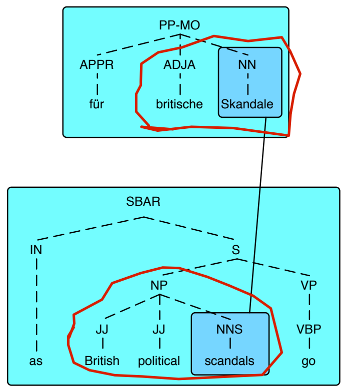
Hierarchical phrase pairs but with embedded tree fragments on both sides:



Formalized using STSG

Tree-to-Tree

Differences in source and target syntactic structure increasingly important



Can be differences in treebank annotation style or simply differences in language choice

Summary So Far

- We have introduced four models:

Model	**Formalism**	**Source Syntax**	**Target Syntax**	**Input**
Hiero	SCFG	N	N	string
T2S	STSG	Y	N	tree
S2T	STSG or SCFG	N	Y	string
T2T	STSG	Y	Y	tree

- Next:
  - Rule extraction

Part I - Introduction

Part II - Rule Extraction

Part III - Decoding

Part IV - Extensions

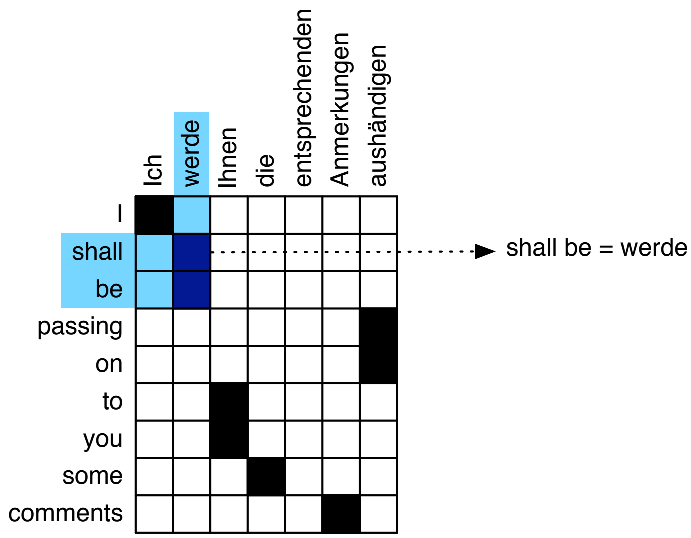
Learning Synchronous Grammars

- Extracting rules from a word-aligned parallel corpus

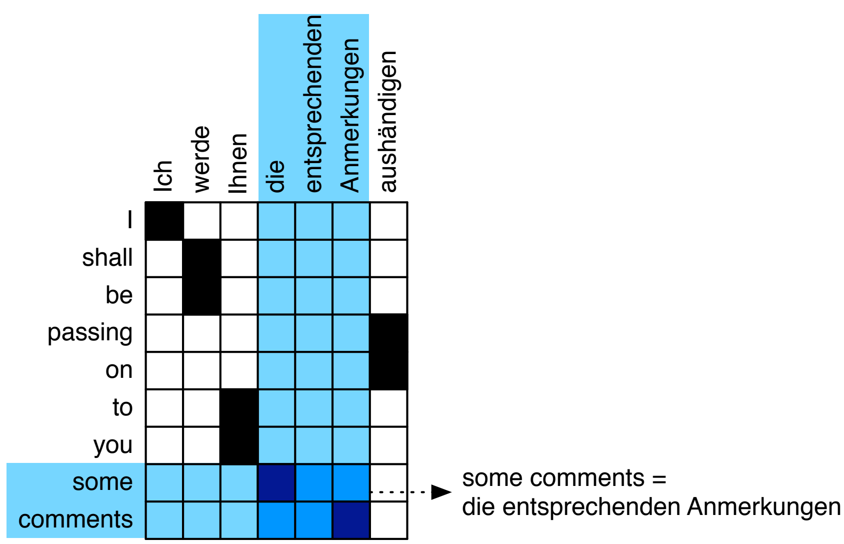
- First: Hierarchical phrase-based model
  - only one non-terminal symbol X
  - no linguistic syntax, just a formally syntactic model

- Then: Synchronous phrase structure model
  - non-terminals for words and phrases: NP, VP, PP, ADJ, ...
  - corpus must also be parsed with syntactic parser

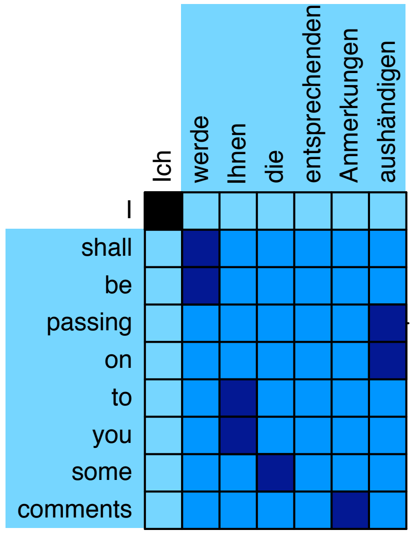
Extracting Phrase Translation Rules



Extracting Phrase Translation Rules

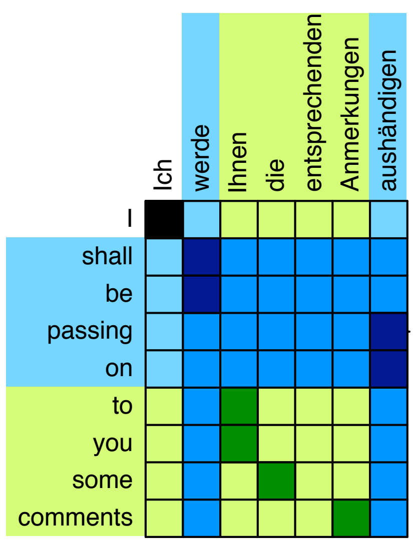


Extracting Phrase Translation Rules



werde Ihnen die entsprechenden Anmerkungen aushändigen
= shall be passing on to you some comments

Extracting Hierarchical Phrase Translation Rules



subtracting subphrase

werde X aushändigen
= shall be passing on X

Formal Definition

- Recall: consistent phrase pairs

$$
\begin{aligned}
(\bar{e}, \bar{f}) \text{ consistent with } A \Leftrightarrow& \\
& \forall e_i \in \bar{e} : (e_i, f_j) \in A \rightarrow f_j \in \bar{f} \\
& \text{AND } \forall f_j \in \bar{f} : (e_i, f_j) \in A \rightarrow e_i \in \bar{e} \\
& \text{AND } \exists e_i \in \bar{e}, f_j \in \bar{f} : (e_i, f_j) \in A
\end{aligned}
$$

- Let P be the set of all extracted phrase pairs $(\bar{e}, \bar{f})$

Formal Definition

- Extend recursively:

$$
\begin{aligned}
\text{if } (\bar{e}, \bar{f}) \in P \text{ AND } (\bar{e}_{\text{SUB}}, \bar{f}_{\text{SUB}}) \in P& \\
\text{AND } \bar{e} = \bar{e}_{\text{PRE}} + \bar{e}_{\text{SUB}} + \bar{e}_{\text{POST}}& \\
\text{AND } \bar{f} = \bar{f}_{\text{PRE}} + \bar{f}_{\text{SUB}} + \bar{f}_{\text{POST}}& \\
\text{AND } \bar{e} \neq \bar{e}_{\text{SUB}} \text{ AND } \bar{f} \neq \bar{f}_{\text{SUB}}& \\
\text{add } (e_{\text{PRE}} + \text{X} + e_{\text{POST}}, f_{\text{PRE}} + \text{X} + f_{\text{POST}}) \text{ to } P&
\end{aligned}
$$

(note: any of e_{PRE}, e_{POST}, f_{PRE}, or f_{POST} may be empty)

- Set of hierarchical phrase pairs is the closure under this extension mechanism

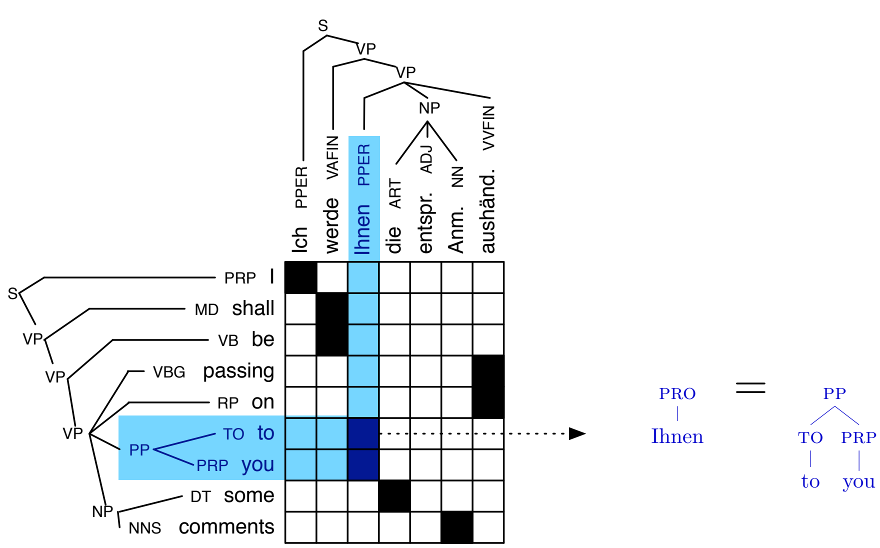
Comments

- Removal of multiple sub-phrases leads to rules with multiple non-terminals, such as:

$$\text{Y} \rightarrow \text{X}_1\ \text{X}_2 \mid \text{X}_2\ \textit{of}\ \text{X}_1$$

- Typical restrictions to limit complexity [Chiang, 2005]
  - at most 2 nonterminal symbols
  - at least 1 but at most 5 words per language
  - span at most 15 words (counting gaps)

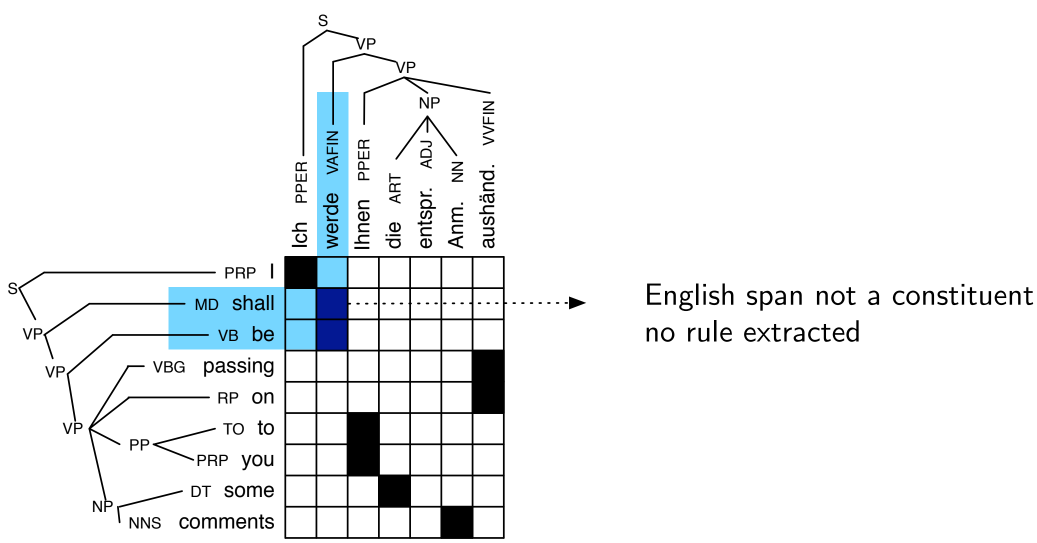
Learning Syntactic Translation Rules



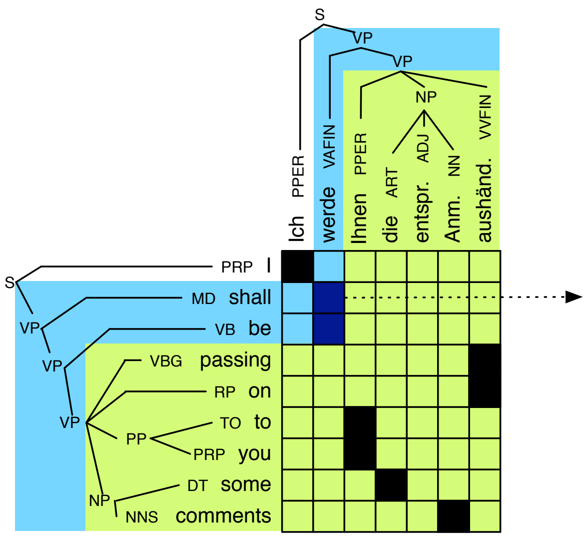
Constraints on Syntactic Rules

- Same word alignment constraints as hierarchical models
- Hierarchical: rule can cover any span
  ⇔ syntactic rules must cover constituents in the tree
- Hierarchical: gaps may cover any span
  ⇔ gaps must cover constituents in the tree
- Much fewer rules are extracted (all things being equal)

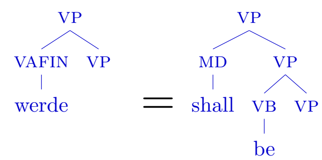
Impossible Rules



Rules with Context



Rule with this phrase pair requires syntactic context



Too Many Rules Extractable

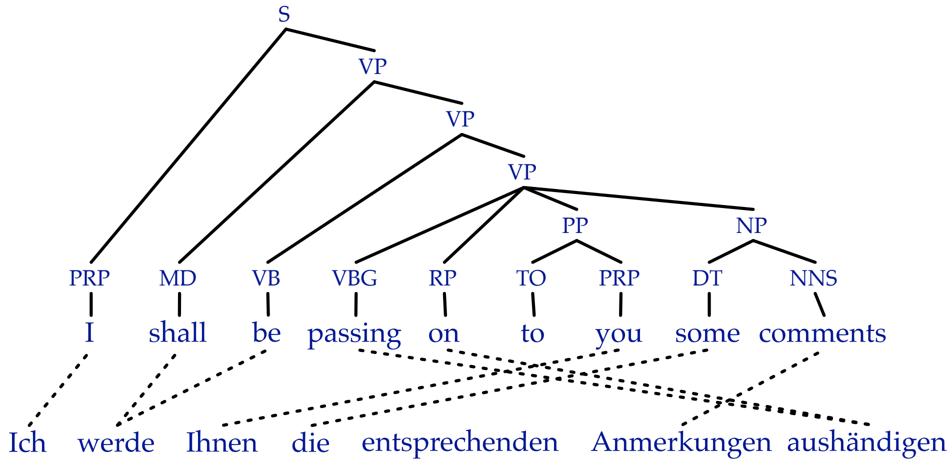
- Huge number of rules can be extracted
  (every alignable node may or may not be part of a rule → exponential number of rules)

- Need to limit which rules to extract

- Option 1: similar restriction as for hierarchical model
  (maximum span size, maximum number of terminals and non-terminals, etc.)

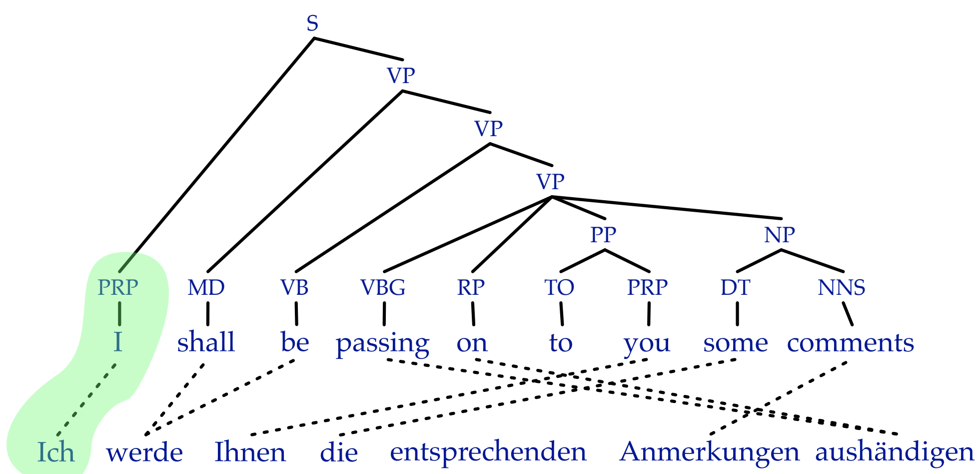
- Option 2: only extract minimal rules ("GHKM" rules)

Minimal Rules



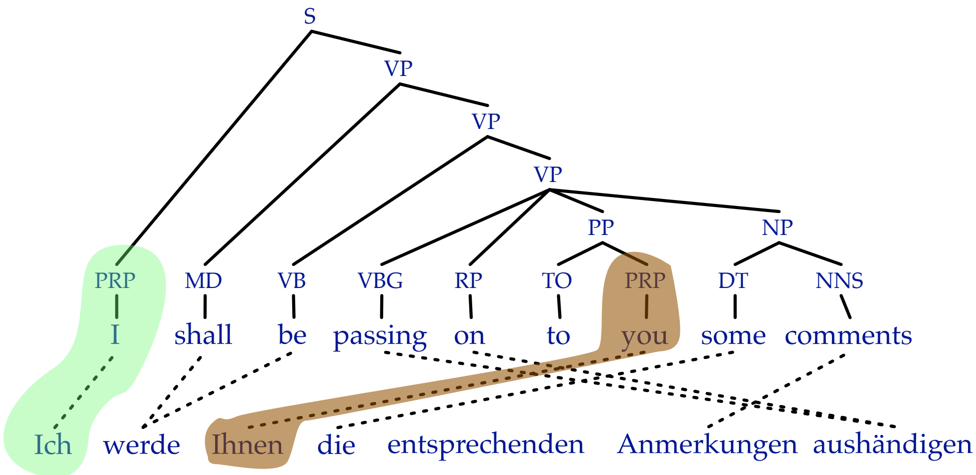
Extract: set of smallest rules required to explain the sentence pair

Lexical Rule



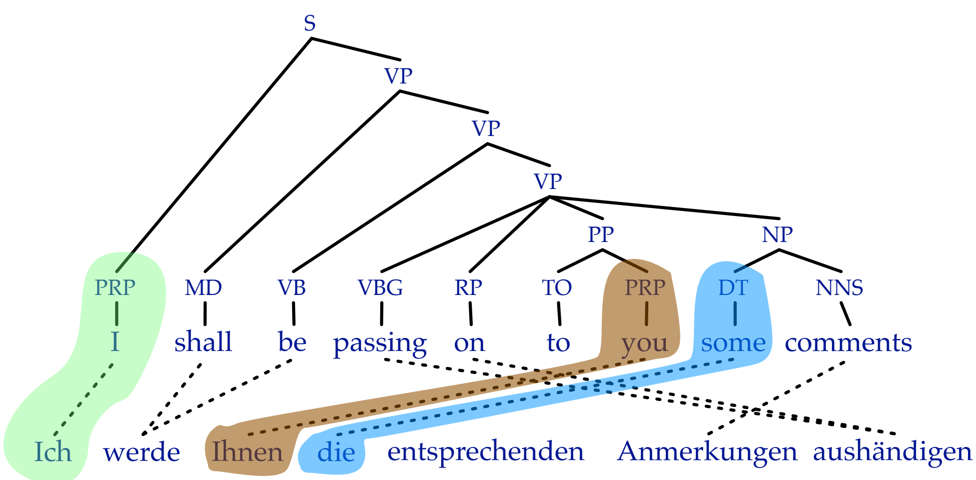
Extracted rule: PRP $\rightarrow$ Ich | I

Lexical Rule



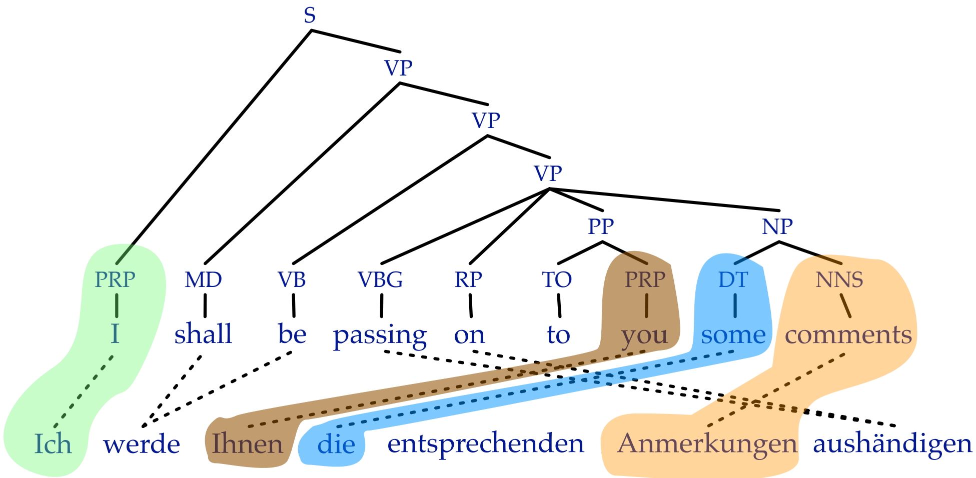
Extracted rule: PRP → Ihnen | you

Lexical Rule



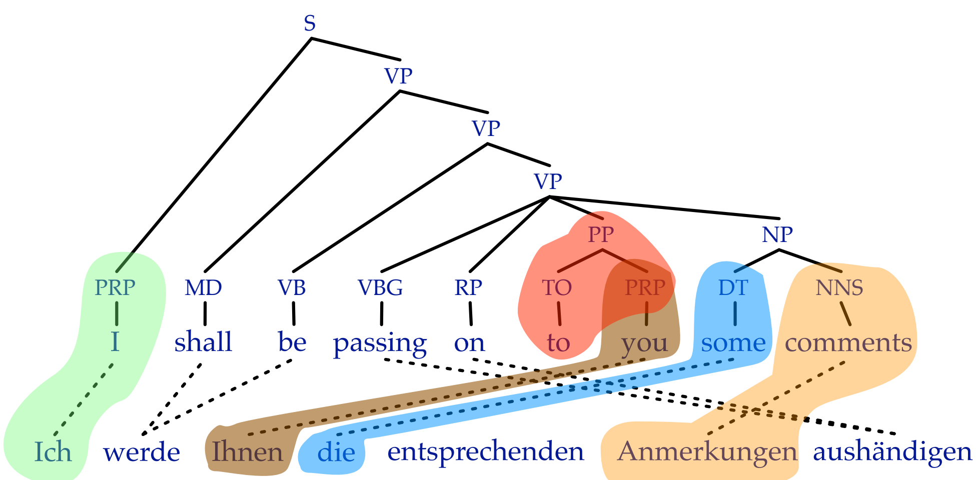
Extracted rule: DT → die | some

Lexical Rule



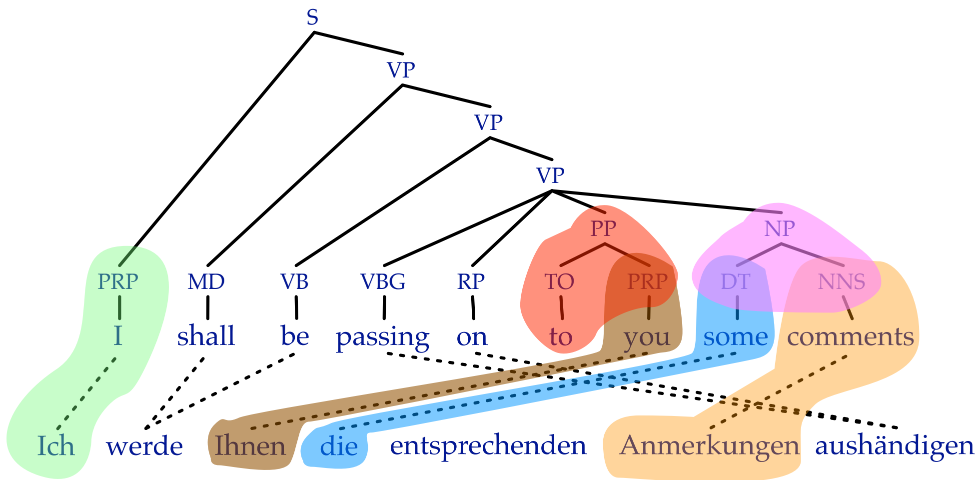
Extracted rule: NNS → Anmerkungen | comments

Insertion Rule



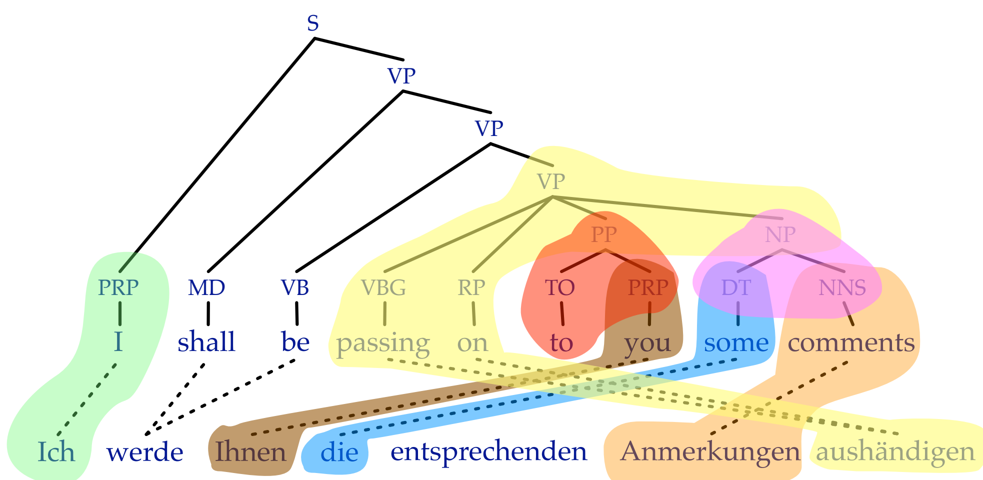
Extracted rule: PP → X | to PRP

Non-Lexical Rule



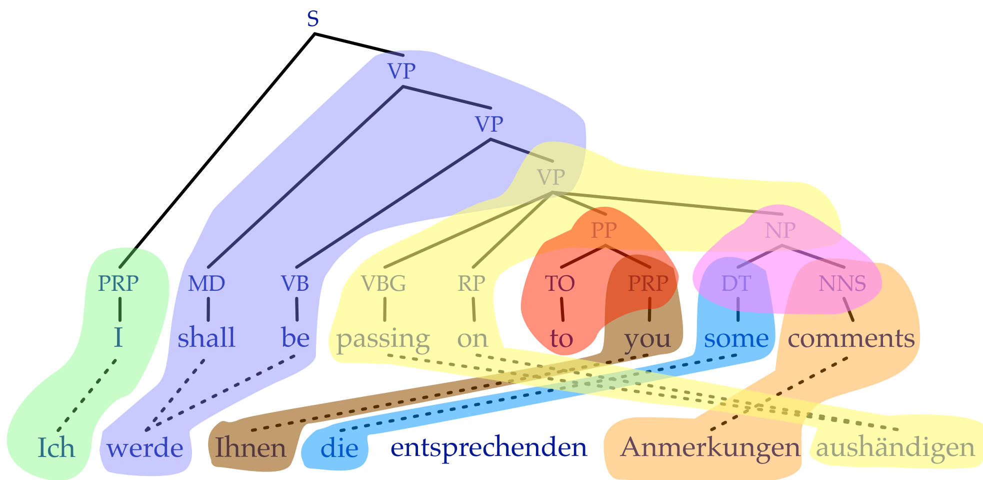
Extracted rule: NP → X_1 X_2 | DT_1 NNS_2

Lexical Rule with Syntactic Context



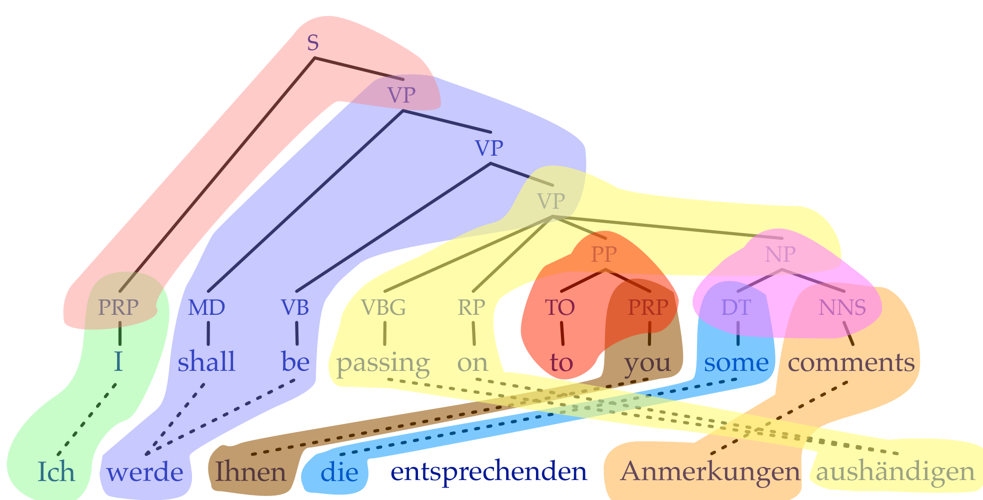
Extracted rule: VP → X_1 X_2 aushändigen | passing on PP_1 NP_2

Lexical Rule with Syntactic Context



Extracted rule: VP → werde X | shall be VP (ignoring internal structure)

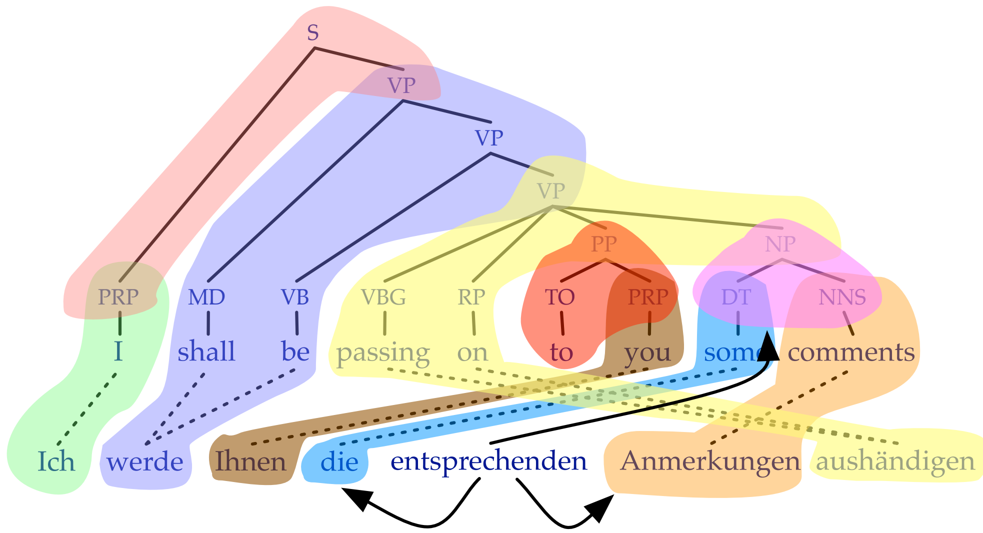
Non-Lexical Rule



Extracted rule: S → X_1 X_2 | PRP_1 VP_2

DONE — note: one rule per alignable constituent

Unaligned Source Words



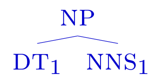
Attach to neighboring words or higher nodes → additional rules

Too Few Phrasal Rules?

- Lexical rules will be 1-to-1 mappings (unless word alignment requires otherwise)
- But: phrasal rules very beneficial in phrase-based models
- Solutions
  - combine rules that contain a maximum number of symbols (as in hierarchical models, recall: "Option 1")
  - compose minimal rules to cover a maximum number of non-leaf nodes

Composed Rules

- Current rules $x_1\ x_2$ =

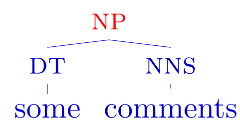


die =

entsprechenden Anmerkungen =

- Composed rule

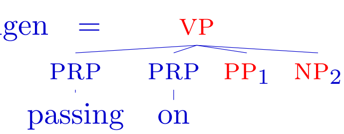
die entsprechenden Anmerkungen =



(1 non-leaf node: NP)

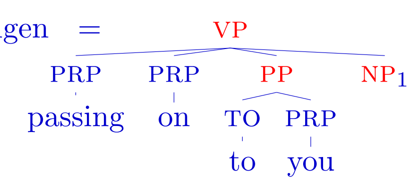
Composed Rules

- Minimal rule: $x_1\ x_2$ aushändigen =



3 non-leaf nodes:
VP, PP, NP

- Composed rule: Ihnen x_1 aushändigen =



3 non-leaf nodes:
VP, PP and NP

Relaxing Tree Constraints

- Impossible rule

$$\text{X} \;/\; \text{werde} \quad = \quad \text{MD} \;/\; \text{shall} \quad \text{VB} \;/\; \text{be}$$

- Create new non-terminal label: MD+VB

⇒ New rule

$$\text{X} \;/\; \text{werde} \quad = \quad \text{MD+VB} \;/\; (\text{MD} \;/\; \text{shall} \quad \text{VB} \;/\; \text{be})$$

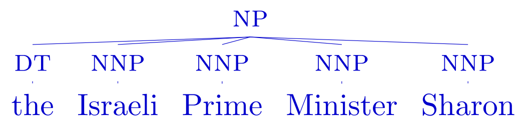
Zollmann Venugopal Relaxation

- If span consists of two constituents , join them: X+Y
- If span conststs of three constituents, join them: X+Y+Z
- If span covers constituents with the same parent X and include
  - every but the first child Y, label as X\Y
  - every but the last child Y, label as X/Y
- For all other cases, label as FAIL

⇒ More rules can be extracted, but number of non-terminals blows up

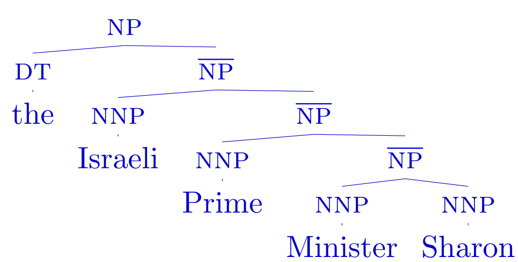
Special Problem: Flat Structures

- Flat structures severely limit rule extraction



- Can only extract rules for individual words or entire phrase

Relaxation by Tree Binarization



More rules can be extracted

Left-binarization or right-binarization?

Scoring Translation Rules

- Extract all rules from corpus
- Score based on counts
  - joint rule probability: $p(\text{LHS}, \text{RHS}_f, \text{RHS}_e)$
  - rule application probability: $p(\text{RHS}_f, \text{RHS}_e | \text{LHS})$
  - direct translation probability: $p(\text{RHS}_e | \text{RHS}_f, \text{LHS})$
  - noisy channel translation probability: $p(\text{RHS}_f | \text{RHS}_e, \text{LHS})$
  - lexical translation probability: $\prod_{e_i \in \text{RHS}_e} p(e_i | \text{RHS}_f, a)$

Part I - Introduction
Part II - Rule Extraction
Part III - Decoding
Part IV - Extensions

Outline

1. Hiero/S2T decoding (SCFG with string input)
   - Viterbi decoding with local features (-LM)
   - k-best extraction
   - LM integration (cube pruning)
   - The S2T algorithm, as implemented in Moses
2. T2S decoding (STSG with tree input)
   - Vanilla T2S: non-directional, cube pruning
3. T2T decoding (STSG with tree input)
   - Included for completeness — better alternatives explored later

Viterbi S2T Decoding (-LM)

Objective Find the highest-scoring synchronous derivation d^*

Input $s_1\ s_2\ \ldots\ s_n$

Grammar

$$
\begin{array}{lcclr}
r_1 & \mathrm{C}_1 & \rightarrow & \alpha_1 \mid \beta_1 & w_1 \\
r_2 & \mathrm{C}_2 & \rightarrow & \alpha_2 \mid \beta_2 & w_2 \\
r_3 & \mathrm{C}_3 & \rightarrow & \alpha_3 \mid \beta_3 & w_3 \\
\ldots & & & & \\
r_{|G|} & \mathrm{C}_{|G|} & \rightarrow & \alpha_{|G|} \mid \beta_{|G|} & w_{|G|}
\end{array}
$$

- C_i, α_i and β_i are LHS, source RHS, target RHS of rule r_i, respectively.
- w_i is weight of rule r_i (weighted product of rule-local feature functions).
- $|G|$ is the number of rules in the grammar G.

Viterbi S2T Decoding (-LM)

Objective Find the highest-scoring synchronous derivation d^*

Solution

1. Project grammar
   Project weighted SCFG to weighted CFG
   $f : G \rightarrow G'$ (many-to-one rule mapping)

2. Parse
   Find Viterbi parse of sentence wrt G'

3. Translate
   Produce synchronous tree pair by applying inverse projection f'

Example

Input jemand mußte Josef K. verleumdet haben
someone must Josef K. slandered have

Grammar

r_1:	NP	$\rightarrow$	*Josef K.* | *Josef K.*	0.90
r_2:	VBN	$\rightarrow$	*verleumdet* | *slandered*	0.40
r_3:	VBN	$\rightarrow$	*verleumdet* | *defamed*	0.20
r_4:	VP	$\rightarrow$	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
r_5:	S	$\rightarrow$	*jemand* X_1 | *someone* VP_1	0.60
r_6:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
r_7:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

(Six derivations in total)

Example

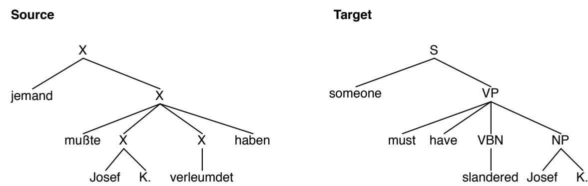
Input jemand mußte Josef K. verleumdet haben

someone must Josef K. slandered have

Grammar

⇒	r_1:	NP	→	*Josef K.* | *Josef K.*	0.90
⇒	r_2:	VBN	→	*verleumdet* | *slandered*	0.40
	r_3:	VBN	→	*verleumdet* | *defamed*	0.20
⇒	r_4:	VP	→	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
⇒	r_5:	S	→	*jemand* X_1 | *someone* VP_1	0.60
	r_6:	S	→	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
	r_7:	S	→	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

Derivation 1



Example

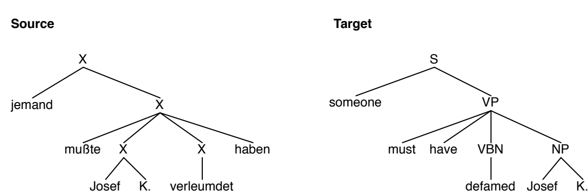
Input jemand mußte Josef K. verleumdet haben

someone must Josef K. slandered have

Grammar

⇒	r_1:	NP	→	*Josef K.* | *Josef K.*	0.90
	r_2:	VBN	→	*verleumdet* | *slandered*	0.40
⇒	r_3:	VBN	→	*verleumdet* | *defamed*	0.20
⇒	r_4:	VP	→	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
⇒	r_5:	S	→	*jemand* X_1 | *someone* VP_1	0.60
	r_6:	S	→	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
	r_7:	S	→	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

Derivation 2



Example

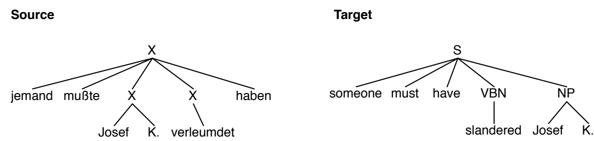
Input jemand mußte Josef K. verleumdet haben

someone must Josef K. slandered have

Grammar

⇒ r_1:	NP	→	*Josef K.*	*Josef K.*	0.90
⇒ r_2:	VBN	→	*verleumdet*	*slandered*	0.40
r_3:	VBN	→	*verleumdet*	*defamed*	0.20
r_4:	VP	→	*mußte* X_1 X_2 *haben*	*must have* VBN_2 NP_1	0.10
r_5:	S	→	*jemand* X_1	*someone* VP_1	0.60
⇒ r_6:	S	→	*jemand mußte* X_1 X_2 *haben*	*someone must have* VBN_2 NP_1	0.80
r_7:	S	→	*jemand mußte* X_1 X_2 *haben*	NP_1 *must have been* VBN_1 *by someone*	0.05

Derivation 3



Example

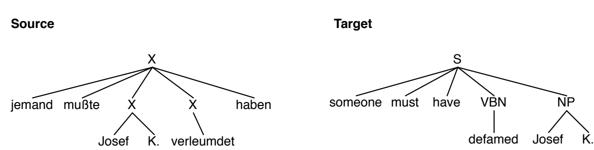
Input jemand mußte Josef K. verleumdet haben

someone must Josef K. slandered have

Grammar

⇒ r_1:	NP	→	*Josef K.*	*Josef K.*	0.90
r_2:	VBN	→	*verleumdet*	*slandered*	0.40
⇒ r_3:	VBN	→	*verleumdet*	*defamed*	0.20
r_4:	VP	→	*mußte* X_1 X_2 *haben*	*must have* VBN_2 NP_1	0.10
r_5:	S	→	*jemand* X_1	*someone* VP_1	0.60
⇒ r_6:	S	→	*jemand mußte* X_1 X_2 *haben*	*someone must have* VBN_2 NP_1	0.80
r_7:	S	→	*jemand mußte* X_1 X_2 *haben*	NP_1 *must have been* VBN_1 *by someone*	0.05

Derivation 4



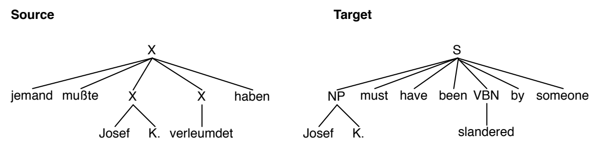
Example

Input jemand mußte Josef K. verleumdet haben
someone must Josef K. slandered have

Grammar

⇒ r_1:	NP	→	*Josef K.* | *Josef K.*		0.90
⇒ r_2:	VBN	→	*verleumdet* | *slandered*		0.40
r_3:	VBN	→	*verleumdet* | *defamed*		0.20
r_4:	VP	→	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1		0.10
r_5:	S	→	*jemand* X_1 | *someone* VP_1		0.60
r_6:	S	→	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1		0.80
⇒ r_7:	S	→	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*		0.05

Derivation 5



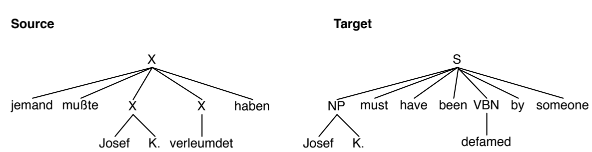
Example

Input jemand mußte Josef K. verleumdet haben
someone must Josef K. slandered have

Grammar

⇒ r_1:	NP	→	*Josef K.* | *Josef K.*		0.90
r_2:	VBN	→	*verleumdet* | *slandered*		0.40
⇒ r_3:	VBN	→	*verleumdet* | *defamed*		0.20
r_4:	VP	→	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1		0.10
r_5:	S	→	*jemand* X_1 | *someone* VP_1		0.60
r_6:	S	→	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1		0.80
⇒ r_7:	S	→	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*		0.05

Derivation 6



Step 1: Project Grammar to CFG

G	r_1:	NP	$\rightarrow$	*Josef K.* | *Josef K.*	0.90
	r_2:	VBN	$\rightarrow$	*verleumdet* | *slandered*	0.40
	r_3:	VBN	$\rightarrow$	*verleumdet* | *defamed*	0.20
	r_4:	VP	$\rightarrow$	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
	r_5:	S	$\rightarrow$	*jemand* X_1 | *someone* VP_1	0.60
	r_6:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
	r_7:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

G'	q_1:	NP	$\rightarrow$	*Josef K.*	0.90
	q_2:	VBN	$\rightarrow$	*verleumdet*	0.40
	q_3:	VP	$\rightarrow$	*mußte* NP VBN *haben*	0.10
	q_4:	S	$\rightarrow$	*jemand* VP	0.60
	q_5:	S	$\rightarrow$	*jemand mußte* NP VBN *haben*	0.80

- G is original synchronous grammar, G' is monolingual projection

Step 1: Project Grammar to CFG

G	$\Rightarrow r_1$:	NP	$\rightarrow$	*Josef K.* | *Josef K.*	0.90
	r_2:	VBN	$\rightarrow$	*verleumdet* | *slandered*	0.40
	r_3:	VBN	$\rightarrow$	*verleumdet* | *defamed*	0.20
	r_4:	VP	$\rightarrow$	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
	r_5:	S	$\rightarrow$	*jemand* X_1 | *someone* VP_1	0.60
	r_6:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
	r_7:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

G'	$\Rightarrow q_1$:	NP	$\rightarrow$	*Josef K.*	0.90
	q_2:	VBN	$\rightarrow$	*verleumdet*	0.40
	q_3:	VP	$\rightarrow$	*mußte* NP VBN *haben*	0.10
	q_4:	S	$\rightarrow$	*jemand* VP	0.60
	q_5:	S	$\rightarrow$	*jemand mußte* NP VBN *haben*	0.80

- Projected rule gets LHS and source RHS (but with target non-terminal labels)

Step 1: Project Grammar to CFG

G

	r_1:	NP	$\rightarrow$	*Josef K.* | *Josef K.*	0.90
$\Rightarrow$	r_2:	VBN	$\rightarrow$	*verleumdet* | *slandered*	0.40
$\Rightarrow$	r_3:	VBN	$\rightarrow$	*verleumdet* | *defamed*	0.20
	r_4:	VP	$\rightarrow$	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
	r_5:	S	$\rightarrow$	*jemand* X_1 | *someone* VP_1	0.60
	r_6:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
	r_7:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

G'

	q_1:	NP	$\rightarrow$	*Josef K.*	0.90
$\Rightarrow$	q_2:	VBN	$\rightarrow$	*verleumdet*	0.40
	q_3:	VP	$\rightarrow$	*mußte* NP VBN *haben*	0.10
	q_4:	S	$\rightarrow$	*jemand* VP	0.60
	q_5:	S	$\rightarrow$	*jemand mußte* NP VBN *haben*	0.80

- Many-to-one: weight of projected rule is the best from set of projecting rules

Step 1: Project Grammar to CFG

G

	r_1:	NP	$\rightarrow$	*Josef K.* | *Josef K.*	0.90
	r_2:	VBN	$\rightarrow$	*verleumdet* | *slandered*	0.40
	r_3:	VBN	$\rightarrow$	*verleumdet* | *defamed*	0.20
$\Rightarrow$	r_4:	VP	$\rightarrow$	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
	r_5:	S	$\rightarrow$	*jemand* X_1 | *someone* VP_1	0.60
	r_6:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
	r_7:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

G'

	q_1:	NP	$\rightarrow$	*Josef K.*	0.90
	q_2:	VBN	$\rightarrow$	*verleumdet*	0.40
$\Rightarrow$	q_3:	VP	$\rightarrow$	*mußte* NP VBN *haben*	0.10
	q_4:	S	$\rightarrow$	*jemand* VP	0.60
	q_5:	S	$\rightarrow$	*jemand mußte* NP VBN *haben*	0.80

- Target non-terminal labels projected to monolingual rule (in source order)

Step 1: Project Grammar to CFG

G		r_1:	NP	$\rightarrow$	*Josef K.* | *Josef K.*	0.90
		r_2:	VBN	$\rightarrow$	*verleumdet* | *slandered*	0.40
		r_3:	VBN	$\rightarrow$	*verleumdet* | *defamed*	0.20
		r_4:	VP	$\rightarrow$	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
	$\Rightarrow$	r_5:	S	$\rightarrow$	*jemand* X_1 | *someone* VP_1	0.60
		r_6:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
		r_7:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

G'		q_1:	NP	$\rightarrow$	*Josef K.*	0.90
		q_2:	VBN	$\rightarrow$	*verleumdet*	0.40
		q_3:	VP	$\rightarrow$	*mußte* NP VBN *haben*	0.10
	$\Rightarrow$	q_4:	S	$\rightarrow$	*jemand* VP	0.60
		q_5:	S	$\rightarrow$	*jemand mußte* NP VBN *haben*	0.80

- And so on. . .

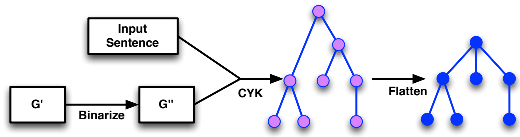
Step 1: Project Grammar to CFG

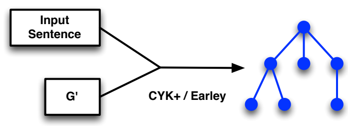
G		r_1:	NP	$\rightarrow$	*Josef K.* | *Josef K.*	0.90
		r_2:	VBN	$\rightarrow$	*verleumdet* | *slandered*	0.40
		r_3:	VBN	$\rightarrow$	*verleumdet* | *defamed*	0.20
		r_4:	VP	$\rightarrow$	*mußte* X_1 X_2 *haben* | *must have* VBN_2 NP_1	0.10
		r_5:	S	$\rightarrow$	*jemand* X_1 | *someone* VP_1	0.60
	$\Rightarrow$	r_6:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | *someone must have* VBN_2 NP_1	0.80
	$\Rightarrow$	r_7:	S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben* | NP_1 *must have been* VBN_1 *by someone*	0.05

G'		q_1:	NP	$\rightarrow$	*Josef K.*	0.90
		q_2:	VBN	$\rightarrow$	*verleumdet*	0.40
		q_3:	VP	$\rightarrow$	*mußte* NP VBN *haben*	0.10
		q_4:	S	$\rightarrow$	*jemand* VP	0.60
	$\Rightarrow$	q_5:	S	$\rightarrow$	*jemand mußte* NP VBN *haben*	0.80

- And so on.

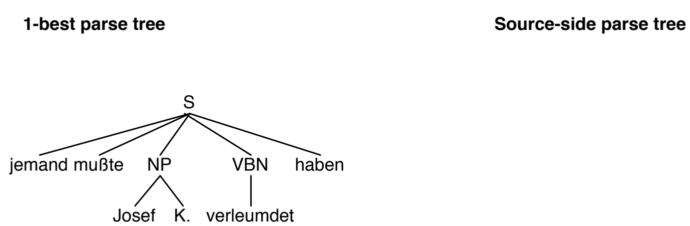
Step 2: Find Viterbi Parse





- Standard weighted parsing algorithms.
- Binarization can be explicit (like CYK) or implicit (like Earley / CYK+)

Step 3: Reconstruct Synchronous Derivation



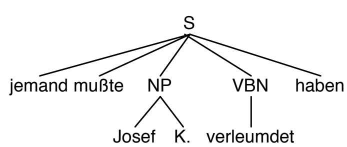
Step 3: Reconstruct Synchronous Derivation

1-best parse tree **Source-side parse tree**

- Source-side: replace non-terminals with Xs

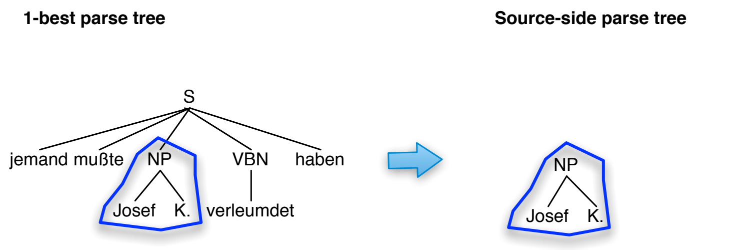
Step 3: Reconstruct Synchronous Derivation

1-best parse tree **Source-side parse tree**



- Target-side: invert grammar projection

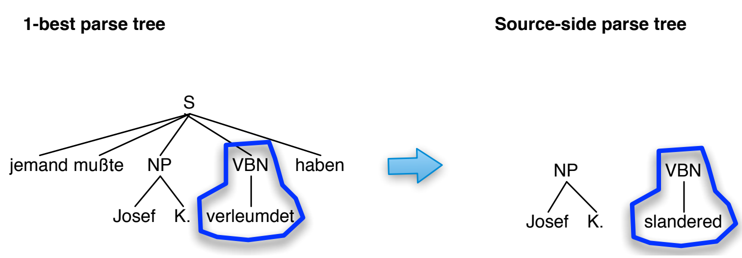
Step 3: Reconstruct Synchronous Derivation



- Target-side: invert grammar projection

NP → *Josef K.* | *Josef K.*

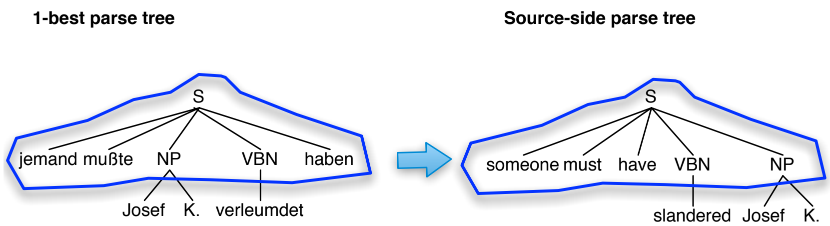
Step 3: Reconstruct Synchronous Derivation



- Target-side: invert grammar projection (multiple rules? pick highest-scoring)

VBN	→	*verleumdet* |	*slandered*	0.4
VBN	→	*verleumdet* |	*defamed*	0.2

Step 3: Reconstruct Synchronous Derivation



- Target-side: invert grammar projection (multiple rules? pick highest-scoring)

S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben*	*someone must have* VBN_2 NP_1	0.80
S	$\rightarrow$	*jemand mußte* X_1 X_2 *haben*	NP_1 *must have been* VBN_2 *by someone*	0.05

k-best Extraction

Objective Find the k-best synchronous derivations $d_1, d_2, \ldots d_k$

Well. . .

1. 1-best derivation is 1-best monolingual parse tree with best set of translations
2. 2-best and 3-best derivations are (in some order):
   (a) 1-best monolingual parse tree with second best set of translations, and
   (b) 2-best monolingual parse tree with best translations
3. 4-best derivation is one of. . .

k-best Extraction

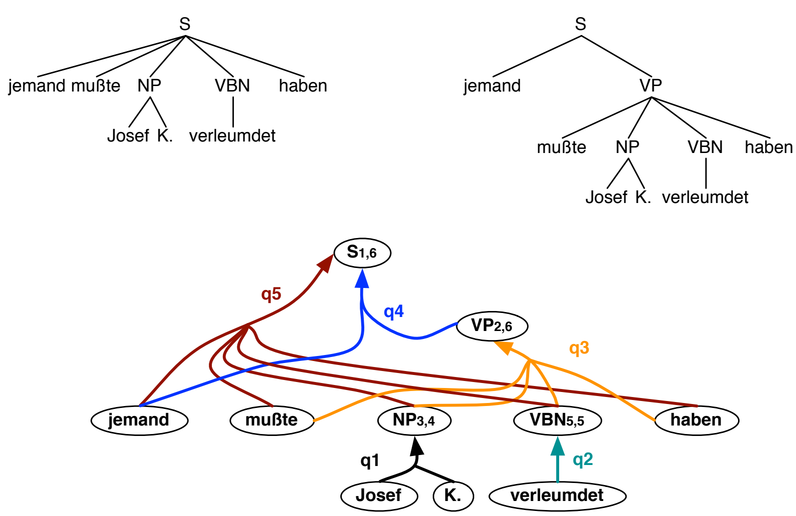
Objective Find the k-best synchronous derivations $d_1, d_2, \ldots d_k$

Well. . .

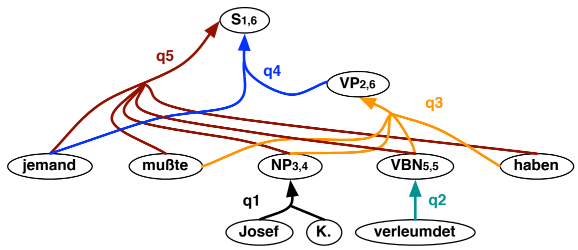
1. 1-best derivation is 1-best monolingual parse tree with best set of translations
2. 2-best and 3-best derivations are (in some order):
   (a) 1-best monolingual parse tree with second best set of translations, and
   (b) 2-best monolingual parse tree with best translations
3. 4-best derivation is one of. . .

We know part of the solution: how to get the k-best monolingual derivations (Huang and Chiang, 2005)

Digression: Parsing and Hypergraphs



Digression: Parsing and Hypergraphs



- Generalization of a graph: hyperedges connect two sets of vertices
- Terminology: vertices and hyperedges (nodes and arcs)
- A parse forest can be represented by a rooted, connected, labelled, directed, acyclic hypergraph (Klein and Manning, 2001)
- Vertices represent parsing states; hyperedges represent rule applications

Monolingual k-best Extraction

Huang and Chiang (2005) provide efficient algorithms for k-best extraction.

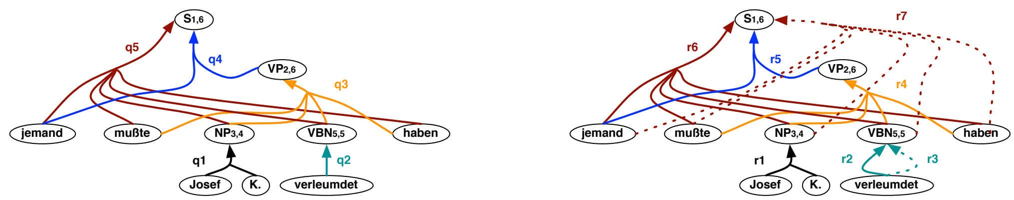
Objective Extract the k-best monolingual derivations $d_1, d_2, \ldots d_k$ from a weighted parse forest

Outline (alg. 3)

1. The 1-best subderivation for every vertex (and its incoming hyperedges) is known from the outset
2. Given the i-best derivation, the next best candidate along the same hyperedge is identical except for a substitution at a single incoming vertex
3. At the top vertex, generates candidates by recursively asking predecessors for next best subderivations.
4. Maintain priority queue of candidates at each vertex

Synchronous k-best Extraction

Replace hyperedges according to f' (invert grammar projection)



- The standard k-best extraction algorithm now gives the k-best synchronous derivations.
- The second hypergraph is sometimes called a "translation hypergraph".
- We'll call the first the "parse forest hypergraph" or the "parse hypergraph."

S2T Decoding (LM-) Summary

Objective Find the k-best synchronous derivations $d_1, d_2, \ldots d_k$

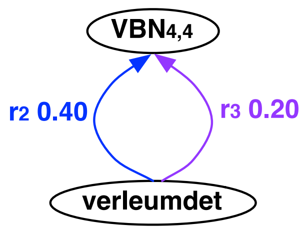
Solution

1. Project grammar
   Project weighted SCFG to unweighted CFG
   $f : G \rightarrow G'$ (many-to-one)
2. Parse
   Build parse hypergraph wrt G'
3. Invert projection
   Expand hypergraph by replacing hyperedges according to f'
4. Extract derivations
   Extract k-best derivations using Huang and Chiang's (2005) algorithm

LM Integration

Without LM k-best derivation is k-best path through translation hypergraph

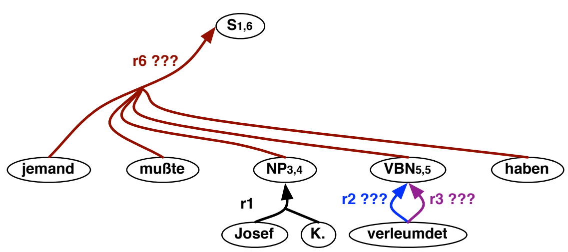
Optimal substructure



If global best path includes $\text{VBN}_{4,4}$ then best path must include hyperedge labelled r_2

LM Integration

Consider the two paths that include the hyperedge labelled r_6:



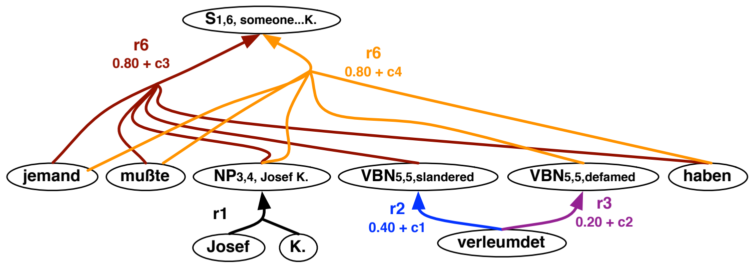
What's the best path through this hypergraph? For bi-gram LM we need to compute:

have [slandered] Josef $p(\text{have} \mid \langle s \rangle) \times p(\text{slandered} \mid \text{have}) \times p(\text{Josef} \mid \text{slandered}) \times \ldots$

have [defamed] Josef $p(\text{have} \mid \langle s \rangle) \times p(\text{defamed} \mid \text{have}) \times p(\text{Josef} \mid \text{defamed}) \times \ldots$

State Splitting?

Restore optimal substructure property by splitting states:



- Vertex labels include first and last words of translation.
- Hyperedges labelled with weights that incorporate LM costs.
- k-best derivation is k-best path.

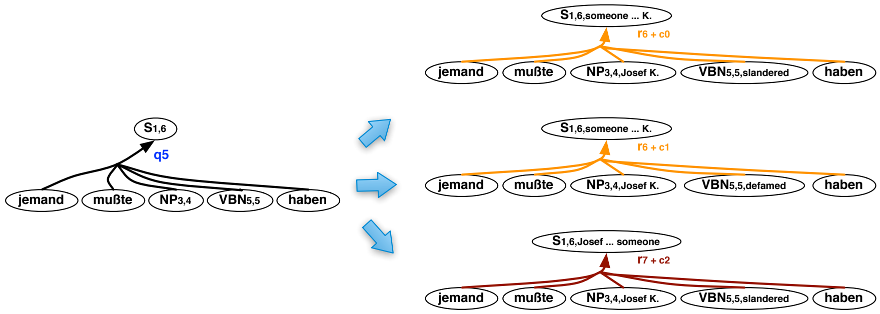
State Splitting?

Objective Find the k-best synchronous derivations $d_1, d_2, \ldots d_k$

Potential Solution

1. Project grammar
   Project weighted SCFG to weighted CFG $f : G \to G'$
2. Parse
   Build parse hypergraph wrt G'
3. Invert projection + **split states**
   Expand hypergraph by replacing hyperedges according to f'. **During replacement, split states and add LM costs**
4. Extract derivations
   Extract k-best derivations (Huang and Chiang, 2005)

State Splitting?



- Pick a search vertex for $\text{NP}_{3,4}$ from the set { $\text{NP}_{3,4,\text{Josef K.}}$ }
- Pick a search vertex for $\text{VBN}_{5,5}$ from the set { $\text{NP}_{5,5,\text{slandered}}$, $\text{NP}_{5,5,\text{defamed}}$ }
- Pick a synchronous rule from the set $f'(q_5) = \{r_6, r_7\}$ (i.e. pick a target-side)

The full set is generated by taking the Cartesian product of these three sets.

The Search Hypergraph is Too Large. . .

The parse hypergraph has $O(n^3)$ space constraints (assuming certain grammar properties. . .)

With a m-gram LM the search hypergraph is *much* larger:

	Vertices	Hyperedges
Parse	$O(n^2\|C\|)$	$O(n^3\|G\|)$
Search	$O(n^2\|C\|\|T\|^{2(m-1)})$	$O(n^3\|G\|\|T\|^{2A(m-1)})$

C is the set of target non-terminals
T is the set of target-side terminals
A is the maximum rule arity

n is the input sentence length
m is the order of the LM

Heuristic Search

- In practice, only part of the search hypergraph can be explored.
- During search, a partial search hypergraph is generated in topological order.
- Three main strategies for reducing search space:

Parse forest pruning Avoid splitting some parse forest hyperedges by pre-pruning the forest (methods can be exact or inexact).

Heuristic best-first splitting e.g. cube pruning. Use a splitting algorithm that finds expanded hyperedges in approximately best-first order.

Beam search Bin vertices according to source word span and category. Keep only the highest-scoring vertices for use later in the search.

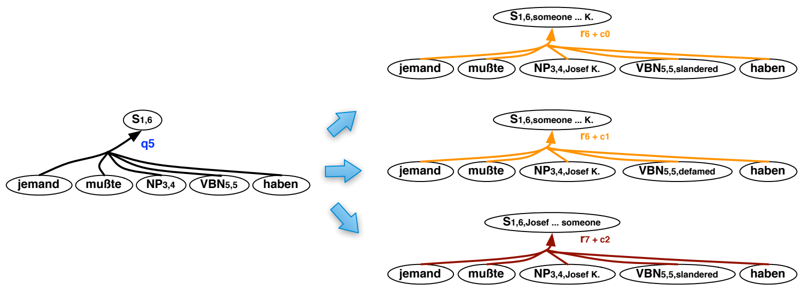
Strategy 1: Parse Forest Pruning

- If parse forest is constructed in full prior to search then dead-ends can be pruned away.
- State splitting can be restricted to a small subset of promising hyperedges.
  - Moses ranks hyperedges according to -LM rule cost plus sums of incoming +LM vertex costs.
- Monolingual forest pruning methods (Inside-outside estimates, see e.g. Charniak and Johnson (2005)).

(Forest pruning methods haven't been widely explored in the MT literature.)

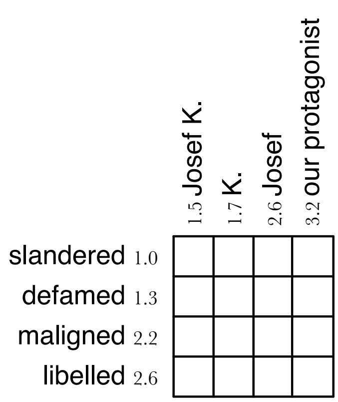
Strategy 2: Heuristic Best-First State Splitting

- For every hyperedge in the parse hypergraph, there can be very many corresponding hyperedges in the search hypergraph.



- Cube pruning (Chiang, 2007) is most widely-used approximate algorithm but see Heafield et al. (2013) for a faster alternative.

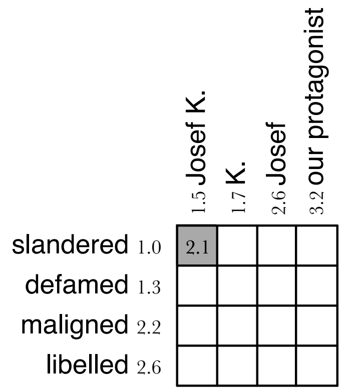
Cube Pruning



Arrange all the choices in a “cube”

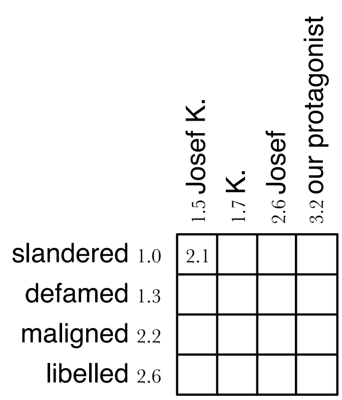
(here: a square, generally an orthotope, also called a hyperrectangle)

Create the First Hyperedge



- Hyperedges created in cube: (0,0)

"Pop" Hyperedge



- Hyperedges created in cube: ϵ
- Hyperedges popped: (0,0)

Create Neighboring Hyperedges

	1.5 Josef K.	1.7 K.	2.6 Josef	3.2 our protagonist
slandered 1.0	2.1	2.5		
defamed 1.3	2.7			
maligned 2.2				
libelled 2.6				

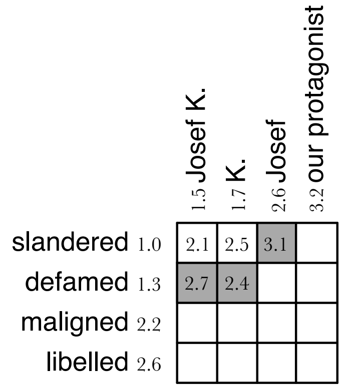
- Hyperedges created in cube: (0,1), (1,0)
- Hyperedges popped: (0,0)

Pop Best Hyperedge

	1.5 Josef K.	1.7 K.	2.6 Josef	3.2 our protagonist
slandered 1.0	2.1	2.5		
defamed 1.3	2.7			
maligned 2.2				
libelled 2.6				

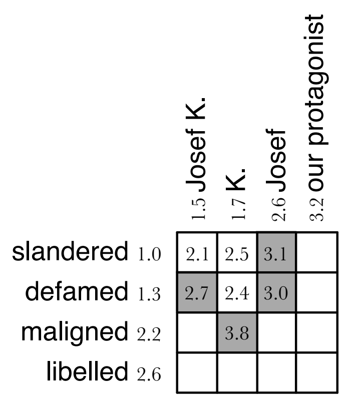
- Hyperedges created in cube: (0,1)
- Hyperedges popped: (0,0), (1,0)

Create Neighboring Hyperedges



- Hyperedges created in cube: (0,1), (1,1), (2,0)
- Hyperedges popped: (0,0), (1,0)

More of the Same



- Hyperedges created in cube: (0,1), (1,2), (2,1), (2,0)
- Hyperedges popped: (0,0), (1,0), (1,1)

Queue of Cubes

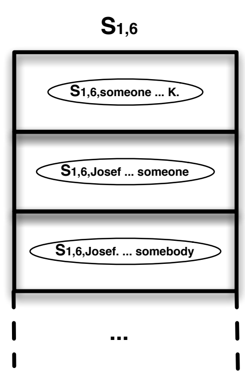
- Many parse hyperedges for any given span
- Each of them will have a cube
- We can create a queue of cubes

⇒ Always pop off the most promising hyperedge, regardless of cube

- May have separate queues for different target constituent labels

Strategy 3: Beam search

- Bin vertices according to source word span and category.
- Keep only the highest-scoring vertices for use later in the search.



Putting it All Together: The S2T Decoding Algorithm in Moses

Objective Find the k-best synchronous derivations $d_1, d_2, \ldots d_k$

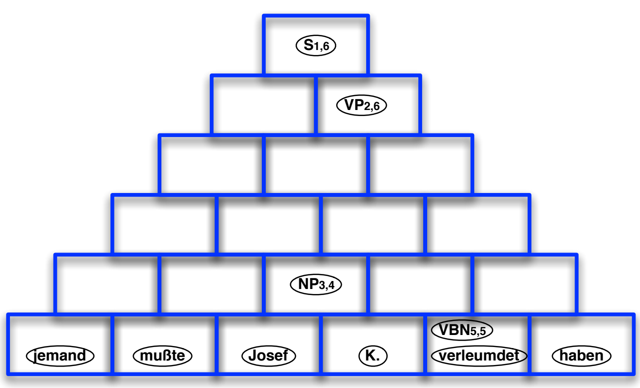
Outline

1. Project grammar
   Project weighted SCFG to weighted CFG $f : G \rightarrow G'$

2. Interleaved parse + search
   Span-by-span, build parse hypergraph wrt G' and build partial search hypergraph

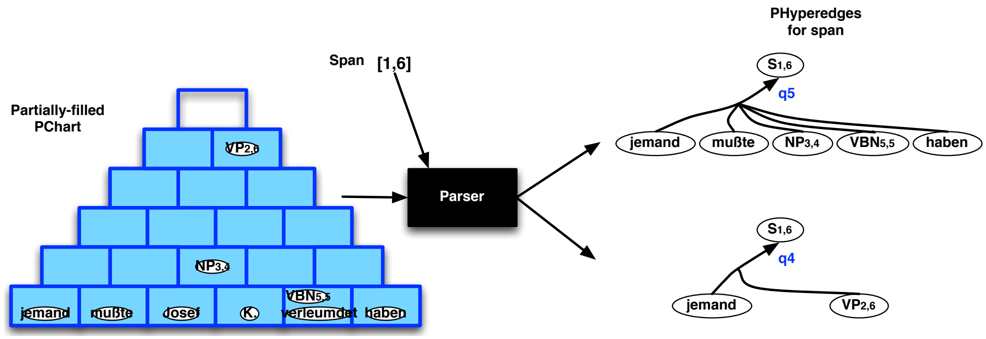
3. Extract derivations
   Extract k-best derivations (Huang and Chiang, 2005)

Decoding: Components



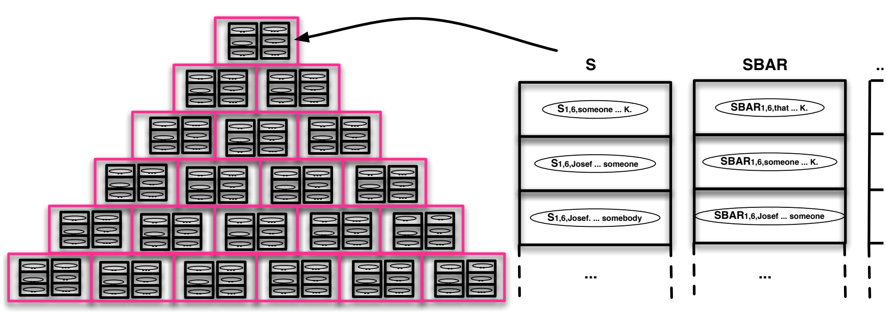
- Vertices of the parse hypergraph are stored in a chart (includes input sentence)
- Hyperedges are enumerated but not stored in chart
- Terminology: PChart, PVertex, PHyperedge

Decoding: Components



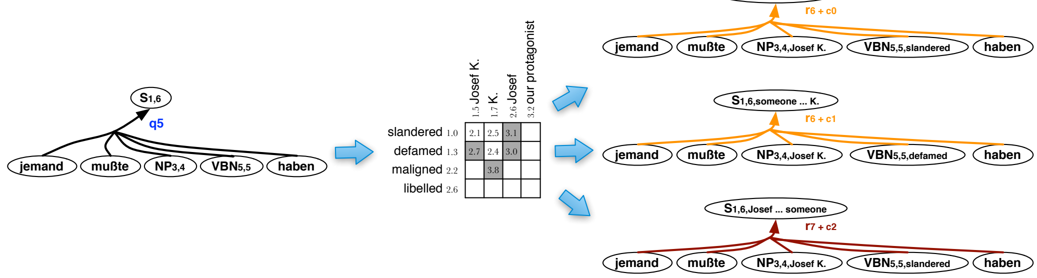
- Parser generates PHyperedges for given span of PChart
- Parser has access to partially-completed PChart
- For now, the parser is a black-box component but we'll return to parsing. . .

Decoding: Components



- Vertices of the search hypergraph are stored in a chart (includes input sentence)
- Vertices are stored in stacks (one per span + category), which are sorted
- Hyperedges are stored (unlike in PChart)
- Terminology: SChart, SVertex, SHyperedge

Decoding: Components

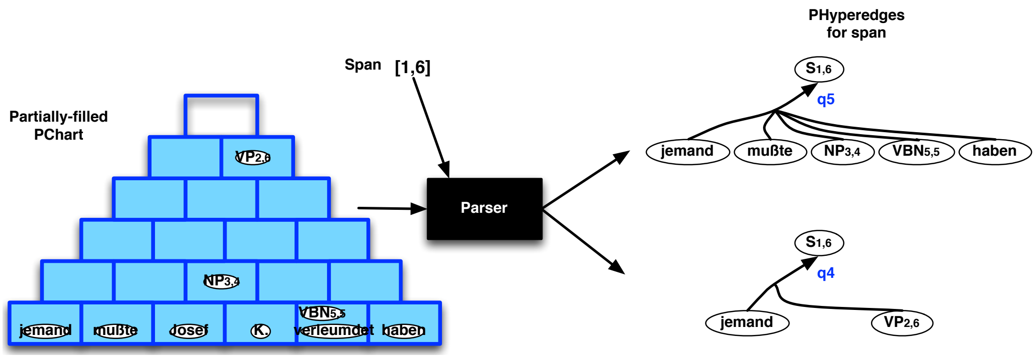


- Cube pruning algorithm (or similar) produces SHyperedges from PHyperedges
- A single SVertex can be produced multiple times so must check for this ('recombination')

The Moses S2T Decoding Algorithm

```
initialize PChart and SChart by adding vertices for input words
for each span (in parser-defined order) do
    p-hyperedges = ForestPrune(parser.EnumerateHyperedges(span, p-chart), s-chart)
    for all p-hyperedges do
        create a cube for it
        create first s-hyperedge in cube
        place cube in queue
    end for
    for specified number of pops do
        pop off best s-hyperedge of any cube in queue
        add it to a category-specific buffer
        create its neighbors
    end for
    for category do
        recombine s-hyperedges from buffer and move into s-chart stack
        sort stack
    end for
end for
```

Parsing for S2T Decoding



- Parser's job is to enumerate PHyperedges, span-by-span.
- Parser has access to partially-filled PChart.

Parsing for S2T Decoding

- Can we just use CYK / CYK+ / Earley?
  - All require binarization (implicit or explicit).
  - Wasn't a problem for Viterbi -LM case.
- **Idea 1** Binarize G'
  - Binary normal forms exist for monolingual CFG grammars.
  - *But* we still need to know the synchronous rules for +LM search.
- **Idea 2** Binarize G before projection to CFG
  - Binarization impossible for some SCFG rules with rank ≥ 4
  - Not necessarily a problem: non-binarizable cases are rare in word-aligned translation data (Zhang et al., 2006)
  - But tricky in practice: how do we weight rules? And what about grammar inflation?

How to Avoid Binarization

- Hopkins and Langmead (2010) define a grammar property called scope:

Pattern	Scope	Pattern	Scope
a b c d e	0	a ◇ ◇ ◇ e	2
a ◇ c ◇ e	0	◇ b c d ◇	2
a ◇ ◇ d e	1	◇ ◇ c d ◇	3
◇ b c d e	1	◇ ◇ ◇ ◇ ◇	6

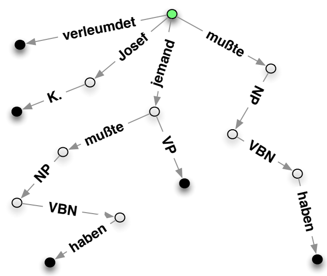
- They prove that a sentence of length n can be parsed with a scope k grammar in $O(nk)$ chart updates without binarization.
- They demonstrate empirically that reducing a GHKM grammar to scope-3 by pruning does not harm translation quality compared to synchronous binarization (and pruning is much simpler).
- Chung et al. (2011) perform similar comparison and achieve same result.

Specialized Parsing Algorithms

- CYK+ and Earley are popular choices for S2T decoding.
- But storing large numbers of dotted rules is problematic in practice (Chung et al. 2011 find scope-3 slower than binarized grammar with Earley parser, which they attribute to dotted rule storage).
- Several parsing algorithms have been designed specifically for synchronous translation grammars: DeNero et al. (2009), Hopkins and Langmead (2010), Sennrich (2014).
- We use Sennrich (2014)'s recursive variant of CYK+:
  - Good performance on WMT-scale task: fast, low-memory overhead
  - Simpler than CYK+ and alternatives
  - No dotted rule storage

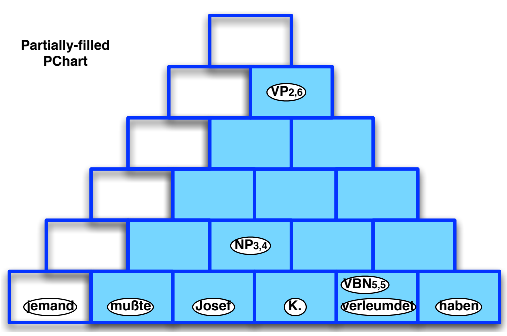
Parsing for S2T Decoding (Moses-style)

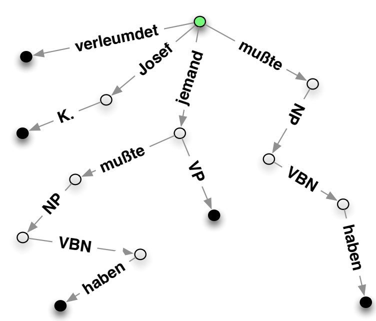
q_1:	NP	$\rightarrow$	*Josef K.*
q_2:	VBN	$\rightarrow$	*verleumdet*
q_3:	VP	$\rightarrow$	*mußte* NP VBN *haben*
q_4:	S	$\rightarrow$	*jemand* VP
q_5:	S	$\rightarrow$	*jemand mußte* NP VBN *haben*



- Projected grammar G' is represented as a trie (sometimes called a prefix tree)
- Edges are labelled with terminals and non-terminals
- Labels along path (from root) represent prefix of rule RHS
- Vertices in black are associated with group of rules from G (sub-grouped by rule LHS)

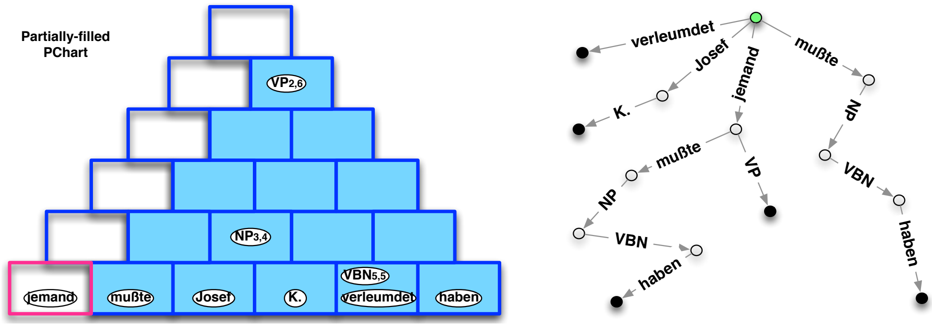
Parsing for S2T Decoding - Example





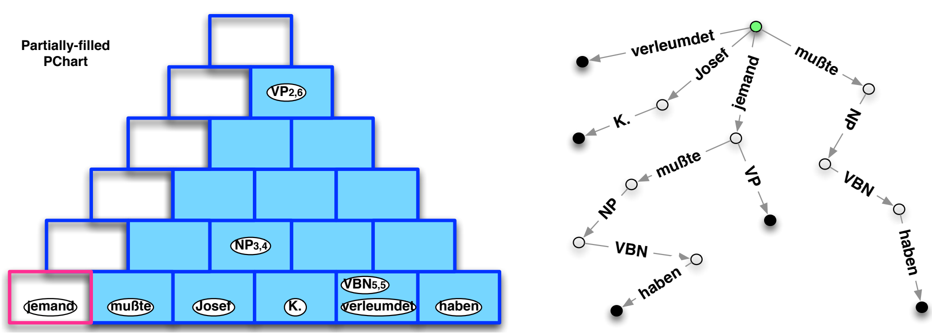
- Sennrich (2014)'s parsing algorithm visits cells in right-to-left, depth-first order.
- We consider situation where all of PChart filled except for left-most diagonal.
- Recall that PVertices are stored, but PHyperedges are not.

Parsing for S2T Decoding - Example



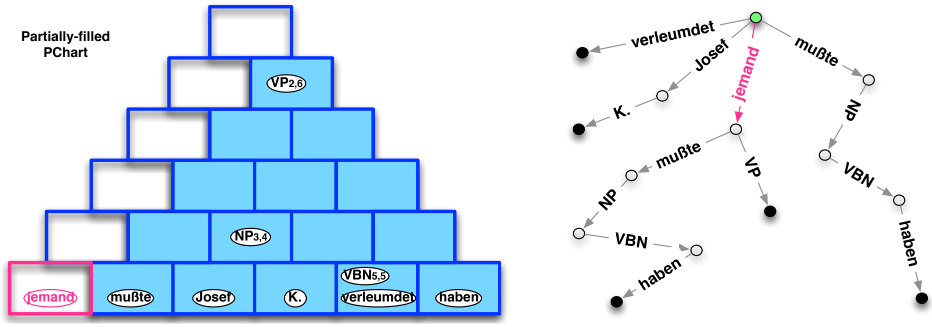
- Tail prefix: []
- Recursion level: 0

Parsing for S2T Decoding - Example



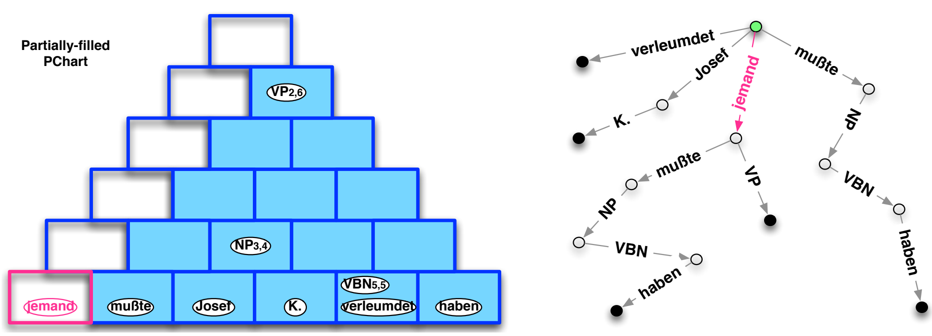
- Tail prefix: []
- Recursion level: 0
- Look for edge labelled ‘jemand’ at root node

Parsing for S2T Decoding - Example



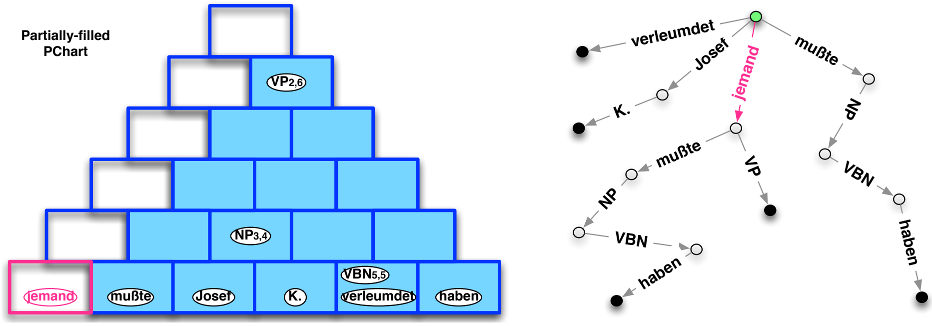
- Tail prefix: $[\text{jemand}_{1,1}]$
- Recursion level: 0
- Look for edge labelled 'jemand' at root node - found

Parsing for S2T Decoding - Example



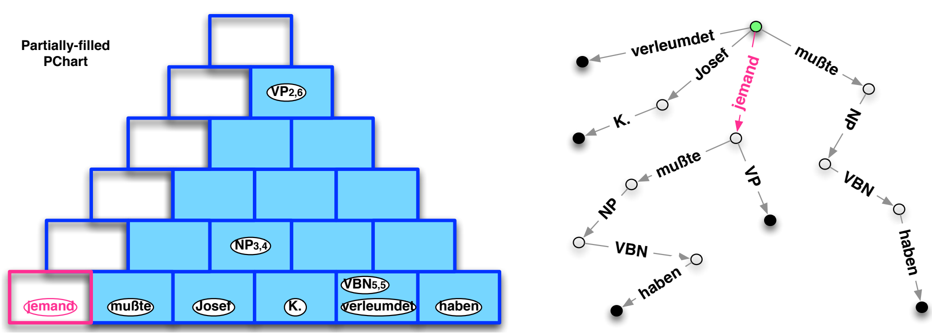
- Tail prefix: $[\text{jemand}_{1,1}]$
- Recursion level: 0
- Check for rules at current node - none

Parsing for S2T Decoding - Example



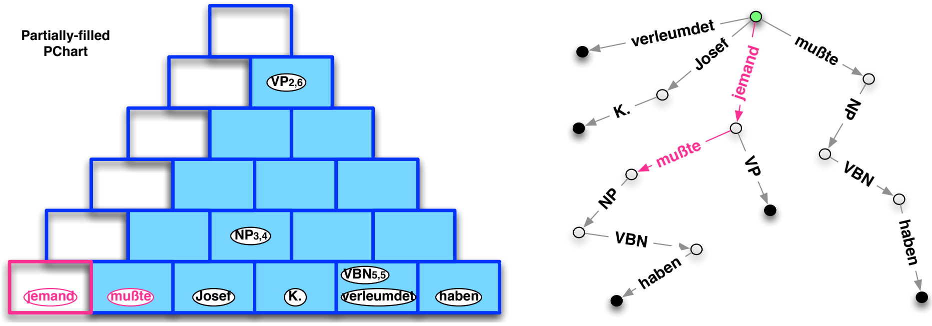
- Tail prefix: $[\text{jemand}_{1,1}]$
- Recursion level: 0
- Now visit each cell along previous diagonal (recursive step)

Parsing for S2T Decoding - Example



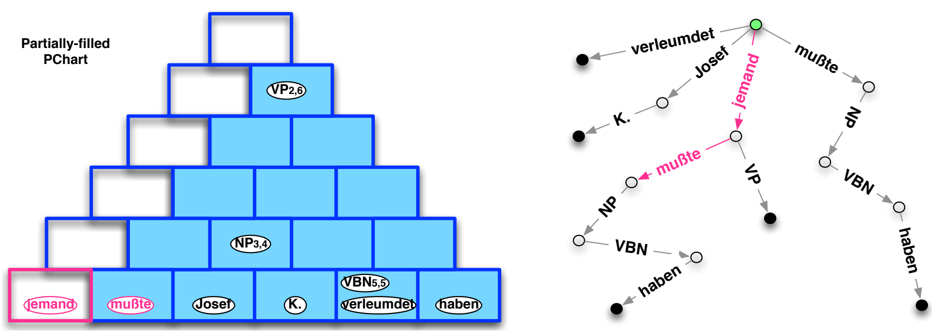
- Tail prefix: $[\text{jemand}_{1,1}]$
- Recursion level: 1
- Look for edge labelled ‘mußte’ at current node

Parsing for S2T Decoding - Example



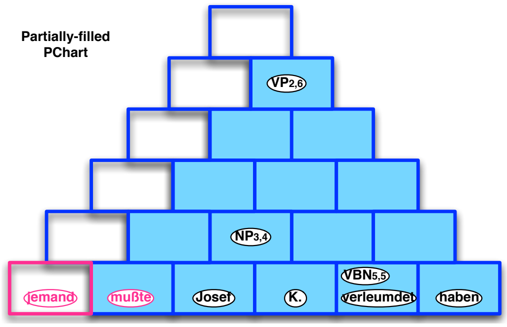
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}]$
- Recursion level: 1
- Look for edge labelled ‘mußte’ at current node - found

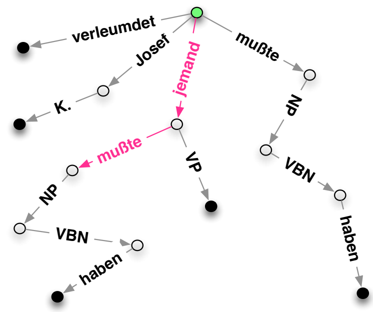
Parsing for S2T Decoding - Example



- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}]$
- Recursion level: 1
- Now visit each cell along previous diagonal

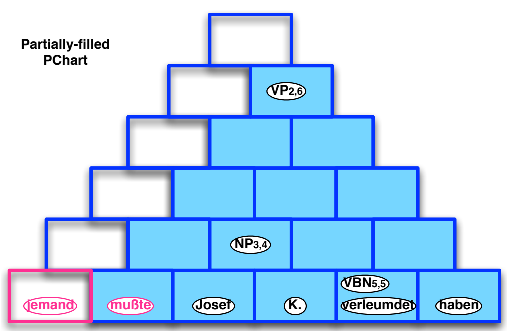
Parsing for S2T Decoding - Example

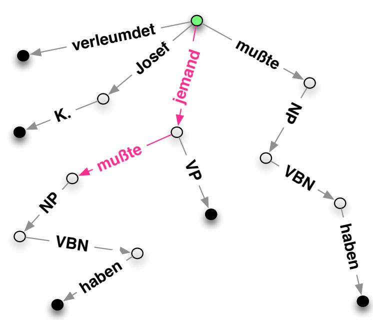




- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}]$
- Recursion level: 2
- Look for edge labelled 'Josef' at current node

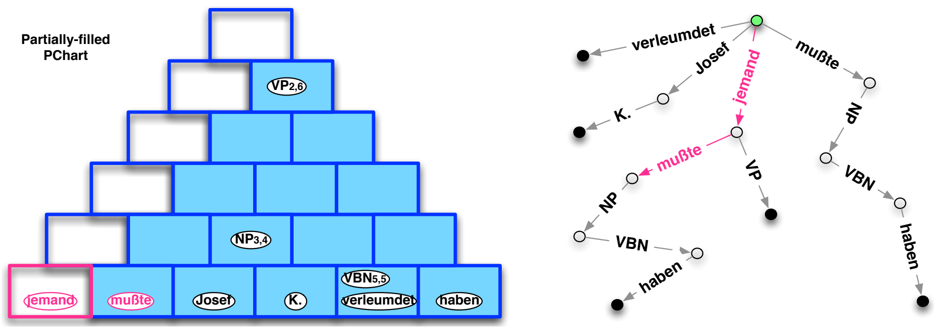
Parsing for S2T Decoding - Example





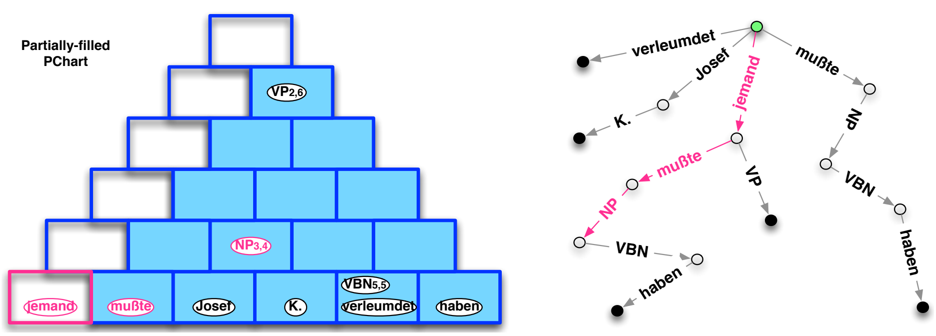
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}]$
- Recursion level: 2
- Look for edge labelled 'Josef' at current node - not found

Parsing for S2T Decoding - Example



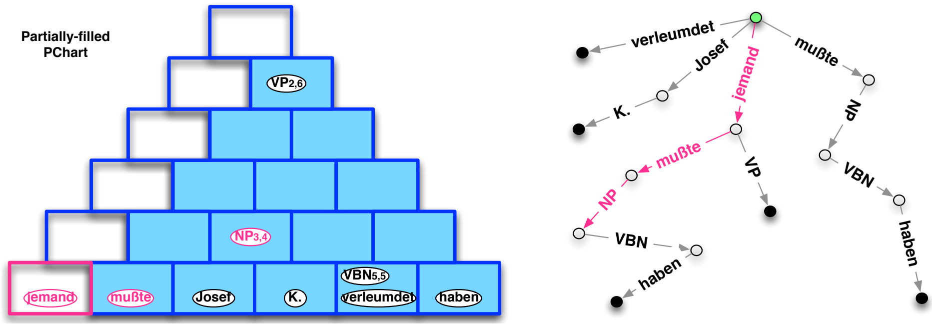
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}]$
- Recursion level: 2
- Look for edge labelled ‘NP’ at current node

Parsing for S2T Decoding - Example



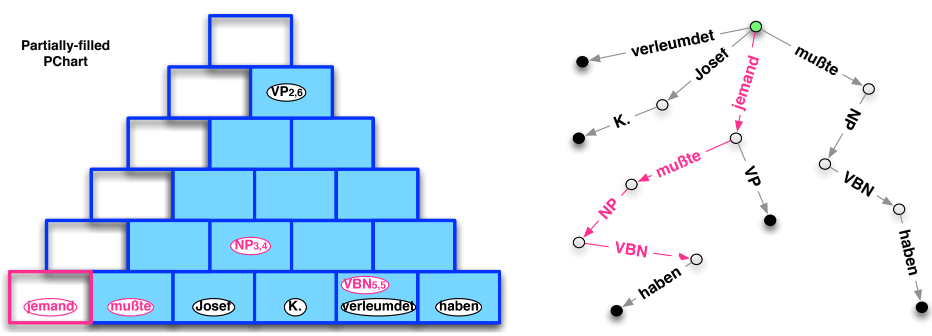
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}]$
- Recursion level: 2
- Look for edge labelled ‘NP’ at current node - found

Parsing for S2T Decoding - Example



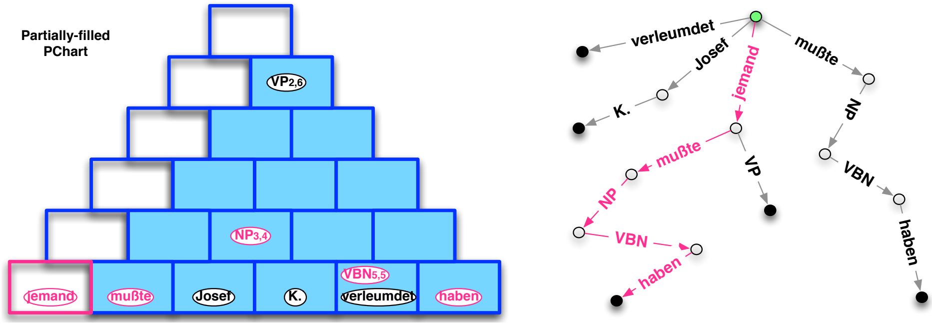
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}]$
- Recursion level: 3
- And so on. . .

Parsing for S2T Decoding - Example



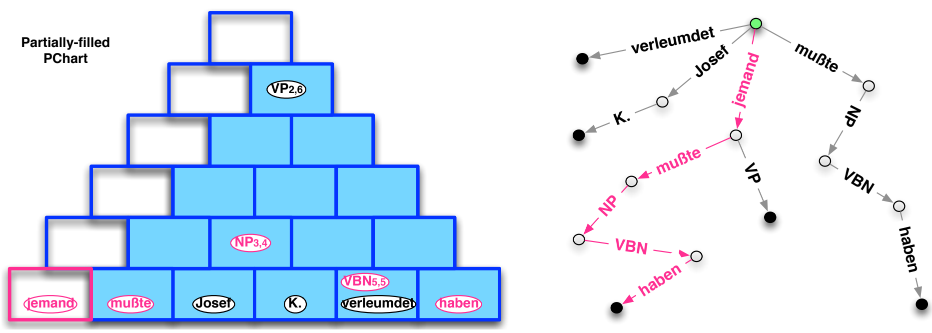
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}, \text{VBN}_{5,5}]$
- Recursion level: 3
- And so on. . .

Parsing for S2T Decoding - Example



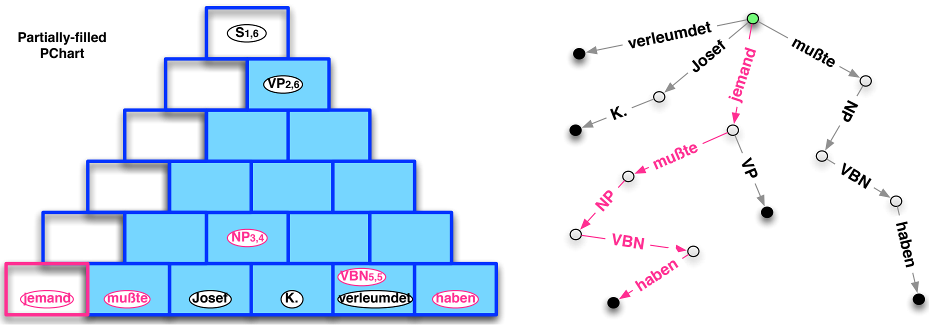
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}, \text{VBN}_{5,5}, \text{haben}_{6,6}]$
- Recursion level: 4
- And so on. . .

Parsing for S2T Decoding - Example



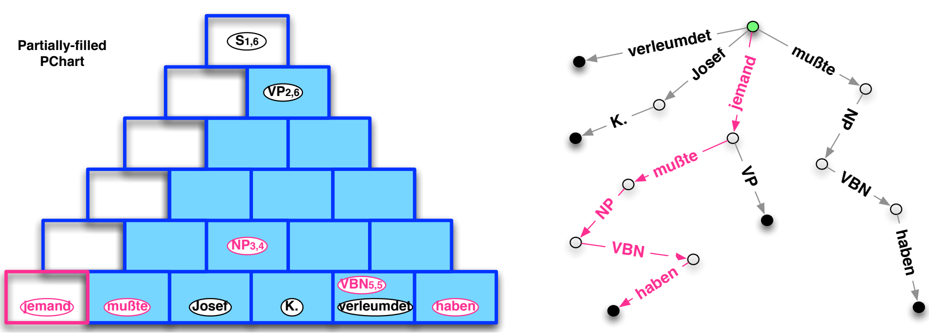
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}, \text{VBN}_{5,5}, \text{haben}_{6,6}]$
- Recursion level: 4
- At this point we add a PVertex for each LHS from trie node's rule group

Parsing for S2T Decoding - Example



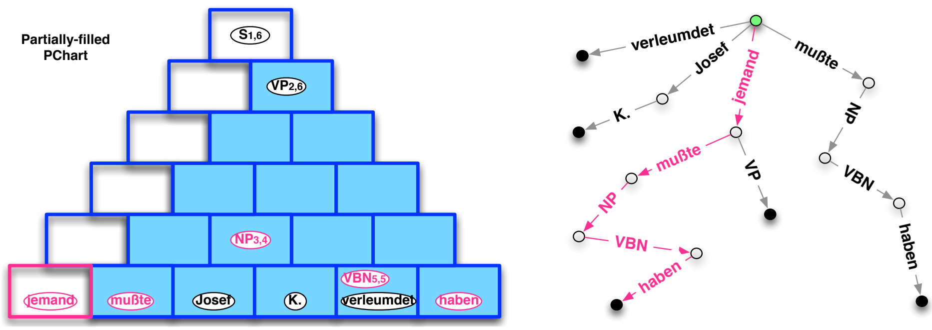
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}, \text{VBN}_{5,5}, \text{haben}_{6,6}]$
- Recursion level: 4
- At this point we add a PVertex for each LHS from trie node's rule group

Parsing for S2T Decoding - Example



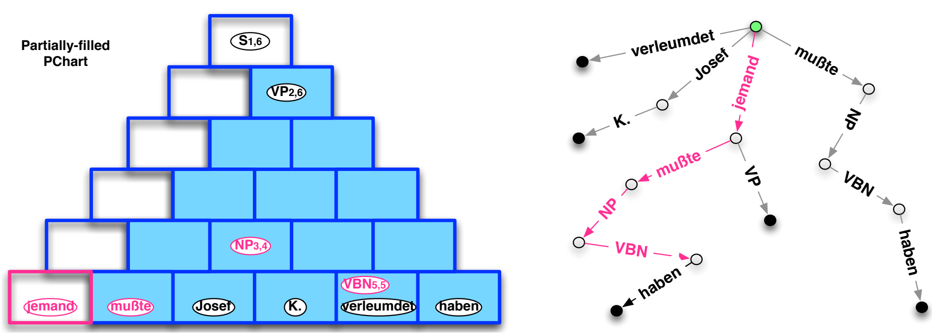
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}, \text{VBN}_{5,5}, \text{haben}_{6,6}]$
- Recursion level: 4
- Together the PVertex and tail prefix constitute a complete PHyperedge.

Parsing for S2T Decoding - Example



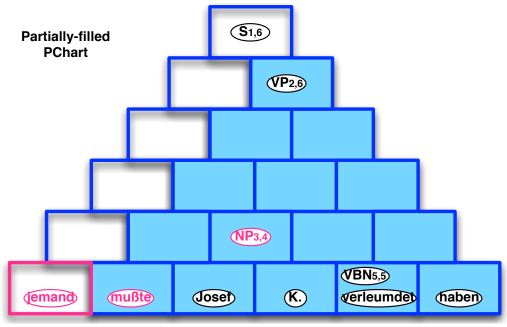
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}, \text{VBN}_{5,5}, \text{haben}_{6,6}]$
- Recursion level: 4
- Reached end of sentence, so now the recursion stack unwinds

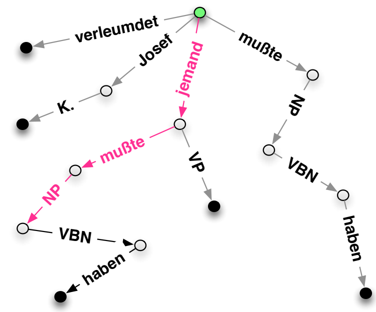
Parsing for S2T Decoding - Example



- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}, \text{VBN}_{5,5}]$
- Recursion level: 3
- The recursion stack unwinds. . .

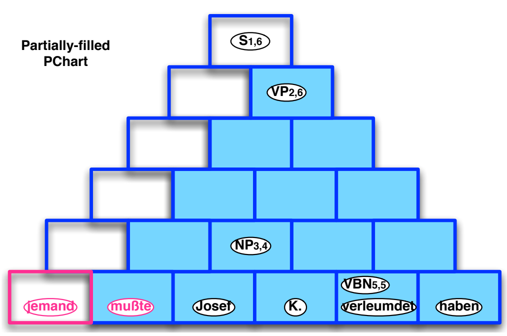
Parsing for S2T Decoding - Example

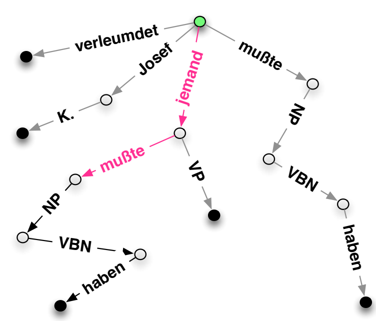




- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}, \text{NP}_{3,4}]$
- Recursion level: 2
- The recursion stack unwinds. . .

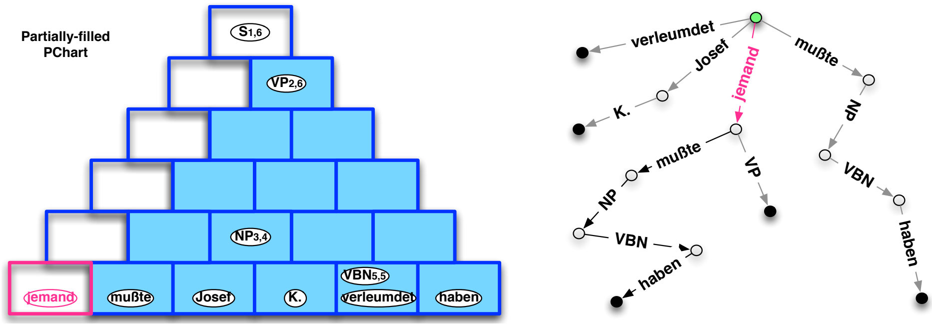
Parsing for S2T Decoding - Example





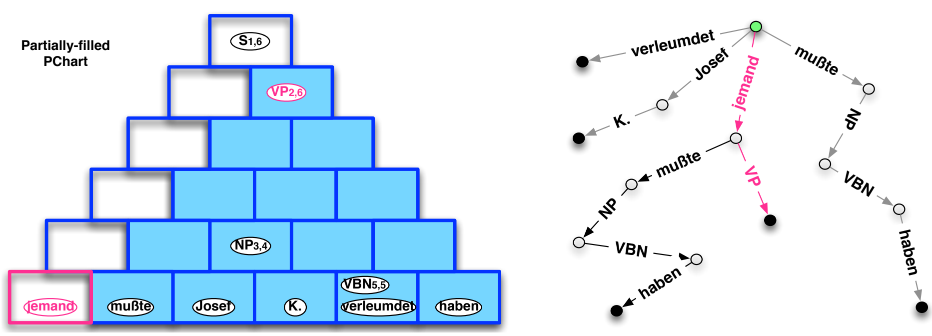
- Tail prefix: $[\text{jemand}_{1,1}, \text{mußte}_{2,2}]$
- Recursion level: 1
- The parser continues trying to extend the tail. . .

Parsing for S2T Decoding - Example



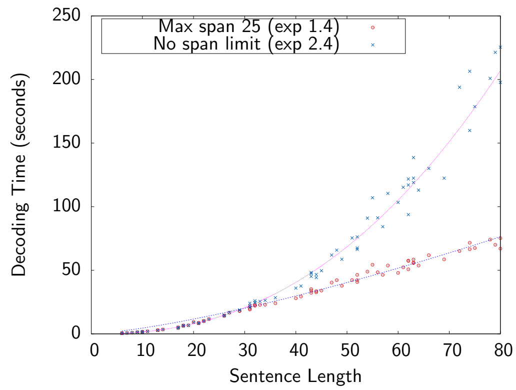
- Tail prefix: $[\text{jemand}_{1,1}]$
- Recursion level: 1
- The parser continues trying to extend the tail. . .

Parsing for S2T Decoding - Example



- Tail prefix: $[\text{jemand}_{1,1}, \text{VP}_{2,6}]$
- Recursion level: 1
- PVertex $\text{S}_{1,6}$ has already been added, but new tail means new PHyperedge

Decoding Performance in Practice



- S2T Moses system trained using all English-German data from WMT14
- Span limit can be used to reduce decoding time (limit is typically 10-15 for Hiero; can be higher or unlimited for S2T)

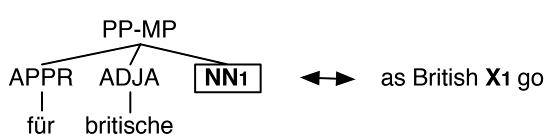
String-to-Tree Decoding - Summary

- Input sentence is a string.
- Decoding algorithm based on monolingual parsing.
- Hiero decoding is special-case of S2T decoding.
- To integrate a m-gram LM, the parse forest hypergraph is expanded to a (much-larger) search hypergraph.
- Heavy pruning is required in practice.

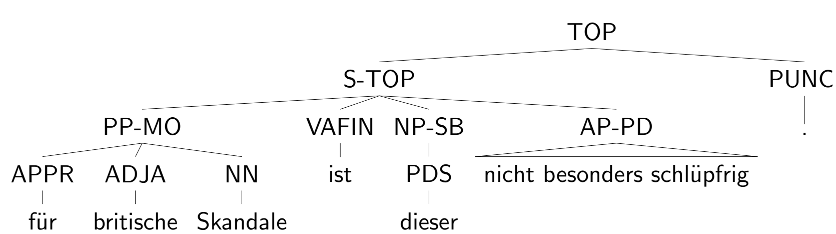
Tree-to-String Decoding

Reminder

- Translation rules are STSG rules with source-side syntax



- Input is parse tree



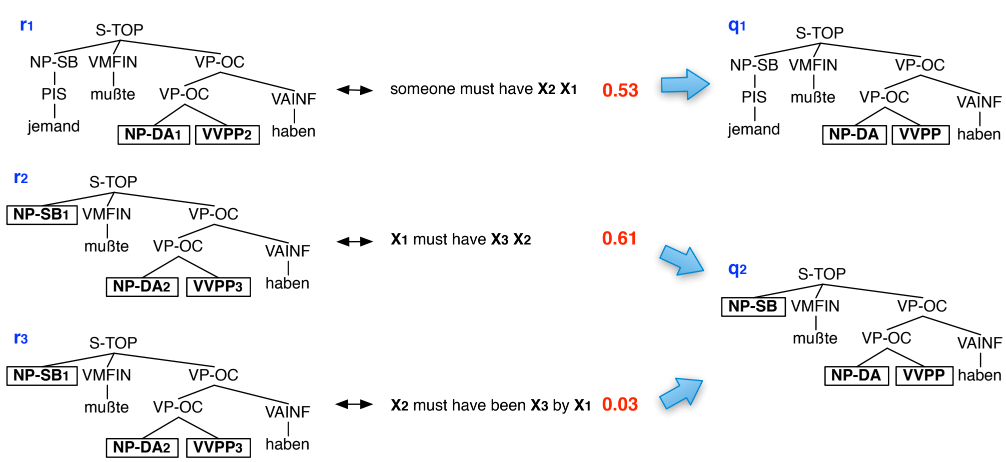
Outline

Objective Find the k-best synchronous derivations $d_1, d_2, \ldots d_k$

Outline

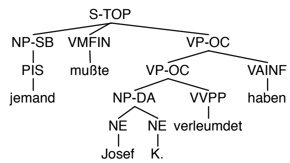
1. Project grammar
   Project weighted STSG to unweighted TSG $f : G \rightarrow G'$
2. Match rules
   Find rules from G' that match input tree, record in match hypergraph
3. Search
   In post-order traversal of match hypergraph, build partial search hypergraph
4. Extract derivations
   Extract k-best derivations (Huang and Chiang, 2005)

Step 1: Project Grammar



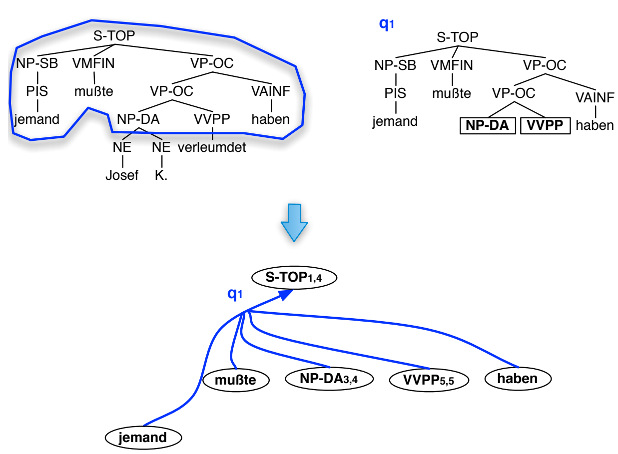
- Take source-side of rule, ignore weights.

Step 2: Match Rules, Build Match Hypergraph



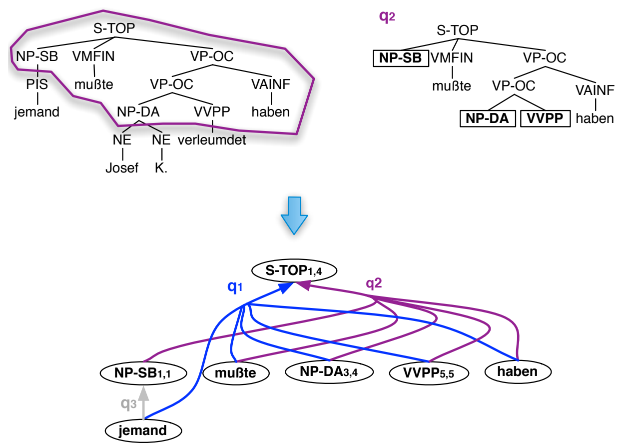
- Look for rules that match input tree

Step 2: Match Rules, Build Match Hypergraph



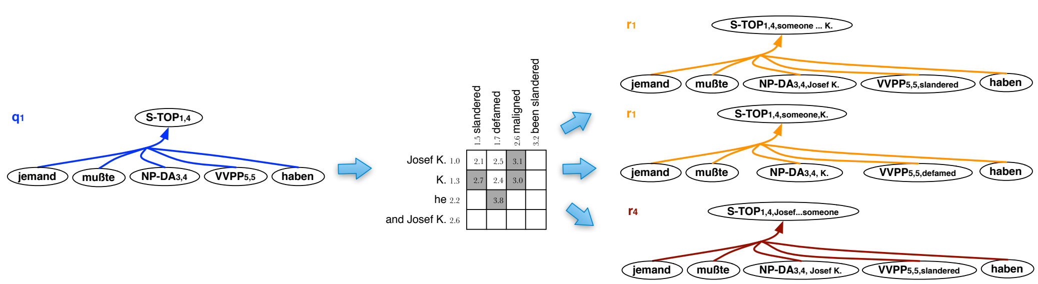
- For each matching rule, add hyperedge to match hypergraph

Step 2: Match Rules, Build Match Hypergraph



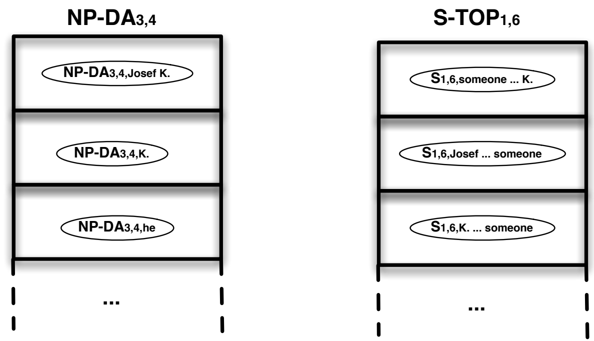
- Match hypergraph encodes forest of possible derivation trees from G'

Step 3: Build Partial Search Hypergraph



- Cube pruning algorithm produces SHyperedges from MHyperedges
- Translations not necessarily constituents (unlike S2T)

Step 3: Build Partial Search Hypergraph



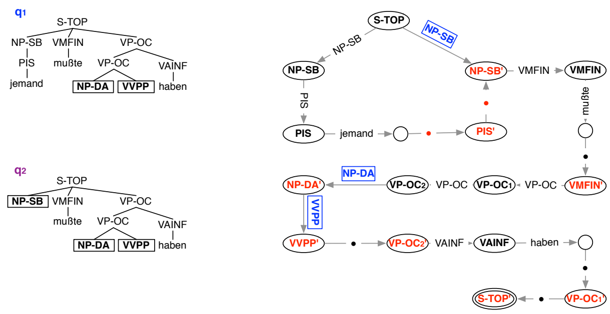
- Vertices are stored in stacks, one per input tree node

The T2S Decoding Algorithm

build match hypergraph by matching grammar rules to input tree
for each m-vertex (post-order) **do**
    for all incoming m-hyperedges **do**
        create a cube for it
        create first s-hyperedge in cube
        place cube in queue
    end for
    for specified number of pops **do**
        pop off best s-hyperedge of any cube in queue
        add it to a buffer
        create its neighbors
    end for
    recombine s-hyperedges from buffer and move into stack
    sort and prune stack
end for

Rule Matching by DFA Intersection

- Rules are encoded as DFAs. Scheme here is from Matthews et al. (2014)
- Input tree encoded in same way.
- Standard DFA intersection algorithm produces rule match hypergraph.



Tree-to-String Decoding - Summary

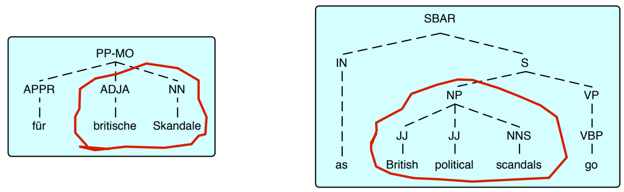
- Input sentence is a parse tree.
- Tree constrains rule choice: much smaller search space than S2T
- Decoding algorithm based on rule matching with LM integration.
- LM integration identical to S2T.

A Sketch of Tree-to-Tree Decoding

- STSG with tree input.
- T2T decoding is combination of S2T and T2S:
  - Search state expanded to include target-side category
  - Rule matching used to select rules; further constrained by target categories
  - Multiple category-specific stacks per input tree node
  - LM integration identical to S2T / T2S.
- Exact T2T not widely used in practice due to syntactic divergence.

“Fuzzy” Syntax

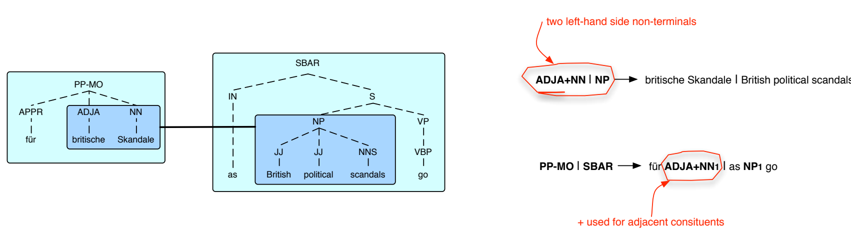
- In a nutshell: move syntax out of grammar and into feature functions
  - Syntax becomes a soft constraint
  - Motivated by syntactic divergence problem in tree-to-tree model



- “Learning to Translate with Source and Target Syntax” (Chiang, 2010)
  - Zhang et al (2011) use fuzzy syntax on source-side of string-to-tree model and explore alternative feature functions

“Fuzzy” Syntax

- Parse trees on both sides of training data
- Uses Hiero rule extraction but with SAMT-style labelling



- Only most frequent labelling kept (one-to-one correspondence with Hiero rules)

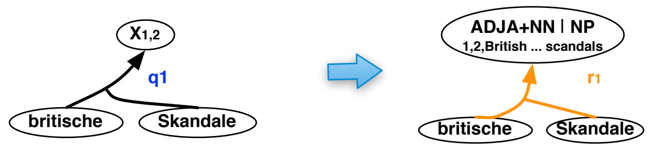
r1 **ADJA+NN | NP** → britische Skandale | British political scandals
r2 **PP-MO | SBAR** → für **ADJA+NN1** | as **NP1** go

⇔

q1 **X** → britische Skandale | British political scandals
q2 **X** → für **X1** | as **X1** go

“Fuzzy” Syntax

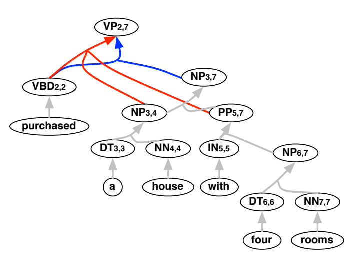
- Rule labels not used during parsing but retrieved for search



- Feature functions score substitutions
  - e.g. if a NP is rewritten as a ADJA+NN on source side then the feature $\texttt{subst}^{s}_{\texttt{NP}\rightarrow\texttt{ADJA+NN}}$ fires
- Tens of thousands of features
- Outperforms exact tree-to-tree (0.4 BLEU on Zh-En; 1.5 BLEU on Ar-En)

Forest-to-String

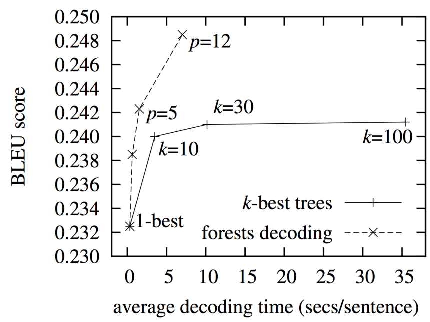
- Translation quality of T2S model depends on accuracy of 1-best (or k-best) parse tree(s) for input sentences
- Forest-to-string extends T2S by using (pruned) parse forest as input



- Algorithm is identical to T2S except for rule matching step
- “Forest-based Translation” (Mi et al., 2008)

Forest-to-String

- Using forest gives better speed-quality trade-off than using k-best trees

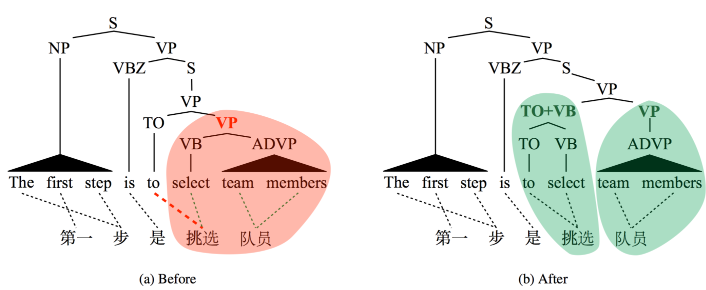


(Figure taken from Mi et al., 2008)

Tree Transformation

- Adapting training data for syntax-based MT is active area of research (tree binarization, label coarsening / refinement, word alignment edits)

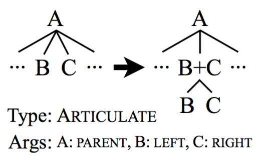
- "Transforming Trees to Improve Syntactic Convergence" (Burkett and Klein, 2012) proposes tree restructuring method to improve rule extraction:

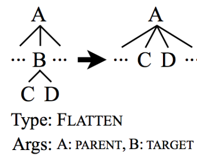


(Figure taken from Burkett and Klein, 2012)

Tree Transformation

- Defines six classes of transformation



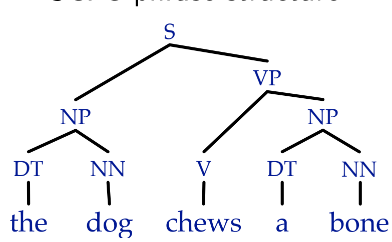


- Error-based learning method using GHKM frontier node count as metric
- Sequence of transformations learned from subset of training data then applied to full corpus
- Gain of 0.9 BLEU over baseline on Chinese to English; outperforms simple left and right binarization

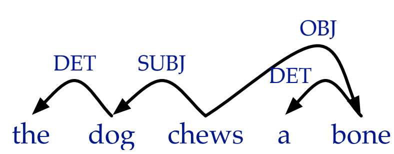
Dependency

A different view on syntax

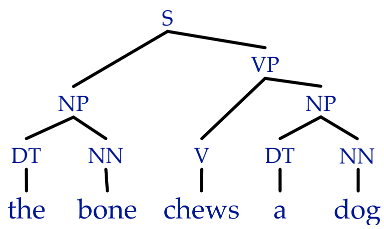
SCFG phrase structure

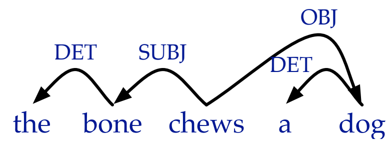


Syntactic dependency grammar



Phrase Structure is not Enough



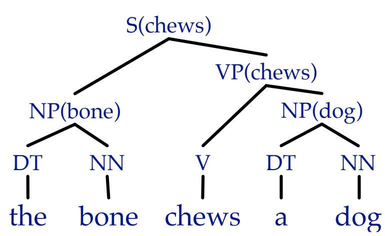


syntactically well-formed

semantically implausible

Dependency in SCFG

- Add head word to constituents

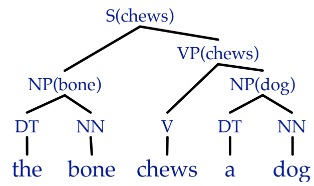


- Add mapping of head words to rules

$$\text{VP}(w_1) \rightarrow \text{V}(w_1)\ \text{NP}(w_2)$$

requires identification of head child

Semantic Plausibility

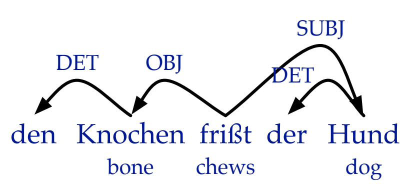


Score each lexical relationship

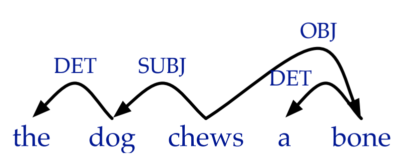
- Rule: VP(chews) → V(chews) NP(dogs)
  - Feature: VP(chews)→V-HEAD(chews) **OK**
  - Feature: VP(chews)→NP(dog) **BAD**
- Rule: S(chews) → NP(bone) VP(chews)
  - Feature: S(chews)→NP(bone) **BAD**
  - Feature: S(chews)→V-HEAD(chews) **OK**

Informed by Source

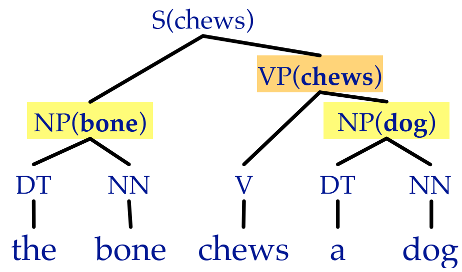
- Languages with case marking
  - different word order
  - same dependency relationships



- Give preference to translations that preserve dependency relationships



Verb Frames



- Check if full verb frame is properly filled
  - intransitive / transitive / ditransitive
  - not just binary relationships
  - appropriate type of subjects / objects
- However: tracking verb frame is not trivial

Towards Semantics

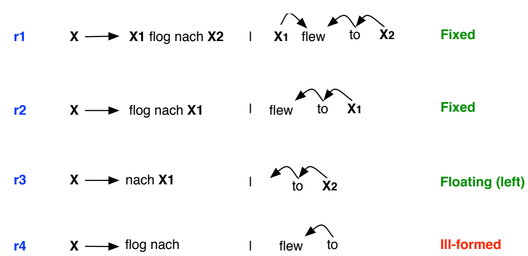
- Different syntax — same verb-noun semantic relationships
  - The bone is chewed by the dog.
  - The dog chews the bone.
  - The bone, the dog chews.
  - A dog chewed a bone.
- Even more abstract representations
  e.g., Abstract Meaning Representation (AMR):

```
(c / chew-01
  :arg0 (d / dog)
  :arg1 (b / bone))
```

- Generation of these types of representation open research problem

String-to-Dependency: Shen et al. (2008)

- Hiero rules but with unlabelled dependencies on target side
- Target-side allowed one head to which floating dependencies can attach



- "A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model" (Shen et al., 2008)

String-to-Dependency

- Decoding algorithm modified to combine dependency structures.
- Restriction to well-formed rules reduces grammar size from 140M to 26M rules (no significant effect on translation quality).
- Gains of 1.2 Bleu on Zh-En from addition of dependency LM (Markov model over dependency heads).

References

- *Parsing and Hypergraphs*
  Dan Klein and Christopher Manning. IWPT 2001.
- *What's in a Translation Rule?*
  Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. HLT-NAACL 2004.
- *A Hierarchical Phrase-based Model for Statistical Machine Translation*
  David Chiang. ACL 2005.
- *Better k-best Parsing*
  Liang Huang and David Chiang. IWPT 2005.
- *Syntax Augmented Machine Translation via Chart Parsing*
  Andreas Zollmann and Ashish Venugopal. WMT 2006.
- *Synchronous Binarization for Machine Translation*
  Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. NAACL 2006.
- *Hierarchical Phrase-Based Translation*
  David Chiang. Computational Linguistics 2007.

References

- *A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model*
  Libin Shen, Jinxi Xu, and Ralph Weischedel. ACL 2008.
- *Forest-Based Translation*
  Haitao Mi, Liang Huang, and Qun Liu. ACL 2008.
- *Efficient Parsing for Transducer Grammars*
  John DeNero, Mohit Bansal, Adam Pauls, and Dan Klein. NAACL 2009.
- *SCFG Decoding Without Binarization*
  Mark Hopkins and Greg Langmead. EMNLP 2010.
- *Learning to Translate with Source and Target Syntax*
  David Chiang, ACl 2010.
- *Issues Concerning Decoding with Synchronous Context-free Grammar*
  Tagyoung Chung, Licheng Fang, and Daniel Gildea. ACL 2011.
- *Transforming Trees to Improve Syntactic Convergence*
  David Burkett and Dan Klein. EMNLP 2012.

References

- *Grouping Language Model Boundary Words to Speed K-Best Extraction from Hypergraphs*
  Kenneth Heafield, Philipp Koehn, and Alon Lavie. NAACL 2013.

- *Tree Transduction Tools for cdec*
  Austin Matthews, Paul Baltescu, Phil Blunsom, Alon Lavie, Chris Dyer. PBML Vol 102. (2014)

- *A CYK+ Variant for SCFG Decoding Without a Dot Chart*
  Rico Sennrich. SSST 2014.