Natural Language Engineering **17** (3): 331–365. © Cambridge University Press 2010
doi:10.1017/S1351324910000240

Deterministic shift-reduce parsing for unification-based grammars

TAKASHI NINOMIYA[1], TAKUYA MATSUZAKI[2], NOBUYUKI SHIMIZU[3] and HIROSHI NAKAGAWA[3]

[1]*Graduate School of Science and Engineering, Ehime University,*
3 Bunkyo-cho, Matsuyama, Ehime 790-8577, Japan
e-mail: ninomiya@cs.ehime-u.ac.jp
[2]*Graduate School of Information Science and Technology, The University of Tokyo,*
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
e-mail: matuzaki@is.s.u-tokyo.ac.jp
[3]*Information Technology Center, The University of Tokyo,*
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
e-mail: shimizu@r.dl.itc.u-tokyo.ac.jp, n3@dl.itc.u-tokyo.ac.jp

(*Received 20 May 2009; revised 13 April 2010; accepted 20 August 2010; first published online 22 October 2010*)

Abstract

Many parsing techniques assume the use of a packed parse forest to enable efficient and accurate parsing. However, they suffer from an inherent problem that derives from the restriction of locality in the packed parse forest. Deterministic parsing is one solution that can achieve simple and fast parsing without the mechanisms of the packed parse forest by accurately choosing search paths. We propose new deterministic shift-reduce parsing and its variants for unification-based grammars. Deterministic parsing cannot simply be applied to unification-based grammar parsing, which often fails because of its hard constraints. Therefore, this is developed by using default unification, which almost always succeeds in unification by overwriting inconsistent constraints in grammars.

1 Introduction

Since the mid-1980s, unification-based grammars have been studied by many researchers as a device for providing precise and in-depth analyses based on linguistic theories, such as head-driven phrase structure grammar (HPSG) (Pollard and Sag 1994), lexical functional grammar (LFG) (Bresnan and Kaplan 1982) and combinatory categorial grammar (CCG) (Steedman 2000). As a numerical model for selecting the best parse tree, conditional random fields for a packed parse forest have been used for probabilistic unification-based grammars (Geman and Johnson 2002; Miyao and Tsujii 2002; Kaplan *et al.* 2004; Clark and Curran 2004b), and its parsing is empirically known to be fast and accurate (Clark and Curran 2004a; Miyao and Tsujii 2005; Ninomiya *et al.* 2006; Ninomiya *et al.* 2007). Both estimation and parsing with the packed parse forest, however, have an inherent problem that derives from the restriction of locality. Feature functions can only be

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

defined for local structures, which limit the parser's accuracy. This is because parsers segment parse trees into constituents and factor equivalent constituents into a single constituent (edge) in a chart. However, this problem has not been adequately studied in unification-based grammar parsing, where only a few studies exist, i.e., a method of sampling for estimating parameters (Malouf and van Noord 2004), and selective *n*-best unpacking (Zhang, Oepen and Carroll 2007).

This paper discusses our investigations into a transition-based parsing approach to unification-based grammars without the mechanisms of the packed parse forest. The transition-based parsing for context-free grammar (CFG) trees or dependencies has recently been studied (Ratnaparkhi 1997; Yamada and Matsumoto 2003; Collins and Roark 2004; Nivre and Scholz 2004; Sagae and Lavie 2005; Sagae and Lavie 2006; Zhang and Clark 2009) through approaches that are essentially based on deterministic parsing, in which global features can be easily defined for the entire structures of partial parse trees that have been built thus far in the process of parsing. The main advantage of this approach is that it can achieve simple and fast parsing by accurately choosing a search path with a large number of local and global features. Its current accuracy is competitive with that of state-of-the-art CFG parsers, and it is one of the most promising methodologies thus far. These techniques, however, cannot simply be applied to unification-based grammar parsing because it can fail as a result of its hard constraints in the grammar. Therefore, we propose deterministic parsing for unification-based grammars by using default unification, which almost always succeeds in unification by overwriting inconsistent constraints in the grammars such that the amount of overwriting is minimized. Because parsing failure is caused by unification failure of grammar constraints, default unification enables us to almost always succeed in parsing an entire sentence. Deterministic shift-reduce parsing itself is simple without the mechanism of the packed parse forest. This enables us to write global feature functions flexibly. We further pursue the best-first shift-reduce parsing for unification-based grammars by introducing nondeterminism in the same way as Sagae and Lavie (2006). The experiments revealed that deterministic parsing with default unification achieved a high degree of accuracy of 87.6 per cent for the Penn Treebank with gold part-of-speech (POS) tags. The best-first parsing with a beam search achieved the best accuracy of 87.0 per cent in settings with normal unification. Default unification further increased accuracy from 87.0 to 88.4 per cent. The experimental results with a very wide beam indicate that our deterministic parsing has the potential to reach the accuracy of state-of-the-art parsers due to its deterministic mechanism that offers flexibility in its feature design.

Section 2 describes the background and related works on unification-based grammar parsing and transition-based parsing. Section 3 explains unification-based grammars and Section 4 explains default unification. Shift-reduce parsing for unification-based grammars is presented in Section 5. Section 6 discusses our experiments and Section 7 concludes the paper.

2 Background and related work

Developing a unification-based grammar parser for parsing real-world text has been difficult for a long time because of the difficulty of developing grammars

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

and the absence of well-defined numerical models for selecting the best parse tree in unification-based grammars. The problem with grammar development was overcome by using a method of constraint ranking for manually tailored LFG grammars (Butt *et al.* 1999; Frank *et al.* 2001) and corpus-based methods, through which consistent and robust unification-based grammars could be acquired from a CFG-style annotated corpus (Hockenmaier and Steedman 2002; O'Donovan *et al.* 2004; Clark and Curran 2004b; Miyao, Ninomiya and Tsujii 2005). The problem with numerical models was solved by using a probabilistic model for unification-based grammars. Previous studies (Abney 1997; Johnson *et al.* 1999; Riezler *et al.* 2000; Geman and Johnson 2002; Miyao and Tsujii 2002; Toutanova *et al.* 2002; Kaplan *et al.* 2004; Malouf and van Noord 2004; Clark and Curran 2004b) defined a probabilistic model of unification-based grammars, including HPSG, LFG and CCG, as conditional random fields for parse trees, which provided the probabilities of discriminating a correct parse tree from candidate parse trees in a multi-class logistic regression (also known as, a log-linear or maximum entropy model) with many features for parse trees.

Conditional random fields for packed parse trees have been used successfully for probabilistic unification-based grammars (Geman and Johnson 2002; Miyao and Tsujii 2002; Kaplan *et al.* 2004; Clark and Curran 2004b), and their parsing has empirically been known to be fast and accurate, especially with supertagging (Clark and Curran 2004a; Ninomiya *et al.* 2006, 2007). Both estimation and parsing with the packed parse forest, however, raise an inherent problem that derives from the restriction of locality. Feature functions can only be defined for local structures, which limit the parser's accuracy. This is because parsers segment parse trees into constituents and factor equivalent constituents into a single constituent (edge) in a chart. However, this problem has not been adequately studied in unification-based grammar parsing, where only a few studies exist, i.e., a method of sampling for estimating parameters (Malouf and van Noord 2004), and selective *n*-best unpacking (Zhang *et al.* 2007).

In many studies on CFG parsing, several approaches, such as reranking (Collins 2000; Charniak and Johnson 2005; Huang 2008), transition-based parsing (Ratnaparkhi 1997; Yamada and Matsumoto 2003; Collins and Roark 2004; Nivre and Scholz 2004; Sagae and Lavie 2005, 2006; Zhang and Clark 2009), learning as search optimization (Daumé and Marcu 2005) and sampling methods (Nakagawa 2007) have been studied to solve the problem of locality. The reranking methods rerank the *n*-best unpacked parse trees obtained from a packed parse forest, but these kinds of methods inherently suffer from two drawbacks; first, the accuracy of reranking greatly depends on the precision of first-stage parsing, which only uses local features, and second, the correct answer can slip from the list of the *n*-best candidates; e.g., around 40 per cent of the correct parse trees did not exist in the 30-best candidates (Collins 2000). Huang (2008) proposed a fine-grained method of reranking, which tried to process the reranking and enumeration of *n*-best candidates simultaneously by extracting the *n*-best partial parse trees for each node in the packed parse forest. Huang's method enables approximate reranking of the packed parse forest with non-local features, but it is limited by

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

the beam search and the enumeration of partial parse trees instead of locality restriction.

The transition-based parsing, including incremental parsing with a perceptron (Collins and Roark 2004), shift-reduce parsing (Yamada and Matsumoto 2003; Sagae and Lavie 2005; Sagae and Lavie 2006; Zhang and Clark 2009) and dependency parsing (Nivre and Scholz 2004), has intensively been studied as an empirical method of parsing natural languages. It regards the parsing problem as a search problem of how to choose the parser's action in which the scores for choosing the parser's action are associated with the history of the parsing process, such as previous parsing actions and partial parse trees that have been built thus far. One advantage of this approach is that it can achieve simple and fast parsing by accurately choosing a search path. For example, Sagae and Lavie (2005) have shown that deterministic shift-reduce parsing can be highly efficient and accurate by using a large number of local and global features extracted from entire structures of unambiguous partial parse trees, which have been deterministically built during the parsing process. Likewise, parsing methods in recent studies on the transition-based parsing have assumed that each parsing state retains its own unambiguous partial parse trees to extract global features, and have not assumed a packed parse forest. They have introduced nondeterminism simply by processing parsing states in parallel (Sagae and Lavie 2006). This means that its hypothesis space becomes exponentially larger because the intermediate results are not shared among parsing states. However, this approach enables us to extract global features that can greatly contribute to choosing a better search path. Collins and Roark's (2004) incremental parsing and Zhang and Clark's (2009) shift-reduce parsing nondeterministically optimized their global discrimination models by using a perceptron. Their probabilistic models and search strategies were different from Sagae and Lavie's methods, but these were also based on 'deterministic parsing,' i.e., these did not use the packed parse forest, each parsing state retained its own parsing history and their parsing order was fixed from left to right. The transition-based parsing currently achieves competitive accuracy with state-of-the-art parsers in analyzing CFG trees or dependencies, and is one of the most promising methodologies thus far. These methods, however, cannot simply be applied to unification-based grammar parsing because it can fail as a result of its hard constraints in the grammar. Even though parsing failure can be recovered by nondeterministically exploring the search space, it is hard to find a search path that produces a parse tree that satisfies all grammar constraints because its hypothesis space is exponentially large.

There have been several studies related to our work. The nondeterministic shift-reduce parsing for unification-based grammars has been studied by Briscoe and Carroll (1993b). However, their algorithm works nondeterministically with the mechanism of the packed parse forest, and hence it has the problem of locality in the packed parse forest. Bouma (1990) exemplified the application of grammar rules by default unification, in which some constraints in grammar rules were specified as default information. Optimality theory in LFG (Frank *et al.* 2001) is also a sort of default-rule application for developing wide-coverage grammars and giving preference to parsing results. Constraints in a grammar rule are ordered in

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

optimality theory, and the grammar writer can provide robust and specific controls for each language phenomenon corresponding to a constraint. The difference between our study and others is that (i) we used shift-reduce parsing, (ii) preferences were determined by a machine learning classifier and (iii) non-head daughters were specified as default information instead of default information being specified for each constraint in the grammar rules. Zhang *et al.* (2007) proposed selective *n*-best unpacking for unification-based grammars. Using the same approach as Huang (2008), they tried to process the reranking and enumeration of *n*-best candidates simultaneously by extracting the *n*-best partial parse trees for each node in the packed parse forest. Their method provided precise reranking of the packed parse forest with non-local features, but it was limited by the types of non-local features. There is a trade-off between the decoding time and the types of non-local features. If further non-local features were used, exact decoding would be difficult within a reasonable time.

3 Unification-based grammars

A unification-based grammar is defined as a pair consisting of a set of lexical entries and a set of grammar rules. The lexical entries express word-specific characteristics, while the grammar rules describe constructions of constituents in parse trees. Both the grammar rules and the lexical entries are represented by feature structures (Carpenter 1992), and constraints in the grammar are forced by unification. Of the grammar rules, a binary rule is a partial function: $F \times F \rightarrow F$, where F is the set of all possible feature structures. The binary rule takes two partial parse trees as daughters and returns a larger partial parse tree that consists of the daughters and their mother. A unary rule is a partial function: $F \rightarrow F$, which corresponds to a unary branch.

In the experiments, we used an HPSG (Pollard and Sag 1994), which is one of the most sophisticated unification-based grammars in linguistics. An HPSG generally has a small number of grammar rules and a large number of lexical entries where a head daughter is determined for each binary rule. Let H be a partial parse tree corresponding to a head daughter and NH a partial parse tree corresponding to a non-head daughter. The binary rule and unary rule are defined as

$$g_R(H, NH) = R \sqcup \left[\text{DTRS:H:}H\right] \sqcup \left[\text{DTRS:NH:}NH\right]$$
$$g_R(H) = R \sqcup \left[\text{DTRS:H:}H\right]$$

where R is a feature structure corresponding to a binary/unary rule, and $F \sqcup G$ returns the result of the unification of feature structures F and G. In what follows, we write $\bot$ for the most general type, $F \sqcup G$ for the unification of two feature structures, $F \sqcap G$ for the generalization of two feature structures and $F \sqsubseteq G$ for subsumption relations, in which G is more specific than or equal to F (F is more general than or equal to G)[1].

[1] We have followed the notation convention in (Carpenter 1992). The use of these operators might be opposite for some readers, e.g., $F \sqcap G$ is a unification of F and G.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

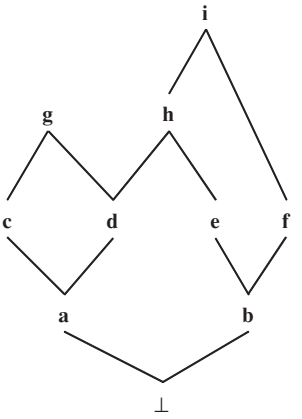


Fig. 1. Example of partially ordered types.

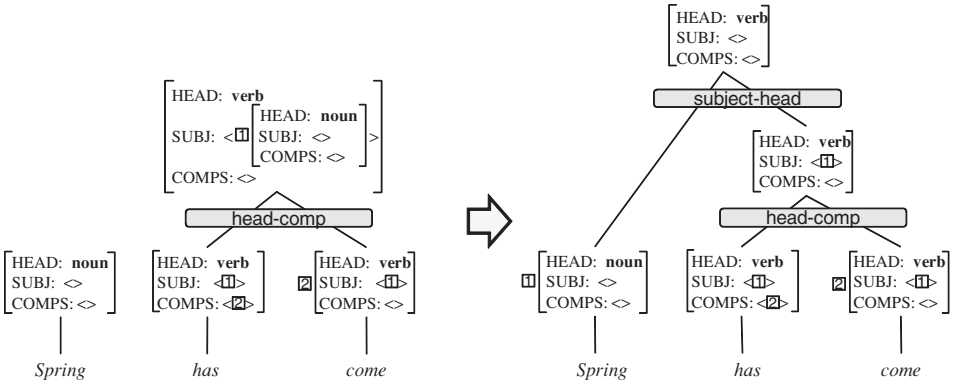


Fig. 2. Example of HPSG parsing.

Feature structures in an HPSG are strictly typed. Let Type be a finite set of types that are partially ordered according to their specificity. We write $\sigma \sqcup \tau$ for the type unification of two types, and $\sigma \sqsubseteq \tau$ for subsumption relations in which τ is more specific than or equal to σ. The type unification of two types, $\sigma \in \mathtt{Type}$ and $\tau \in \mathtt{Type}$, is defined as their least upper bound, i.e., the most general type $\gamma \in \mathtt{Type}$ such that $\sigma \sqsubseteq \gamma$ and $\tau \sqsubseteq \gamma$. Figure 1 has an example of partially ordered types, in which the types in the upper positions are more specific than the types in the lower positions. The type unification of **d** and **b** returns **h** because **d** and **b** have in common more specific types (**h** and **i**) and **h** is the most general type. However, the type unification of **c** and **b** fails because **c** and **b** do not have in common more specific types. Each node in the typed feature structures is assigned a single type in Type. In the process of unifying two feature structures, a pair of two nodes is unified one by one through type unification. If type unification fails at any point during the unification, the latter also returns unification failure as a result.

Figure 2 shows an example of HPSG parsing of the sentence, 'Spring has come'. The left side of the figure shows a partial parse tree for 'has come', which is obtained

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

by unifying each of the lexical entries for 'has' and 'come' with a daughter feature structure of the head-complement rule. Larger partial parse trees are obtained by repeatedly applying phrase-structure rules to lexical/phrasal partial parse trees. Suppose the subject-head rule is

$$g(NH, H) = \left[\begin{array}{l} \text{HEAD}:\boxed{1} \\ \text{SUBJ}:\boxed{3} \\ \text{COMPS}:\langle\rangle \\ \text{DTRS}: \left[\begin{array}{l} \text{H}: \left[\begin{array}{l} \text{HEAD}:\boxed{1} \\ \text{SUBJ}:\langle\boxed{2}|\boxed{3}\rangle \\ \text{COMPS}:\langle\rangle \end{array}\right] \\ \text{NH}:\boxed{2} \end{array}\right] \end{array}\right] \sqcup \left[\text{DTRS:H:}H\right] \sqcup \left[\text{DTRS:NH:}NH\right].$$

By applying it to the partial parse trees on the left of the figure, a larger partial parse tree is obtained, as on the right. Finally, the parse results in output as a parse tree that dominates the sentence. We have written $g_R(H, NH) = R \sqcup H \sqcup NH$ for the binary rules in what follows for simplicity.

We used Enju, a treebank grammar of an HPSG for English (Miyao *et al.* 2005), in the experiments. Enju is a sort of treebank grammar; the lexicon of Enju is semiautomatically developed from a CFG-style treebank. Therefore, it is occasionally considered to be a data-driven parsing grammar, such as a probabilistic context-free grammar (PCFG), but Enju is more theoretically and linguistically developed than PCFGs or simple dependency grammars. Enju was developed by using a corpus-oriented method of grammar development (Miyao *et al.* 2005), in which the grammar rules were developed manually, and the lexicon was automatically developed from an HPSG treebank by harvesting leaves in the treebank. The crux of this method of grammar development was how to develop an HPSG treebank that satisfied the grammar constraints defined in the original HPSG. An HPSG treebank was developed as follows: an existing CFG-style treebank was first converted to a primitive HPSG treebank, and then the primitive HPSG treebank was specialized so that it satisfied the grammar constraints in the grammar rules by applying them to each branch. Nonterminal/terminal symbols in the CFG-style treebank were converted to phrasal/lexical feature structures in the primitive HPSG treebank so that they satisfied the constraints represented by symbols in the CFG-style treebank. Tree skeletons in the CFG-style treebank were also changed to tree skeletons that conformed with the HPSG theory. Other constraints that could be inferred from the primitive HPSG treebank were also added to the primitive HPSG treebank, such as wh-phrases and a grammar rule name for each branch. Finally, the primitive HPSG treebank was constrained by the grammar rules. Consequently, constraints derived from the nonterminal/terminal symbols were strictly checked in the derived HPSG treebank as to whether they conformed with the HPSG theory through constraints imposed by the grammar rules. For example, a noun phrase having a verb head word was strictly prohibited by the head feature principle, i.e., constraints had to have the same category between a mother and a head daughter in the grammar rules.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

4 Default unification

Default unification was investigated in a series of studies that involved robust parsing by unification-based grammars and lexical semantics in order to deal with default inheritance in a lexicon (Bouma 1990; Briscoe 1993a; Ninomiya, Miyao and Tsujii 2002). This is advantageous to achieve robust processing, because (i) it almost always succeeds and (ii) a feature structure is relaxed such that the amount of information is maximized (Bouma 1990; Ninomiya *et al.* 2002). We tested a simplified version of Copestake's (1993) default unification in our experiments. Before explaining this, let us first explain Carpenter's two definitions of default unification (Carpenter 1993).

4.1 Carpenter's default unification

Carpenter proposed the following two types of default unification (Carpenter 1993):

Credulous default unification

$$F \overset{<}{\sqcup}_c G = \{F \sqcup G' \mid G' \sqsubseteq G \text{ is maximal such that} F \sqcup G' \text{ is defined}\} \tag{1}$$

Skeptical default unification

$$F \overset{<}{\sqcup}_s G = \bigsqcap (F \overset{<}{\sqcup}_c G) \tag{2}$$

F is called a strict feature structure, whose information must not be lost, and G is called a default feature structure, whose information can be lost, although this needs to be as limited as possible so that F and G can be unified.

Credulous default unification (1) is greedy, in that it tries to maximize the amount of information from the default feature structure, but it does result in a set of feature structures. Skeptical default unification (2) simply generalizes the set of feature structures resulting from credulous default unification. Skeptical default unification thus leads to a unique result so that there remains the default information that can be found in every result for credulous default unification. The following is an example of skeptical default unification:

$$\left[\mathrm{F}{:}\mathbf{a}\right] \overset{<}{\sqcup}_s \begin{bmatrix} \mathrm{F}{:}\boxed{1}\mathbf{b} \\ \mathrm{G}{:}\boxed{1} \\ \mathrm{H}{:}\mathbf{c} \end{bmatrix} = \sqcap \left\{ \begin{bmatrix} \mathrm{F}{:}\mathbf{a} \\ \mathrm{G}{:}\mathbf{b} \\ \mathrm{H}{:}\mathbf{c} \end{bmatrix}, \begin{bmatrix} \mathrm{F}{:}\boxed{1}\mathbf{a} \\ \mathrm{G}{:}\boxed{1} \\ \mathrm{H}{:}\mathbf{c} \end{bmatrix} \right\} = \begin{bmatrix} \mathrm{F}{:}\mathbf{a} \\ \mathrm{G}{:}\bot \\ \mathrm{H}{:}\mathbf{c} \end{bmatrix} \tag{3}$$

4.2 Copestake's default unification

Copestake (1993) mentioned that the problem with Carpenter's default unification (1 and 2) is its time complexity. Carpenter's default unification takes exponential time to find the optimal answer because it requires the unifiability of the power set of constraints to be checked in a default feature structure. Copestake (1993) thus proposed the following definition of default unification. Let $PV(G)$ be a function that returns a set of path values in G, and let $PE(G)$ be a function that returns a set of path equations, i.e., information about structure-sharing in G.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Copestake's default unification

$$F \overset{<}{\sqcup}_a G = H \sqcup \bigsqcup \left\{ I \;\middle|\; \begin{array}{l} I \in PV(G) \text{ and } H \sqcup I \text{ is defined and there is no } I' \in PV(G) \\ \text{such that } H \sqcup I' \text{ is defined and } H \sqcup I \sqcup I' \text{ is not defined} \end{array} \right\} \tag{4}$$

where $H = F \sqcup \bigsqcup PE(G)$. It forces to first unify all path equations in the default feature structure, then all path values that are consistent with the strict feature structure and all other path values are unified. Copestake's default unification returns the following result for the example in (3):

$$\left[\text{F:}\mathbf{a}\right] \overset{<}{\sqcup}_a \begin{bmatrix} \text{F:}\boxed{1}\mathbf{b} \\ \text{G:}\boxed{1} \\ \text{H:}\mathbf{c} \end{bmatrix} = \begin{bmatrix} \text{F:}\boxed{1}\mathbf{a} \\ \text{G:}\boxed{1} \\ \text{H:}\mathbf{c} \end{bmatrix}$$

Copestake's default unification can fail if one of the path equations in the default feature structure is not consistent with the strict feature structure. The following is an example for which Carpenter's default unification returns an answer but Copestake's default unification fails:

$$\begin{bmatrix} \text{F:}\mathbf{a} \\ \text{G:}\mathbf{b} \end{bmatrix} \overset{<}{\sqcup}_s \begin{bmatrix} \text{F:}\boxed{1} \\ \text{G:}\boxed{1} \\ \text{H:}\mathbf{c} \end{bmatrix} = \begin{bmatrix} \text{F:}\mathbf{a} \\ \text{G:}\mathbf{b} \\ \text{H:}\mathbf{c} \end{bmatrix} \qquad \text{(Carpenter)}$$

$$\begin{bmatrix} \text{F:}\mathbf{a} \\ \text{G:}\mathbf{b} \end{bmatrix} \overset{<}{\sqcup}_a \begin{bmatrix} \text{F:}\boxed{1} \\ \text{G:}\boxed{1} \\ \text{H:}\mathbf{c} \end{bmatrix} \quad \text{not defined} \qquad \text{(Copestake)}$$

Copestake's default unification (4) works efficiently. All path equations in the default feature structure are unified with the strict feature structures, and then the unifiability of path values is checked one by one for each node in the result of unifying the path equations. The algorithm for Copestake's default unification runs as fast as normal unification, and can be executed even for feature structures that include cycles.

4.3 Typed version of Copestake's default unification

Copestake's default unification is well defined and works efficiently. When types are involved, however, it is not easy to find unifiable path values in the default feature structure. She introduced type decomposition function $TypeDecomp(F, G)$ instead of path value function $PV(F)$. Let F be $\langle Q, q_0, \delta, \theta \rangle$ and G be $\langle Q', q'_0, \delta', \theta' \rangle$, where Q is a set of feature structure nodes, q_0 is the root node, $\delta(f, q)$ is a partial function that returns a feature structure node reached by following f from q, and $\theta(q)$ is a type assignment function for q. $TypeDecomp(F, G)$ is defined as

- If $\theta(q_0) \sqsubseteq \theta'(q'_0)$, then $TypeDecomp(F, G)$ is the set of all feature structures, $H = \langle Q'', q''_0, \delta'', \theta'' \rangle$ such that:
  - $\theta''(q''_0) = \theta(q_0)$.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

— If F has features, i.e., $\delta(f, q_0)$ is defined for some f, then $\delta''(f, q_0'')$ for some f such that $\delta(f, q_0)$ is defined is a member of $TypeDecomp(\delta(f, q_0), \delta'(f, q_0'))$ and $\delta''(f', q_0'')$ for all $f' \neq f$ is not defined.

- If $\theta(q_0) \not\sqsubseteq \theta'(q_0')$, then $TypeDecomp(F, G) = \{F\}$.

The typed version of Copestake's default unification is defined as

$$(5)\quad F \overset{<}{\sqcup}_a G = H \sqcup \bigsqcup \left\{ I \;\middle|\; \begin{array}{l} I \in D \text{ and } H \sqcup I \text{ is defined and there is no } I' \in D \\ \text{such that } H \sqcup I' \text{is defined and } H \sqcup I \sqcup I' \text{ is not defined} \end{array} \right\}$$

where $H = F \sqcup \bigsqcup PE(G)$ and D is $TypeDecomp(\sqcup PV(G), F)$. In the typed version, $TypeDecomp(\sqcup PV(G), F)$ is used instead of $PV(G)$, and it returns a set of feature structures. A feature structure returned by $TypeDecomp(\sqcup PV(G), F)$ is constructed by following and copying a path on $\sqcup PV(G)$, and following the same path on F simultaneously. If type σ on a current node on $\sqcup PV(G)$ is neither more general than nor equal to type τ on a current node on F, i.e., $\sigma \not\sqsubseteq \tau$, then the remaining feature structure on $\sqcup PV(G)$ is attached to the end of the copied path. Intuitively, $TypeDecomp(\sqcup PV(G), F)$ returns a set of path values that are neither more general nor equal to the strict feature structure. The algorithm for this definition is more efficient than finding the optimal answer, and this definition is well suited to a grammar based on a type constraint system, in which each type is constrained with a bag of constraints. If a type in the default is not incorporated into the strict feature structure, then a bag of constraints defined with the type should also not be incorporated.

Suppose we have two feature structures as

$$(6)\quad F = \begin{bmatrix} \mathbf{a} \\ \text{F:} \begin{bmatrix} \mathbf{b} \\ \text{F:}\mathbf{d} \\ \text{G:}\bot \end{bmatrix} \\ \text{G:} \begin{bmatrix} \mathbf{c} \\ \text{F:}\mathbf{d} \\ \text{G:}\bot \end{bmatrix} \end{bmatrix}, G = \begin{bmatrix} \mathbf{a} \\ \text{F:} \begin{bmatrix} \mathbf{b} \\ \text{F:}\mathbf{e} \\ \text{G:}\mathbf{f} \end{bmatrix} \\ \text{G:} \begin{bmatrix} \mathbf{g} \\ \text{F:}\mathbf{e} \\ \text{G:}\mathbf{f} \end{bmatrix} \end{bmatrix}$$

where $\mathbf{d} \sqcup \mathbf{e}$ is not defined and $\mathbf{c} \sqsubseteq \mathbf{g}$. $TypeDecomp(\sqcup PV(G), F)$ is

$$\left\{ \begin{bmatrix} \mathbf{a} \\ \text{F:} \begin{bmatrix} \mathbf{b} \\ \text{F:}\mathbf{e} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \mathbf{a} \\ \text{F:} \begin{bmatrix} \mathbf{b} \\ \text{G:}\mathbf{f} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \mathbf{a} \\ \text{G:} \begin{bmatrix} \mathbf{g} \\ \text{F:}\mathbf{e} \\ \text{G:}\mathbf{f} \end{bmatrix} \end{bmatrix} \right\}$$

Thus, the result of the default unification of F and G becomes

$$F \overset{<}{\sqcup}_a G = \begin{bmatrix} \mathbf{a} \\ \text{F:} \begin{bmatrix} \mathbf{b} \\ \text{F:}\mathbf{d} \\ \text{G:}\mathbf{f} \end{bmatrix} \\ \text{G:} \begin{bmatrix} \mathbf{c} \\ \text{F:}\mathbf{d} \\ \text{G:}\bot \end{bmatrix} \end{bmatrix}$$

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

```
procedure type-unconstrained_unification(p, q)
  queue := {⟨p, q⟩};
  while( queue is not empty )
    ⟨p, q⟩ := shift(queue);
    p := deref(p); q := deref(q);
    if p ≠ q
      θ(p) := θ(p) ∪ θ(q);
      θ(q) := ptr(p);
      forall f ∈ feat(p) ∪ feat(q)
        if f ∈ feat(p) ∧ f ∈ feat(q)
          queue := queue ∪ ⟨δ(f, p), δ(f, q)⟩;
        if f ∉ feat(p) ∧ f ∈ feat(q)
          δ(f, p) := δ(f, q);

procedure mark(p, m)
  p := deref(p);
  if p has not been visited
    θ(p) := {⟨θ(p), m⟩};
    forall f ∈ feat(p)
      mark(δ(f, p), m);

procedure collapse_defaults(p)
  p := deref(p);
  if p has not been visited
    ts := ⊥; td := ⊥;
    forall ⟨t, strict⟩ ∈ θ(p)
      ts := ts ⊔ t;
    forall ⟨t, default⟩ ∈ θ(p)
      td := td ⊔ t;
    if ts is not defined
      return false;
    if ts ⊔ td is defined
      θ(p) := ts ⊔ td;
    else
      θ(p) := ts;
    forall f ∈ feat(p)
      collapse_defaults(δ(f,p));

procedure default_unification(p, q)
  mark(p, strict);
  mark(q, default);
  type-unconstrained unification(p, q);
  collapse_defaults(p);
```

$\theta(p)$ is (i) a single type, (ii) a pointer or (iii) a set of pairs of types and markers in the feature structure node p. A marker indicates that the types in a feature structure node originally belong to the strict feature structures or the default feature structures. A pointer indicates that the node has been unified with other nodes and it points to the unified node. A function *deref* traverses pointer nodes until it reaches non-pointer node.

$\delta(f, p)$ returns a feature structure node reached by following a feature f from p.

Fig. 3. Algorithm for the simply typed version of Copestake's default unification.

4.4 Simply typed version of Copestake's default unification

The typed version of Copestake's default unification (5) is suited to a type constraint system, but it assumes the acyclicity of feature structures, and still involves huge computational cost to find the compatible feature structures in $TypeDecomp(\sqcup PV(G), F)$. Therefore, we implemented a simply typed version of Copestake's default unification, which does not assume the acyclicity of feature structures, and runs as fast as normal unification.

Figure 3 has the algorithm by which we implemented the simply typed version. First, each node is marked 'strict', if it belongs to a strict feature structure, otherwise it is marked 'default.' The marked strict and default feature structures are unified by

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

type-unconstrained unification. Type-unconstrained unification is a sort of unification but the types in the feature structure nodes are not unified but merged as a set of types, and hence type-unconstrained unification never fails. The following is an example of type-unconstrained unification of marked feature structures:

$$(7)\quad \begin{bmatrix} \mathbf{t}(strict) \\ \mathrm{F}{:}\mathbf{a}(strict) \\ \mathrm{G}{:}\mathbf{b}(strict) \end{bmatrix} \sqcup_u \begin{bmatrix} \mathbf{u}(default) \\ \mathrm{F}{:}\boxed{1}\mathbf{c}(default) \\ \mathrm{G}{:}\boxed{1} \\ \mathrm{H}{:}\mathbf{d}(default) \end{bmatrix} = \begin{bmatrix} \{\mathbf{u}(default), \mathbf{t}(strict)\} \\ \mathrm{F}{:}\boxed{1}\{\mathbf{a}(strict), \mathbf{b}(strict), \mathbf{c}(default)\} \\ \mathrm{G}{:}\boxed{1} \\ \mathrm{H}{:}\{\mathbf{d}(default)\} \end{bmatrix}$$

Then all types marked 'strict' are unified into one type for each node. If this fails, then default unification also returns unification failure as its result. Finally, each node is assigned a single type, which is the result of type unification for all types marked both "default' and 'strict,' if it succeeds, otherwise all types marked only 'strict.' otherwise Supposing that $\mathbf{t} \sqsubseteq \mathbf{u}$, $\mathbf{a} \sqcup \mathbf{b} = \mathbf{e}$ and $\mathbf{e} \sqcup \mathbf{c}$ is not defined, the result for the above example (7) becomes

$$\begin{bmatrix} \mathbf{t} \\ \mathrm{F}{:}\mathbf{a} \\ \mathrm{G}{:}\mathbf{b} \end{bmatrix} \overset{<}{\sqcup}_{a'} \begin{bmatrix} \mathbf{u} \\ \mathrm{F}{:}\boxed{1}\mathbf{c} \\ \mathrm{G}{:}\boxed{1} \\ \mathrm{H}{:}\mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{u} \\ \mathrm{F}{:}\boxed{1}\mathbf{e} \\ \mathrm{G}{:}\boxed{1} \\ \mathrm{H}{:}\mathbf{d} \end{bmatrix}.$$

The following example illustrates the difference in behaviour between the typed version and the simply typed version for the two feature structures in (6):

$$\begin{bmatrix} \mathbf{a} \\ \mathrm{F}{:}\begin{bmatrix} \mathbf{b} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\bot \end{bmatrix} \\ \mathrm{G}{:}\begin{bmatrix} \mathbf{c} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\bot \end{bmatrix} \end{bmatrix} \overset{<}{\sqcup}_{a} \begin{bmatrix} \mathbf{a} \\ \mathrm{F}{:}\begin{bmatrix} \mathbf{b} \\ \mathrm{F}{:}\mathbf{e} \\ \mathrm{G}{:}\mathbf{f} \end{bmatrix} \\ \mathrm{G}{:}\begin{bmatrix} \mathbf{g} \\ \mathrm{F}{:}\mathbf{e} \\ \mathrm{G}{:}\mathbf{f} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \mathbf{a} \\ \mathrm{F}{:}\begin{bmatrix} \mathbf{b} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\mathbf{f} \end{bmatrix} \\ \mathrm{G}{:}\begin{bmatrix} \mathbf{c} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\bot \end{bmatrix} \end{bmatrix} \quad \text{(Typed version, Copestake)}$$

$$\begin{bmatrix} \mathbf{a} \\ \mathrm{F}{:}\begin{bmatrix} \mathbf{b} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\bot \end{bmatrix} \\ \mathrm{G}{:}\begin{bmatrix} \mathbf{c} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\bot \end{bmatrix} \end{bmatrix} \overset{<}{\sqcup}_{a'} \begin{bmatrix} \mathbf{a} \\ \mathrm{F}{:}\begin{bmatrix} \mathbf{b} \\ \mathrm{F}{:}\mathbf{e} \\ \mathrm{G}{:}\mathbf{f} \end{bmatrix} \\ \mathrm{G}{:}\begin{bmatrix} \mathbf{g} \\ \mathrm{F}{:}\mathbf{e} \\ \mathrm{G}{:}\mathbf{f} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \mathbf{a} \\ \mathrm{F}{:}\begin{bmatrix} \mathbf{b} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\mathbf{f} \end{bmatrix} \\ \mathrm{G}{:}\begin{bmatrix} \mathbf{g} \\ \mathrm{F}{:}\mathbf{d} \\ \mathrm{G}{:}\mathbf{f} \end{bmatrix} \end{bmatrix} \quad \text{(Simply typed version)}$$

where $\mathbf{d} \sqcup \mathbf{e}$ is not defined and $\mathbf{c} \sqsubseteq \mathbf{g}$. The simply typed version greedily tries to unify types for each node in the default feature structure, but the typed version tries to unify decomposed parts of the default feature structure. In the example above, the feature structure reached by following feature G: from the root node of the default feature structure is a decomposed part of the default feature structure. The typed version does not unify this decomposed part because the type **e** in the decomposed part is inconsistent with the type **d** in the strict feature structure. The simply typed version greedily unifies the type **g** and **f** in the decomposed part with the strict feature structure.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

4.5 Ninomiya et al.'s default unification

Ninomiya *et al.* (2002) proposed another definition of default unification. It first unified the strict and default feature structures through type-unconstrained unification. Then, each node with inconsistent types was coerced to have the most general type, and was marked with 'failpoint.' All structure-sharings on the path that led to the fail points were unfolded. Finally, the strict feature structure was unified with it by normal unification. Ninomiya *et al.*'s (2002) default unification assumed two reasons for unification failure. The first involved the types on the failpoints, and the second involved the structure-sharings on the path that led to the failpoints. Their algorithm generalized the type on the failpoints and removed all structure-sharings on the path to the failpoints. Copestake's default unification forced all path equations to be unified with the strict feature structure, and hence default unification failed if a path equation in the default was inconsistent with the strict feature structure. However, Ninomiya *et al.*'s (2002) default unification always succeeds because it relaxes path equations. Suppose grammar rule R and daughters DTR are given as

$$R = \begin{bmatrix} \text{HEAD}:\boxed{1} \\ \text{DTRS}: \begin{bmatrix} \text{H}:\boxed{2}\begin{bmatrix} \text{HEAD}:\boxed{1}\textbf{head} \end{bmatrix} \\ \text{NH:SPR}:\boxed{2} \end{bmatrix} \end{bmatrix}$$

$$DTR = \begin{bmatrix} \text{DTRS}: \begin{bmatrix} \text{H:HEAD:CASE}:\textbf{obj} \\ \text{NH:SPR:HEAD:CASE}:\textbf{nom} \end{bmatrix} \end{bmatrix}.$$

Also Suppose the type **head** has PHON:, CASE:, and INV: as its features, and the type **sign** has HEAD: and VAL:. Copestake's default unification results in unification failure because all structure-sharings in R are forced to unify with DTR. Ninomiya *et al.*'s (2002) default unification returns much more informative results as follows:

$$DTR \overset{<}{\sqcup}_n R = \begin{bmatrix} \text{HEAD}: \begin{bmatrix} \textbf{head} \\ \text{PHON}:\boxed{3} \\ \text{CASE}:\bot \\ \text{INV}:\boxed{4} \end{bmatrix} \\ \text{DTRS}: \begin{bmatrix} \text{H}: \begin{bmatrix} \textbf{sign} \\ \text{HEAD}: \begin{bmatrix} \textbf{head} \\ \text{PHON}:\boxed{3} \\ \text{CASE}:\textbf{obj} \\ \text{INV}:\boxed{4} \end{bmatrix} \\ \text{VAL}:\boxed{5} \end{bmatrix} \\ \text{NH:SPR}: \begin{bmatrix} \textbf{sign} \\ \text{HEAD}: \begin{bmatrix} \textbf{head} \\ \text{PHON}:\boxed{3} \\ \text{CASE}:\textbf{nom} \\ \text{INV}:\boxed{4} \end{bmatrix} \\ \text{VAL}:\boxed{5} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

The information on structure-sharings is preserved as much as possible. In the example, the structure-sharing tagged as 2 in original grammar rule R is decomposed into the structure-sharing 3 4 5. However, Ninomiya *et al.*'s (2002) default unification cannot be applied to cyclic feature structures. In the preliminary experiments (Section 6.2), cyclic feature structures were coerced to be acyclic by eliminating feature pointers that form cycles.

5 Shift-reduce parsing for unification-based grammars

This section explains our shift-reduce parsing algorithms, which are based on deterministic shift-reduce CFG parsing (Sagae and Lavie 2005) and best-first shift-reduce CFG parsing (Sagae and Lavie 2006). Sagae and Lavie's parser selected the most probable shift/reduce actions and non-terminal symbols without assuming explicit CFG rules. Therefore, their parser could proceed deterministically without failure. However, a deterministic parser for unification-based grammars can fail as a result of its hard constraints in the grammar. We propose three new shift-reduce parsing approaches to unification-based grammars: deterministic shift-reduce parsing and shift-reduce parsing by backtracking and a best-first search. The major difference between our algorithm and Sagae and Lavie's algorithm is that we use default unification. First, let us explain the deterministic shift-reduce parsing algorithm and then shift-reduce parsing with backtracking and a best-first search.

5.1 Deterministic shift-reduce parsing for unification-based grammars

The deterministic shift-reduce parsing algorithm for unification-based grammars is mainly composed of two data structures: a stack, S, and a queue, W. Items in S are partial parse trees, including a lexical entry, and a parse tree that dominates the whole input sentence. Items in W are words and POS tags in the input sentence. The algorithm defines two types of parser actions, shift and reduce, which are as follows

Shift. A shift action removes the first item (a word and a POS tag) from W. Then, one lexical entry is selected from the candidate lexical entries for the item. Finally, the selected lexical entry is placed on the top of the stack.

Binary Reduce. A binary reduce action removes two items from the top of the stack. Then, partial parse trees are derived by applying binary rules to the first removed item as a right daughter and a left daughter as the second removed item. Of the candidate partial parse trees, one is selected and placed on the top of the stack.

Unary Reduce. A unary reduce action removes one item from the top of the stack. Then, partial parse trees are derived by applying unary rules to the removed item. Of the candidate partial parse trees, one is selected and placed on the top of the stack.

Parsing fails if there are no candidates for selection (i.e., a dead end). Parsing is considered to have successfully finished when W is empty and S has only one item

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

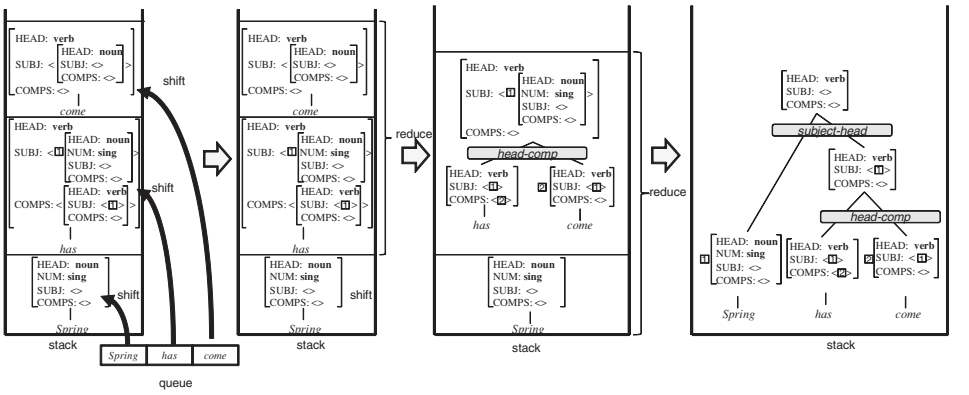


Fig. 4. Example of deterministic shift-reduce parsing for HPSG.

that satisfies the sentential condition: the category is verb and the subcategorization frame is empty. Parsing is considered to be a nonsentential success when W is empty and S has only one item that does not satisfy the sentential condition.

Figure 4 has an example of deterministic shift-reduce HPSG parsing of the sentence, 'Spring has come.' First, shift actions are selected thrice. A lexical entry is selected for each word and placed on the top of the stack. Then, the top two elements in the stack are reduced by unifying them with the head-complement rule. Finally, a parse tree is obtained on the top of the stack by applying a reduce action to the top two elements by unifying them with the head-subject rule.

We used a multi-class logistic regression classifier (also known as a maximum entropy or log-linear model) in our experiments to choose the parser's action. Table 1 lists the feature templates for the classifier, and Table 2 lists the combinations of feature templates. The implemented classifier had sixty-two feature templates, 155 combinations of feature templates, and 1,806,069 features. Many of these features were taken from those used in Ninomiya *et al.*'s (2007), Miyao and Tsujii's (2005), and Sagae and Lavie's (2005) parsers, including global features defined over the information in the stack, which cannot be used in parsing with the packed parse forest. The global features are specifically those including feature templates $Sw(i)$, $Sp(i)$, $Sl(i)$, $Ssy(i)$, $Shsy(i)$, $Snsy(i)$, $Shp(i)$, and $Snp(i)$ such that $i \geq 1$. The number of feature template combinations, including global features, is fifty-two of 155. The features for selecting shift actions are the features used in the supertagger (Ninomiya *et al.* 2007) and the features used in Sagae and Lavie's (2005) parser. Our shift-reduce parsers can be regarded as an extension of the supertagger.

Deterministic parsing can fail because of its grammar's hard constraints. For example, Figure 5 shows parsing failure in the deterministic parsing of the sentence, 'Spring has come.' Deterministic parsing for this sentence fails because an incorrect lexical entry has been selected for the verb, 'has,' and consequently parsing fails due to the disagreement in number between the subject, 'spring', and the verb, 'has.' Because no result is returned when unification failure occurs and the process is deterministic, we cannot continue deterministic parsing. Consequently, we use default unification, which almost always succeeds in unification. We assume that a

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 1. *Feature templates*

Common features:	$Sw(i), Sp(i), Sl(i), Ssy(i), Shsy(i), Snsy(i), Shp(i), Snp(i), w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, w_{i+3}, p_{i-2}, p_{i-1}, p_i, p_{i+1}, p_{i+2}, p_{i+3}$
Shift features:	l
Binary reduce features:	$r, d, c, opl, opr, syl, syr, hwl, hwr, hpl, hpr, hll, hlr, spl, spr, ncl, ncr$
Unary reduce features:	$r, opl, opr, sy, hw, hp, hl, sp, nc$

$Sw(i)$ . . . head word of i-th item from the top of the stack
$Sp(i)$ . . . head POS tag of i-th item from the top of the stack
$Sl(i)$ . . . head lexical entry of i-th item from the top of the stack
$Ssy(i)$ . . . symbol of phrase category of the i-th item from the top of the stack
$Shsy(i)$ . . . symbol of phrase category of the head daughter of the i-th item from the top of the stack
$Snsy(i)$ . . . symbol of phrase category of the non-head daughter of the i-th item from the top of the stack
$Shp(i)$ . . . head POS tag of the head daughter of i-th item from the top of the stack
$Snp(i)$ . . . head POS tag of the non-head daughter of i-th item from the top of the stack
l . . . lexical entry
r . . . rule
d . . . distance between head words of daughters
c . . . whether a comma exists between daughters and/or inside daughter phrases
opl, opr . . . POS tag of a word adjacent to the left/right-side of the phrase
sy, syl, syr . . . symbol of phrase category of the -/left/right daughters
hw, hwl, hwr . . . head word of the -/left/right daughters
hp, hpl, hpr . . . head POS tag of the -/left/right daughters
hl, hll, hlr . . . head lexical entry of the -/left/right daughters
sp, spl, spr . . . the number of words dominated by the -/left/right daughters
nc, ncl, ncr . . . the number of commas dominated by the -/left/right daughters

head daughter (or, an important daughter) is determined for each binary rule in the unification-based grammar. Default unification is used in applying a binary rule, in which a binary rule unified with the head daughter is the strict feature structure and the non-head daughter is the default feature structure, i.e., $(R \sqcup H) \overset{\leftarrow}{\sqcup} NH$, where R is a binary rule, H is a head daughter, and NH is a non-head daughter. We used a simply typed version of Copestake's default unification in applying the binary rule application in the experiments. Note that default unification was always used instead of normal unification in both the training and the evaluation of parsers using default unification. Although the simply typed version of Copestake's default unification almost always succeeds, the binary rule application can fail if the binary rule cannot be unified with the head daughter, or inconsistency is caused by path equations in the default feature structures. If the application of rules fails for all binary rules, backtracking or a best-first search can be used for recovery, as explained in Section 5.2. We had no failures in applying the binary rules with default unification in the experiments.

We also evaluated deterministic parsing with normal unification to evaluate the effect of default unification, and compared it with deterministic parsing with default unification. Deterministic parsing with normal unification can easily fail, and hence

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 2. *Combinations of feature templates*

Shift features

$\langle w_{i-1}, l\rangle\langle w_{i+1}, l\rangle\langle p_{i-2}, l\rangle\langle p_{i-1}, l\rangle\langle p_i, l\rangle\langle p_{i+1}, l\rangle\langle p_{i+2}, l\rangle\langle p_{i+3}, l\rangle\langle w_{i-1}, w_i, l\rangle\langle w_i, w_{i+1}, l\rangle\langle p_{i-1}, w_i, l\rangle$
$\langle p_i, w_i, l\rangle\langle p_{i+1}, w_i, l\rangle\langle p_i, p_{i+1}, p_{i+2}, p_{i+3}, l\rangle\langle p_{i-2}, p_{i-1}, p_i, l\rangle\langle p_{i-1}, p_i, p_{i+1}, l\rangle\langle p_i, p_{i+1}, p_{i+2}, l\rangle$
$\langle p_{i-2}, p_{i-1}, l\rangle\langle p_{i-1}, p_i, l\rangle\langle p_i, p_{i+1}, l\rangle\langle p_{i+1}, p_{i+2}, l\rangle\langle Sw(0), l\rangle\langle Sp(0), l\rangle\langle Sl(0), l\rangle\langle Sw(1), l\rangle\langle Sp(1), l\rangle$
$\langle Sl(1), l\rangle\langle Sw(2), l\rangle\langle Sl(2), l\rangle\langle Sw(3), l\rangle\langle Sp(3), l\rangle\langle Sl(3), l\rangle\langle Ssy(0), l\rangle\langle Shsy(0), l\rangle\langle Snsy(0), l\rangle\langle Shp(0), l\rangle$
$\langle Snp(0), l\rangle\langle Ssy(1), l\rangle\langle Snsy(1), l\rangle\langle Shp(1), l\rangle\langle Snp(1), l\rangle\langle d, l\rangle$

Binary reduce features

$\langle w_{i-2}, r\rangle\langle p_{i-2}, r\rangle\langle w_{i-1}, r\rangle\langle p_{i-1}, r\rangle\langle w_i, r\rangle\langle p_i, r\rangle\langle w_{i+1}, r\rangle\langle p_{i+1}, r\rangle\langle w_{i+2}, r\rangle\langle p_{i+2}, r\rangle\langle w_{i+3}, r\rangle\langle p_{i+3}, r\rangle$
$\langle Sp(0), r\rangle\langle Sl(0), r\rangle\langle Sw(1), r\rangle\langle Sp(1), r\rangle\langle Sl(1), r\rangle\langle Sw(2), r\rangle\langle Sp(2), r\rangle\langle Sl(2), r\rangle\langle Sw(3), r\rangle\langle Sp(3), r\rangle$
$\langle Sl(3), r\rangle\langle Ssy(0), r\rangle\langle Shsy(0), r\rangle\langle Snsy(0), r\rangle\langle Shp(0), r\rangle\langle Snp(0), r\rangle\langle Ssy(1), r\rangle\langle Shsy(1), r\rangle\langle Snsy(1), r\rangle$
$\langle Shp(1), r\rangle\langle Snp(1), r\rangle\langle d, r\rangle\langle r, d, c, hll, ncl, hlr, ncr\rangle\langle r, d, c, hpl, ncl, hpr, ncr\rangle$
$\langle r, d, c, hll, hplncl, nlr, hpr, ncr\rangle\langle r, d, c, syl, ncl, syr, ncr\rangle\langle r, c, hll, spl, ncl, hlr, spr, ncr\rangle$
$\langle r, c, hpl, spl, ncl, hpr, spr, ncr\rangle\langle r, c, hpl, hll, spl, ncl, hpr, hlr, spr, ncr\rangle\langle r, c, syl, spl, ncl, syr, spr, ncr\rangle$
$\langle r, d, c, hwl, hll, ncl, hwr, hlr, ncr\rangle\langle r, d, c, hwl, hpl, ncl, hwr, hpr, ncr\rangle$
$\langle r, d, c, hwl, hpl, hll, ncl, hwr, hpr, hlr, ncr\rangle\langle r, d, c, syl, hwl, ncl, syr, hwr, ncr\rangle$
$\langle r, c, hwl, hll, spl, ncl, hwr, hlr, spr, ncr\rangle\langle r, c, hwl, hpl, spl, ncl, hwr, hpr, spr, ncr\rangle$
$\langle r, c, hwl, hpl, hll, spl, ncl, hwr, hpr, hlr, spr, ncr\rangle\langle r, c, syl, hwl, spl, ncl, syr, hwr, spr, ncr\rangle$
$\langle r, d, c, hwl, hll, ncl, hlr, ncr\rangle\langle r, d, c, hwl, hpl, ncl, hpr, ncr\rangle\langle r, d, c, hwl, hpl, hll, ncl, hpr, hlr, ncr\rangle$
$\langle r, d, c, syl, hwl, ncl, syr, ncr\rangle\langle r, c, hwl, hll, spl, ncl, hlr, spr, ncr\rangle\langle r, c, hwl, hpl, spl, ncl, hpr, spr, ncr\rangle$
$\langle r, c, hwl, hpl, hll, spl, ncl, hpr, hlr, spr, ncr\rangle\langle r, c, syl, hwl, spl, ncl, hpr, hlr, spr, ncr\rangle$
$\langle r, d, c, opl, opr, hll, ncl, hlr, ncr\rangle\langle r, d, c, opl, opr, hpl, ncl, hpr, ncr\rangle$
$\langle r, d, c, opl, opr, hpl, hll, ncl, hpr, hlr, ncr\rangle\langle r, d, c, opl, opr, syl, ncl, syr, ncr\rangle$
$\langle r, c, opl, opr, hll, spl, ncl, hlr, spr, ncr\rangle\langle r, c, opl, opr, hpl, spl, ncl, hpr, spr, ncr\rangle$
$\langle r, c, opl, opr, hpl, hll, spl, ncl, hpr, hlr, spr, ncr\rangle\langle r, c, opl, opr, syl, spl, ncl, syr, spr, ncr\rangle$

Unary reduce features

$\langle w_{i-2}, r\rangle\langle p_{i-2}, r\rangle\langle w_{i-1}, r\rangle\langle p_{i-1}, r\rangle\langle w_i, r\rangle\langle p_i, r\rangle\langle w_{i+1}, r\rangle\langle p_{i+1}, r\rangle\langle w_{i+2}, r\rangle\langle w_{i+3}, r\rangle\langle p_{i+3}, r\rangle\langle Sw(0), r\rangle$
$\langle Sp(0), r\rangle\langle Sl(0), r\rangle\langle Sw(1), r\rangle\langle Sp(1), r\rangle\langle Sl(1), r\rangle\langle Sw(2), r\rangle\langle Sp(2), r\rangle\langle Sl(2), r\rangle\langle Sw(3), r\rangle\langle Sp(3), r\rangle$
$\langle Sl(3), r\rangle\langle Ssy(0), r\rangle\langle Shsy(0), r\rangle\langle Snsy(0), r\rangle\langle Shp(0), r\rangle\langle Snp(0), r\rangle\langle Ssy(1), r\rangle\langle Shsy(1), r\rangle\langle Snsy(1), r\rangle$
$\langle Shp(1), r\rangle\langle Snp(1), r\rangle\langle d, r\rangle\langle r, hl\rangle\langle r, hp\rangle\langle r, hp, hl\rangle\langle r, sy\rangle\langle r, hw, hl\rangle\langle r, hw, hp\rangle\langle r, hw, hp, hl\rangle\langle r, sy, hw\rangle$
$\langle r, opl, opr, hl\rangle\langle r, opl, opr, hp\rangle\langle r, opl, opr, hp, hl\rangle\langle r, opl, opr, sy\rangle$

recall will greatly decrease if a parser outputs nothing when it fails. Therefore, we evaluated it as follows. First, we evaluated partial parse trees that were analyzed until parsing failed. Second, we let the parser continue parsing after it failed parsing a sentence. Specifically, parsing involves the following seven steps:

(1) Let $P := \emptyset$.
(2) Do deterministic parsing with normal unification.
(3) If a fully connected sentential parse tree is the output, then return it.
(4) If a partial parse tree is the output, add it to P.
(5) If the right-most partial parse tree in P reaches the end of the sentence, then return P.
(6) Clear the stack, and continue parsing from the next position in the right-most partial parse tree.
(7) Go to 2.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

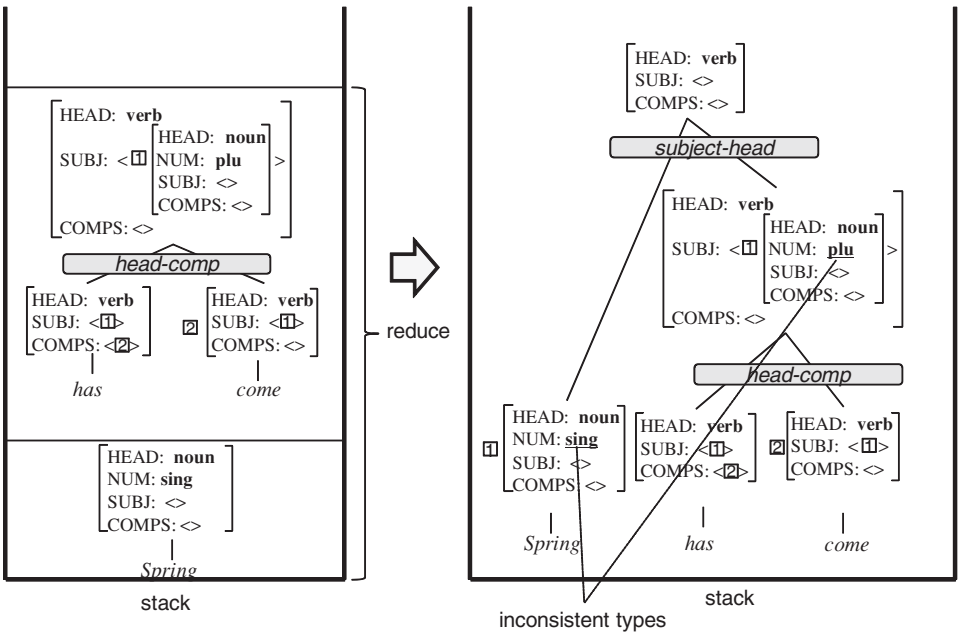


Fig. 5. Example of parsing failure.

This simple mechanism greatly improves the recall of deterministic parsing with normal unification even though the output is not a fully connected parse trees but partial parse trees. Note that proper grammatical constraints are enforced when using normal unification.

5.2 Shift-reduce parsing by backtracking and best-first search

Backtracking is one possible solution for recovering from parsing failure. When parsing fails or ends with nonsentential success, the parser's state reverts to some previous state (backtracking), and it chooses the second-best action and tries parsing again. A parser's state consists of a tree stack, a word queue – the same stack and queue as explained in the shift-reduce parsing algorithm – and a probability, which are sufficient information to restart parsing from any point of deterministic shift-reduce parsing. There are several metrics to choose a previous state as a restarting point, e.g., the smallest probability for transition, the smallest difference or ratio of probabilities for the top-ranked and second-ranked transitions, or the largest entropy in the transition candidates. We used the difference in probability between the top-ranked and the second-ranked transition candidates as the metric in this study, and the previous state with the smallest difference in probability was selected. Let us define the maximum number of backtracking steps while parsing a sentence. Backtracking repeats until parsing finishes with sentential success or reaches the maximum number of backtracking steps. When parsing finishes with sentential success, a fully connected parse tree is output. However, when parsing finishes, because the maximum number of backtracking steps has been reached, a

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

fully connected nonsentential parse tree or a partial parse tree is output. When a partial parse tree is output, recall will greatly decrease because words located at the right side of the partial parse tree remain unanalyzed. Therefore, we let the parser continue parsing from the right side of the partial parse tree until a partial parse tree is reached at the end of the sentence. If parsing fails to find a fully connected parse tree, then the best continuous partial parse trees are output for evaluation. Parsing involves the following five specific steps:

(1) Let $P := \emptyset$.
(2) Do deterministic parsing with normal unification/default unification.
- If a fully connected sentential parse tree is output, then return it.
- Add the output to P.
- If the number of backtracking steps reaches its maximum number, then go to 3.
- Backtrack parsing state, and go to 2.

(3) If the right-most partial parse tree reaches the end of the sentence, then return the best continuous partial parse trees from P.
(4) Clear the stack, and continue parsing from the next position in the right-most partial parse tree.
(5) Go to 2.

Note that this algorithm can be used for both deterministic parsing with normal unification and default unification.

From the viewpoint of search algorithms, parsing with backtracking is a sort of depth-first search algorithm. Another possibility is to use the best-first search algorithm. The best-first parser has a state priority queue. Parsing continues by applying shift-reduce actions to the best state in the state queue. First, the best state is removed from the state queue, and then shift-reduce actions are applied to the state. The newly generated states as results of the shift-reduce actions are placed in the queue. This process repeats until it generates a state satisfying the sentential condition. We define the score of a parsing state as the product of the probabilities of selecting actions that have been taken to reach the state. We regard the state score as the objective function in the best-first search algorithm, i.e., the state with the highest score is always chosen in the algorithm. However, the best-first algorithm with this objective function searches as done by the breadth-first search; a shorter search path tends to have a higher score as it is the product of the probabilities, and therefore the best-first algorithm tends to choose a shorter search path as done by the breadth-first search. Consequently, parsing is very slow or cannot be processed within a reasonable time. Thus, we introduce beam thresholding to the best-first algorithm. The search space is pruned by only adding a new state to the state queue if its score is greater than $1/b$ of the score of the best state in the states that had the same number of shift-reduce actions. We also used the iterative parsing algorithm (Tsuruoka and Tsujii 2005) for the best-first search algorithm. It first starts parsing with a narrow beam. If parsing fails, then the beam is widened, and parsing continues until the parser outputs results or the beam width reaches some limit.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

We tested both backtracking and the best-first search with default unification or normal unification in the experiments. Note that the best-first search parsing for unification-based grammars was much slower than the shift-reduce CFG parsing with the best-first search. This was because we had to copy parse trees, which consisted of large feature structures, in every step of searching to retain many states in the state queue. Copying is not necessary in backtracking, and the original feature structures can be restored by rolling back (trailing) the operations applied to the feature structures.

5.3 Estimation of model parameters

We trained two models of multi-class logistic regression for selecting shift-reduce actions in the experiments. The first was trained using normal unification, and the second was trained using default unification. The probability of an action, y, given an input, x, in a multi-class logistic regression is defined as

$$p(y|\mathbf{x}) = \frac{\exp \sum_u w_u f_u(\mathbf{x}, y)}{\sum_{y' \in Y(\mathbf{x})} \exp \sum_u w_u f_u(\mathbf{x}, y')}, \tag{8}$$

where f_u is a feature function, w_u is a model parameter (a weight for f_u), and $Y(\mathbf{x})$ is a set of all possible candidates for $\mathbf{x}$. Multi-class logistic regression is a kind of supervised learning; given a set of inputs and correct answers to these, the model parameters are estimated by using methods of numerical optimization (Malouf 2002) to maximize the log-likelihood of the training data. We used an HPSG treebank (Miyao *et al.* 2005) in the experiments to train the classifiers. The correct answer was provided by normal unification, but we needed two classifiers trained with normal and default unifications. This was because the sets of all possible candidates were different in using normal unification and default unification, i.e., the denominators in (8) were different. For example, suppose that we acquire ten candidates by using normal unification and forty candidates in default unification for some input $\mathbf{x}$. $|Y(\mathbf{x})|$ is 10 in normal unification and $|Y(\mathbf{x})|$ is 40 in default unification. Consequently, the probabilities, $p(y|\mathbf{x})$, are different, and hence the optimal points of the model parameters are different. The accuracy of deterministic shift-reduce parsing dropped by around 2 per cent points in our preliminary experiments, when we trained the model with normal unification, and used default unification in the test run, compared to deterministic shift-reduce parsing using default unification in both the training and test runs.

6 Experiments

We mainly carried out two sets of experiments; preliminary experiments to test different default unifications and different settings for default feature structures (Section 6.2), and main experiments to evaluate our new approach with different search strategies (Section 6.3). A simply typed version of Copestake's default unification with the default non-head daughter achieved the best accuracy in the preliminary experiments. After obtaining the results for the preliminary experiments,

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

we only tested the simply typed version of Copestake's default unification with the default non-head daughter in the main experiments. Section 6.1 explains the settings and evaluation metrics for the experiments. Section 6.4 discusses the differences between our parsing scheme and Matsuzaki *et al.*'s parser (Matsuzaki, Miyao and Tsujii 2007). Section 6.5 provides analyses of our parsers, including the profiles of parsing, the path values overwritten by default unification, error analysis, and an evaluation of the effects of global features.

6.1 Settings

We evaluated the speed and accuracy of parsing with Enju 2.3β, an HPSG for English (Miyao *et al.* 2005). The lexicon for the grammar was extracted from Sections 02–21 of the Penn Treebank (39,832 sentences). The grammar consisted of twelve grammar rules and 2,302 lexical entries for 11,187 words. We used multi-class logistic regression (also known as the maximum entropy or log-linear model) as classifiers. Two probabilistic classifiers for selecting shift-reduce actions were trained using the same portion of the treebank. The first was trained using normal unification, and the second was trained using default unification. The model trained with normal unification was used for evaluating the test run with normal unification, and the model trained with default unification was used for evaluating the test run with default unification.

We measured the accuracy of output for the predicate–argument relations of the parser. A predicate–argument relation is defined as a tuple $\langle \sigma, w_h, a, w_a \rangle$, where σ is the predicate type (e.g., adjective, intransitive verb), w_h is the head word of the predicate, a is the argument label (MODARG, ARG1, ..., ARG4), and w_a is the head word of the argument. Labeled precision (LP)/labeled recall (LR) is the ratio of tuples correctly identified by the parser, and the labeled F-score (LF) is the harmonic mean of the LP and LR. This evaluation scheme was the same as that used in previous evaluations of lexicalized grammars (Hockenmaier 2003; Clark and Curran 2004b; Miyao and Tsujii 2005). The experiments were conducted on an Intel Xeon 5160 server with 3.0 GHz CPUs. Section 22 of the Penn Treebank was used as the development set, and the accuracy was evaluated using sentences of ≤ 100 words in Section 23 of the Penn Treebank.

6.2 Preliminary experiments

We did preliminary experiments to evaluate parsing accuracy using Ninomiya *et al.*'s (2002) default unification and a simply typed version of Copestake's default unification with several ways of applying binary rules as $(R \sqcup H) \overleftarrow{\sqcup} NH$, $(R \sqcup NH) \overleftarrow{\sqcup} H$, and $(H \sqcup NH) \overleftarrow{\sqcup} R$, where R is a grammar rule, H is a head-daughter, and NH is a non-head daughter. We measured the accuracy of output for the predicate–argument relations of the deterministic parser with Enju 2.3β for the Penn Treebank Section 22 (development set) with gold POS tags.

Table 3 lists the results from these preliminary experiments. The '$\overleftarrow{\sqcup}_{d'}$' means a simply typed version of Copestake's default unification and '$\overleftarrow{\sqcup}_{n}$' means Ninomiya

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 3. *Preliminary experiments with Ninomiya et al.'s and Copestake's default unification cases (Penn Treebank Section 22 with gold POS tags)*

	LP (%)	LR (%)	LF (%)
$(R \sqcup H) \overset{<}{\sqcup}_{a'} NH$	87.75	87.47	87.61
$(R \sqcup NH) \overset{<}{\sqcup}_{a'} H$	84.96	84.73	84.84
$(H \sqcup NH) \overset{<}{\sqcup}_{a'} R$	85.60	85.35	85.48
$(R \sqcup H) \overset{<}{\sqcup}_{n} NH$	87.49	87.22	87.35
$(R \sqcup NH) \overset{<}{\sqcup}_{n} H$	84.61	84.47	84.54
$(H \sqcup NH) \overset{<}{\sqcup}_{n} R$	83.47	83.36	83.41

Table 4. *Experimental results for section 23 (gold POS tags)*

	Normal unification				Default unification			
	LP (%)	LR (%)	LF (%)	Time (ms)	LP (%)	LR (%)	LF (%)	Time (ms)
(Miyao and Tsujii 2005)	87.26	86.50	86.88	604	–	–	–	–
(Ninomiya *et al.* 2007)	89.78	89.28	89.53	234	–	–	–	–
Deterministic	76.39	81.97	79.08	115	87.73	87.40	87.56	261
Backtrack	81.59	85.11	83.31	494	87.93	87.58	87.75	300
Best-first	86.23	87.79	87.01	512	88.62	88.22	88.42	465
Best-first (wide)	89.95	89.90	89.92	14,760	–	–	–	–

et al.'s (2002) default unifications. From the table, we can see that the accuracy of parsing with the default non-head daughters was better than that with the default head daughters and that with the default rules in both Ninomiya *et al.*'s and Copestake's default unification cases. The labeled F-score dropped by 2.8 points when the defaults were switched from non-head daughters to head daughters. Of these methods, the simply typed version of Copestake's default unification with the default non-head daughter, $(R \sqcup H) \overset{<}{\sqcup}_{a'} NH$, achieved the best accuracy. We only tested $(R \sqcup H) \overset{<}{\sqcup}_{a'} NH$ in the main experiments.

6.3 Evaluation

We evaluated the speed and accuracy of parsing Section 23 of the Penn Treebank, using Enju 2.3β and a simply typed version of Copestake's default unification with the default non-head daughter. Tables 4 and 5 list the results of parsing Section 23. 'Time' in the table is the average parsing time for the tested sentences. 'Deterministic' means the deterministic shift-reduce parsing proposed in this paper. 'Backtrack' means shift-reduce parsing with backtracking. 'Best-first' means the best-first shift-reduce parsing with beam thresholding. 'Default unification' indicates that default unification was used, and 'normal unification' means that default unification was not used. Table 4 lists the results obtained using gold POS tags, while Table 5

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 5. *Experimental results for section 23 (automatic POS tags)*

	Normal unification				Default unification			
	LP (%)	LR (%)	LF (%)	Time (ms)	LP (%)	LR (%)	LF (%)	Time (ms)
(Miyao and Tsujii 2005)	84.96	84.25	84.60	674	–	–	–	–
(Ninomiya *et al.* 2007)	87.28	87.05	87.17	260	–	–	–	–
(Matsuzaki *et al.* 2007)	86.93	86.47	86.70	30	–	–	–	–
(Sagae, Miyao, Tsujii 2007)	88.50	88.00	88.20	–	–	–	–	–
Deterministic	74.20	80.06	77.02	113	85.84	85.63	85.74	262
Backtrack	78.64	82.77	80.65	541	86.06	85.83	85.94	311
Best-first	83.75	85.74	84.73	529	86.57	86.34	86.46	565
Best-first(wide)	87.66	87.85	87.75	19,154	–	–	–	–

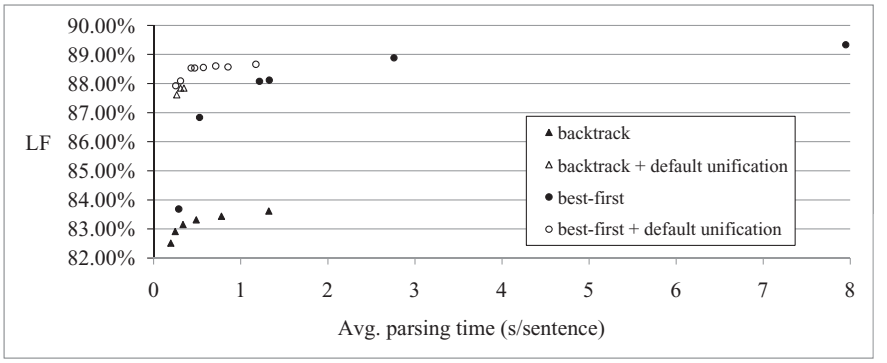


Fig. 6. Relation between LF and the average parsing time (section 22, gold POS tags).

lists the results obtained using an automatic POS tagger. The maximum number of backtracking steps and the beam threshold were determined by observing the speed and accuracy for the development set (Section 22) such that the LF was maximized with a parsing time of less than 500 ms/sentence (except 'best-first(wide)'). The maximum number of backtracks was forty for normal unification and twenty for default unification. The beam width, b, was 7.39 for normal unification, and 20.09 for default unification. The accuracy of 'best-first(wide)' was evaluated to find the limit of accuracy of the best-first parsing, and beam width b was set to 403.4.

Deterministic parsing with normal unification achieved accuracy with an LF of around 79.1 per cent (Section 23, gold POS tags).[2] The LF increased to 83.3 per cent with backtracking. Figure 6 shows the relation between LF and parsing time

[2] Some readers may be confused with this result as we have claimed that deterministic parsing with normal unification can easily fail, and consequently recall should decrease greatly. This is because when deterministic parsing with normal unification fails parsing, it outputs partial parse trees instead of a fully connected parse tree, and it continues parsing the rest of the sentence. Therefore, the parser outputs continuous partial parse trees for each sentence, and the predicate–argument relations in these partial parse trees are evaluated to measure precision and recall. See Section 5.1 for details.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

for the development set (Section 22, gold POS tags). As seen in the figure, the LF increased with increase in the parsing time. The increase in LF for deterministic parsing with normal unification, however, seems to have saturated at around 83.5 per cent. Table 4 also indicates that deterministic parsing with default unification achieved greater accuracy, with an LF of around 87.6 per cent (Section 23, gold POS tags) without backtracking. Default unification was more effective; it ran faster and achieved better accuracy than deterministic parsing with normal unification. The best-first parsing without default unification achieved a high degree of accuracy, with an LF of around 87.0 per cent, but it was still worse than deterministic parsing with default unification. However, with default unification, it achieved the best accuracy, with an LF of around 88.4 per cent, in settings with parsing time of less than 500 ms/sentence for Section 22.

The speed and accuracy of Miyao and Tsujii's (2005), Ninomiya *et al.*'s (2007), Matsuzaki *et al.*'s (2007), and Sagae *et al.*'s (2007) parsers have also been listed in Tables 4 and 5 for the purpose of cross referencing. Miyao and Tsujii's (2005) parser is based on a probabilistic model only estimated with a feature forest. Ninomiya *et al.*'s (2007) parser is a mixture of the feature forest and an HPSG supertagger. Sagae *et al.*'s (2007) parser is a hybrid parser with a shallow dependency parser. As seen in the table, our deterministic/best-first parser with default unification achieved an accuracy that was between Miyao and Tsujii's (2005) and Ninomiya *et al.*'s (2007) parsers. Note that these have been listed for the purpose of cross referencing and not to stress the superiority of the packed-forest-based parsing or the deterministic/best-first parsing. The main focus here has been to demonstrate to what extent simple deterministic/best-first parsing could reach state-of-the-art parsers, which have basically been developed with sophisticated parsing technologies based on the packed parse forest. The table also lists the average parsing time, and although our deterministic parser was competitive with Ninomiya *et al.*'s (2007) parser, it was slower than Matsuzaki *et al.*'s (2007) parser. Also note that these were for cross referencing. The difference in parsing speed was mainly due to the difference in implementation and technologies employed for speedups. Parsing time is particularly sensitive to implementation, but our implementation is currently not fully optimized for deterministic parsing with default unification. There is a detailed discussion on Matsuzaki *et al.*'s (2007) parser in Section 6.4, and the profiles on parsing time are provided in Section 6.5.

Our method with the setting of the 'best-first(wide)' provided interesting results. The 'best-first(wide)' in Table 4 and the 'best-first' in Figure 6 indicate great possibilities for beam-search parsing. The 'best-first(wide)' was very slow, but its accuracy was higher than that of any of the other parsers except for Sagae *et al.*'s (2007) parser.

6.4 Discussion

Matsuzaki *et al.* (2007) developed a quite efficient HPSG parser by using a technique called CFG-filtering. All possible lexical assignments (i.e., selection of lexical entries) were scored by a conditional probability model, and then a combination of highly

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 6. *Profiles for section 22*

	#Backtrack		States		#Failure		#Non-sent		#Success	
	NU	DU	NU	DU	NU	DU	NU	DU	NU	DU
Deterministic	0	0	–	–	589	0	10	74	1,100	1,625
Backtrack	13,237	447	–	–	275	0	13	9	1,411	1,690
Best-first	–	–	389,044	359,994	256	0	13	14	1,430	1,685
Best-first(wide)	–	–	9,033,947	–	53	–	4	–	1,642	–

scored lexical assignments, which satisfied most grammatical constraints, was selected by applying a dynamic-programming-based parsing algorithm to the scored lexical entries, with a CFG that approximated the original HPSG. As such, they could only use 'local' features in their lexical assignment model because the joint probability of a combination of lexical assignments was calculated in the dynamic-programming-based CFG parser as a product of the assignment probabilities for each word. This is in good contrast with our lexical assignment model, which is integrated in the shift-reduce parsing process, wherein we can virtually use all types of features taken from partial parse results that have been built up to the lexical-entry assignment step in shift operation. As the accuracy of lexical assignments largely determines the overall accuracy of parse disambiguation (Ninomiya *et al.* 2007), we consider the higher accuracy with our method in the 'best-first (wide)' setting than that with Matsuzaki *et al.*'s (2007) approach is mainly due to our richer model of lexical assignment. Matsuzaki *et al.*'s parser mainly achieved its efficiency by replacing unification operations (or more precisely, their approximation) with CFG rules. They obtained the CFG rules by regarding a feature structure as an atomic symbol, ignoring several of its feature values to obtain a finite set of symbols. Default unification had a similar effect for CFG approximation, as some (conflicting) feature values were ignored to approximate the full search space of parsing to a smaller one. We expect to achieve a similar speedup for Matsuzaki *et al.*'s (2007) method by introducing the technique of CFG approximation to our method, while still enjoying flexibility in feature design due to the truly deterministic nature of our method, even though we have left this for future work.

6.5 Analysis

Table 6 lists the profiles of the total number of backtracks, states, parsing failures, and parsing successes for Section 22. The '#backtrack' is the total number of backtracking steps that occurred during parsing. The '#states' is the total number of states for the sentences in Section 22. The '#failure' is the number of sentences for which parsing failed. The '#non-sent' is the number of sentences for which parsing succeeded but did not generate a parse tree that satisfied the sentential condition. The '#success' is the number of sentences for which parsing succeeded in generating a parse tree that satisfied the sentential condition. 'NU' means that normal unification was used, and 'DU' means that default unification was used. The

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 7. *Path values overwritten by default unification in section 22*

Path	Strict type	Default type	Freq
SYNSEM:LOCAL:CAT:HEAD:MOD:	cons	nil	434
SYNSEM:LOCAL:CAT:HEAD:MOD:hd:CAT:HEAD:MOD:	cons	nil	237
SYNSEM:LOCAL:CAT:VAL:SUBJ:	nil	cons	231
SYNSEM:LOCAL:CAT:HEAD:MOD:hd:CAT:VAL:SUBJ:	nil	cons	125
SYNSEM:LOCAL:CAT:HEAD:	verb	noun	110
SYNSEM:LOCAL:CAT:VAL:SPR:hd:LOCAL:CAT:VAL:SPEC:hd:LOCAL:CAT:HEAD:MOD:	cons	nil	101
SYNSEM:LOCAL:CAT:HEAD:MOD:hd:CAT:VAL:SPR:hd:LOCAL:CAT:VAL:SPEC:hd:LOCAL:CAT:HEAD:MOD:	cons	nil	96
SYNSEM:LOCAL:CAT:HEAD:MOD:	nil	cons	92
SYNSEM:LOCAL:CAT:HEAD:MOD:hd:CAT:HEAD:	verb	noun	91
SYNSEM:LOCAL:CAT:VAL:SUBJ:	cons	nil	79
SYNSEM:LOCAL:CAT:HEAD:	noun	verbal	77
SYNSEM:LOCAL:CAT:HEAD:MOD:hd:CAT:HEAD:	noun	verbal	77
SYNSEM:LOCAL:CAT:HEAD:	nominal	verb	75
SYNSEM:LOCAL:CAT:VAL:CONJ:hd:LOCAL:CAT:HEAD:MOD:	cons	nil	74
SYNSEM:LOCAL:CAT:VAL:CONJ:tl:hd:LOCAL:CAT:HEAD: MOD:	cons	nil	69
SYNSEM:LOCAL:CAT:VAL:CONJ:tl:hd:LOCAL:CAT:VAL:SUBJ:	nil	cons	64
SYNSEM:LOCAL:CAT:VAL:CONJ:hd:LOCAL:CAT:VAL:SUBJ:	nil	cons	64
SYNSEM:LOCAL:CAT:VAL:COMPS:hd:LOCAL:CAT:HEAD:	nominal	verb	63
SYNSEM:LOCAL:CAT:HEAD:MOD:hd:CAT:VAL:SUBJ:	cons	nil	63
...	...	...	...
Total			10,598

number of parsing failures reached zero and the number of sentences with sentential success increased, when default unification was used.

Table 7 lists the behaviors of default unification for 'deterministic' with 'default unification.' The table has the twenty most frequent path values that were overwritten by default unification in Section 22. In most of these, the overwritten path values were in the selection features, i.e., subcategorization frames (COMPS:, SUBJ:, SPR:, and CONJ:) and the modifiee specification (MOD:). The column with 'Default type' indicates the default types that were overwritten by strict types in the 'strict type' column, and the last column is the frequency of overwriting. The 'cons' means a non-empty list, and 'nil' means an empty list. In most of these, modifiee and subcategorization frames were changed from empty to non-empty and vice versa. The table also indicates, head information was overwritten, e.g., a 'noun' was changed to a 'verb.'

We analyzed the predicate and argument types that were incorrectly identified by the parsers. Table 8 has the mapping of predicates and arguments to categories. Tables 9, 10, 11, and 12 have the categories of predicates and arguments that were incorrectly identified by Ninomiya *et al.*'s (2007) parser, the deterministic shift-reduce parser, the shift-reduce parser with backtracking, and the best-first shift-reduce parser, respectively (Section 22, gold POS tags). We first enumerated all predicate–argument pairs that were incorrectly identified by the parser, and then enumerated pairs of a gold argument and incorrectly identified argument for each predicate. In the tables, 'pred' means predicate types, 'arg' means argument types in the gold relations, 'errarg' means argument types output by the parser, and 'pred-arg[errarg]'

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 8. *Mapping of categories for predicates and arguments in error analysis*

root	Root node of the sentence
undef	An empty argument node that does not exist in the surface sentence.
na	An argument node whose arc exists in the gold relations but was not identified by the parser.
verb	Verb category. Corresponding POS tags are VB, VBD, VBG, VBN, VBP, and VBZ.
auxv	Auxiliary verb category. Corresponding POS tag is MD.
noun	Noun category. Corresponding POS tags are CD, NN, NNP, NNPS, NNS, $, #, FW, and LS.
pron	Pronoun category. Corresponding POS tags are PRP, PRP$, and EX.
adj	Adjective category. Corresponding POS tags are JJ, JJR, JJS, and PDT.
coord	Coordination category. Corresponding POS tag is CC.
det	Article category. Corresponding POS tag is DT.
prep	Preposition category. Corresponding POS tag is IN.
adverb	Adverb category. Corresponding POS tags are RB, RBR, RBS and RP.
to	Word of 'to' category. Corresponding POS tag is TO.
uh	Interjection category. Corresponding POS tag is UH.
wh	Wh-word category. Corresponding POS tags are WDT, WP, WP$, and WRB.
comma	Comma category. Corresponding words are , and :.
paren	Parenthesis category. Corresponding words are “, ”, (, and)
poss	Possesive word category ('s). Corresponding POS tag is POS.

Table 9. *Incorrectly identified predicate and argument types (Ninomiya et al.'s parser, Section 22, gold POS tags)*

pred (#)	arg (#)	errarg (#)	pred–arg[errarg] (#)
verb (1,033)	noun (1,719)	noun (1,615)	verb–noun[noun] (365)
prep (741)	verb (1,234)	verb (1,077)	prep–noun[noun] (155)
comma (439)	undef (210)	na (518)	comma–noun[noun] (153)
to (304)	adj (191)	undef (182)	prep–verb[noun] (142)
noun (278)	pron (103)	pron (95)	prep–verb[verb] (135)
adverb (191)	prep (101)	adj (71)	prep–noun[verb] (122)
coord (189)	adverb (91)	adverb (55)	comma–verb[verb] (122)
paren (182)	wh (27)	prep (53)	paren–verb[verb] (92)
adj (147)	det (25)	wh (39)	to–verb[verb] (78)
wh (64)	to (16)	det (17)	verb–noun[undef] (72)
det (61)	paren (9)	to (6)	verb–verb[na] (72)
root (52)	auxv (7)	auxv (6)	coord–noun[noun] (71)
auxv (32)	poss (5)	coord (4)	to–noun[noun] (68)
pron (14)	coord (2)	uh (2)	verb–verb[verb] (65)
poss (13)	comma (2)	paren (2)	verb–undef[noun] (65)
uh (2)			. . .
Total (3,742)	Total (3,742)	Total (3,742)	Total (3,742)

means triples of a predicate, gold argument, and incorrectly identified argument. The numbers in brackets are the frequencies of each error. The tables indicate that the effect of default unification was the reduced number of undefined arguments ('undef'

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 10. *Incorrectly identified predicate and argument types (deterministic, section 22, gold POS tags)*

Deterministic, normal unification			
pred (#)	arg (#)	errarg (#)	pred–arg[errarg] (#)
verb (2,506)	noun (3,347)	undef (3,632)	verb–noun[undef] (996)
prep (1,229)	verb (2,614)	noun (1,492)	verb–noun[noun] (370)
comma (783)	adj (289)	verb (931)	verb–verb[undef] (307)
coord (463)	pron (278)	na (718)	prep–verb[undef] (304)
to (427)	undef (198)	pron (102)	comma–verb[undef] (217)
adverb (338)	prep (135)	adj (90)	prep–noun[undef] (196)
noun (334)	adverb (130)	adverb (65)	coord–verb[undef] (176)
paren (311)	det (52)	prep (56)	prep–verb[noun] (145)
adj (212)	wh (45)	wh (26)	comma–noun[noun] (138)
det (180)	paren (18)	det (19)	prep–noun[noun] (137)
auxv (169)	to (15)	coord (6)	verb–pron[undef] (136)
wh (115)	comma (10)	paren (5)	prep–noun[verb] (122)
root (36)	poss (8)	to (3)	comma–verb[verb] (119)
pron (24)	auxv (7)	auxv (2)	verb–verb[na] (109)
poss (19)	coord (1)	uh (1)	prep–verb[verb] (108)
uh (2)	uh (1)		. . .
Total (71,48)	Total (7,148)	Total (7,148)	Total (7,148)
Deterministic, default unification			
pred (#)	arg (#)	errarg (#)	pred–arg[errarg] (#)
verb (1,637)	noun (2,373)	noun (1,813)	verb–noun[noun] (475)
prep (868)	verb (1,728)	verb (1,310)	verb–noun[undef] (318)
comma (577)	undef (227)	undef (757)	prep–verb[noun] (169)
to (343)	adj (227)	na (707)	prep–noun[noun] (155)
noun (322)	pron (161)	pron (128)	comma–noun[noun] (152)
coord (261)	adverb (117)	prep (102)	prep–noun[verb] (147)
adverb (259)	prep (116)	adj (94)	prep–verb[verb] (139)
paren (242)	wh (36)	adverb (65)	comma–verb[verb] (131)
adj (187)	det (34)	wh (38)	paren–verb[verb] (115)
root (126)	to (20)	det (36)	verb–verb[na] (108)
det (87)	paren (14)	comma (9)	to–verb[verb] (88)
wh (79)	poss (9)	paren (6)	verb–noun[na] (86)
auxv (57)	comma (9)	coord (5)	noun–verb[na] (85)
pron (19)	coord (4)	poss (3)	verb–verb[verb] (81)
poss (13)	auxv (3)	to (3)	coord–noun[noun] (80)
uh (2)	uh (1)	auxv (2)	noun–noun[na] (77)
		uh (1)	. . .
Total (5,079)	Total (5,079)	Total (5,079)	Total (5,079)

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 11. *Incorrectly identified predicate and argument types (backtrack, section 22, gold POS tags)*

Backtrack, normal unification			
pred (#)	arg (#)	errarg (#)	pred–arg[errarg] (#)
verb (1,993)	noun (2,810)	undef (2,410)	verb–noun[undef] (621)
prep (1,070)	verb (2,102)	noun (1,514)	verb–noun[noun] (397)
comma (641)	adj (257)	verb (1,038)	verb–verb[undef] (202)
to (385)	undef (211)	na (677)	prep–verb[undef] (191)
coord (352)	pron (210)	pronoun (98)	prep–noun[undef] (156)
noun (319)	adverb (121)	adj (82)	prep–noun[verb] (136)
adverb (312)	prep (120)	adverb (56)	prep–noun[noun] (136)
paren (268)	wh (48)	prep (49)	comma–noun[noun] (135)
adj (185)	det (42)	wh (27)	prep–verb[noun] (127)
det (158)	paren (18)	det (17)	prep–verb[verb] (124)
auxv (123)	to (15)	coord (4)	comma–verb[verb] (123)
wh (102)	comma (10)	to (4)	comma–verb[undef] (122)
root (37)	auxv (8)	paren (4)	verb–verb[na] (105)
pron (20)	poss (5)	uh (1)	coord–verb[undef] (103)
poss (15)	coord (4)	auxv (1)	paren–verb[verb] (100)
uh (2)	uh (1)		. . .
Total (5,982)	Total (5,982)	Total (5,982)	Total (5,982)

Backtrack, default unification			
pred (#)	arg (#)	errarg (#)	pred–arg[errarg] (#)
verb (1,601)	noun (2,334)	noun (1,801)	verb–noun[noun] (474)
prep (852)	verb (1,680)	verb (1,287)	verb–noun[undef] (293)
comma (566)	adj (227)	undef (713)	prep–verb[noun] (165)
to (341)	undef (223)	na (704)	prep–noun[noun] (154)
noun (320)	pron (161)	pron (128)	comma–noun[noun] (153)
coord (259)	adverb (117)	prep (97)	prep–noun[verb] (146)
adverb (255)	prep (114)	adj (95)	prep–verb[verb] (136)
paren (238)	wh (35)	adverb (60)	comma–verb[verb] (124)
adj (183)	det (34)	wh (38)	paren–verb[verb] (112)
root (115)	to (20)	det (36)	verb–verb[na] (110)
det (86)	paren (14)	comma (8)	to–verb[verb] (88)
wh (80)	poss (9)	coord (5)	verb–noun[na] (84)
auxv (55)	comma (9)	paren (5)	noun–verb[na] (82)
pron (19)	coord (4)	poss (3)	verb–verb[verb] (80)
poss (13)	auxv (3)	to (2)	coord–noun[noun] (80)
uh (2)	uh (1)	auxv (2)	noun–noun[na] (77)
		uh (1)	. . .
Total (4,985)	Total (4,985)	Total (4,985)	Total (4,985)

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 12. *Incorrectly identified predicate and argument types (best-first, section 22, gold POS tags)*

Best-first, normal unification			
pred (#)	arg (#)	errarg (#)	pred–arg[errarg] (#)
verb (1,609)	noun (2,289)	noun (1,564)	verb–noun[noun] (416)
prep (906)	verb (1759)	undef (1297)	verb–noun[undef] (392)
comma (563)	adj (222)	verb (1,072)	prep–noun[verb] (145)
to (327)	undef (208)	na (646)	prep–verb[noun] (143)
noun (294)	pron (145)	pron (110)	comma–noun[noun] (141)
coord (265)	prep (113)	adj (78)	comma–verb[verb] (130)
adverb (260)	adverb (101)	adverb (75)	prep–noun[noun] (130)
paren (244)	wh (34)	prep (52)	prep–verb[verb] (128)
adj (159)	det (30)	wh (26)	verb–verb[undef] (118)
det (95)	paren (18)	det (16)	prep–verb[undef] (111)
wh (83)	to (16)	paren (9)	verb–verb[na] (100)
auxv (76)	comma (8)	coord (5)	paren–verb[verb] (97)
root (46)	poss (7)	to (4)	to–verb[verb] (82)
pron (18)	auxv (7)	auxv (4)	noun–verb[na] (77)
poss (12)	coord (1)		verb–noun[na] (73)
uh (1)			. . .
Total (4,958)	Total (4,958)	Total (4,958)	Total (4,958)
Best-first, default unification			
pred (#)	arg (#)	errarg (#)	pred–arg[errarg] (#)
verb (1507)	noun (2148)	noun (1741)	verb-noun[noun] (463)
prep (824)	verb (1647)	verb (1270)	verb-noun[undef] (238)
comma (509)	undef (226)	na (665)	prep-verb[noun] (167)
to (328)	adj (216)	undef (569)	prep-noun[verb] (148)
noun (295)	pron (143)	pron (125)	prep-noun[noun] (146)
coord (252)	adverb (110)	adj (88)	comma-noun[noun] (140)
paren (242)	prep (99)	prep(79)	comma-verb[verb] (136)
adverb (236)	wh (36)	adverb (62)	prep-verb[verb] (132)
adj (169)	det (27)	wh (35)	paren-verb[verb] (117)
root (113)	paren (18)	det (31)	verb-verb[na] (103)
det (78)	to (15)	coord (17)	to-verb[verb] (88)
wh (74)	comma (8)	comma (8)	verb-noun[na] (87)
auxv (49)	poss (7)	to (5)	verb-verb[verb] (81)
pron (18)	auxv (7)	paren (5)	noun-verb[na] (79)
poss (13)	coord (1)	poss (3)	noun-noun[na] (68)
uh (1)		auxv (3)	coord-noun[noun] (64)
		uh (2)	. . .
Total (4,708)	Total (4,708)	Total (4,708)	Total (4,708)

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Table 13. *Profiles of deterministic parsing (Section 22, gold POS tags)*

	Parsing main (ms)	Calc. of prob. (ms)	DU (ms)	Total (ms)
Normal unification	22.4 (18.8%)	97.0 (81.2%)	—	119.4 (100.0%)
Default unification	41.7 (15.7%)	133.5 (50.2%)	90.8 (34.1%)	266.1 (100.0%)

in the table), which means that normal unification failed to identify predicate–argument relations at many places due to hard constraints, and default unification outputs some analysis rather than 'undef.' Default unification, on the other hand, tended to output incorrect labels for each 'errarg' type. This implies that default unification outputs some analysis rather than 'undef,' but its analysis tended to be incorrect. However, in total, default unification outputs better analysis than normal unification.

Table 13 lists profiles of deterministic shift-reduce parsing for Section 22 with gold POS tags. The 'calc. of prob.' indicates the average time per sentence for calculating probabilities, including the time for extracting features from feature structures. 'DU' is the average time for processing default unification. The 'parsing main' is the total time excluding the time for calculating probabilities and processing default unification. The 'parsing main' mainly consists of the time for stack and queue operations, and lexical entry look-up and unification operation. As can be seen from the profile, most of the time was spent on calculating probabilities. This is because our parser has many features, i.e., sixty-two feature templates and 155 combinations of feature templates. If we could reduce the time for calculating probabilities, the time for parsing with normal unification would be closer to the that for Matsuzaki *et al.*'s (2007) parser. In parsing with default unification, much of the time was spent on default unification. This is partly because default unification generated more candidates for classifiers than normal unification because default unification almost always succeeded. Consequently, more time needs to be spent on default unification than normal unification, as well as on calculating the probabilities.

We also evaluated the effectiveness of the global features. There were fifty-two feature template combinations, including global features (the total number of feature template combinations was 155). We extracted each global feature from the full set of features in deterministic shift-reduce parsing with default unification. Each feature contributed to an accuracy from 0.0003 per cent points to 0.14 per cent points, and 0.029 per cent points on average. When we removed all global features, the F-score dropped by 2.9 per cent points from 87.61 to 84.72 per cent.

7 Conclusion and future work

We presented a shift-reduce parsing approach to unification-based grammars, based on deterministic shift-reduce parsing. Deterministic parsing was difficult in the framework of unification-based grammar parsing, which often fails because of its hard constraints. We introduced default unification to avoid parsing failures. Our experimental results demonstrated the effectiveness of deterministic parsing with

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

default unification. The experiments revealed that deterministic parsing with default unification achieved a high degree of accuracy, with a labeled LF of 87.6 per cent for Section 23 of the Penn Treebank with gold POS tags. We also presented the best-first parsing with a beam search for unification-based grammars. The best-first parsing with a beam search achieved the best accuracy, with an LF of 87.0 per cent, in settings with normal unification. Default unification further increased LF from 87.0 to 88.4 per cent.

We also tested the best-first parsing by normal unification with a very wide beam, ignoring its efficiency. The best-first parsing achieved the best LF of 89.9 per cent with an average parsing time of around 15 s. Even though this parser cannot be used in practice due to this efficiency, this indicates that our deterministic parsing has the potential to reach state-of-the-art parsers in terms of accuracy due to its truly deterministic nature and flexibility in feature design.

We have not yet exhaustively pursued the combinations of feature templates, although there are many possibilities for these, e.g., more words/POS tags in relative positions, more elements in the parser's stack, and exponentially more possibilities of combinations. Although at present these features are found manually, it is very difficult to find them manually in settings where global features are available like they are in deterministic parsing. Automatic feature construction/selection that has been studied in the field of machine learning, such as grafting (Perkins, Lacker, Theiler 2003) should solve this problem, but it is also known that it can easily overfit the training data. We have left the problem of efficiency in the best-first parsing with normal unification and the problem of feature construction/selection for future work.

References

Abney, S. P. 1997. Stochastic attribute-value grammars. *Computational Linguistics* **23**(4): 597–618.

Bouma, G. 1990. Defaults in unification grammar. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, PA, pp. 165–172.

Bresnan, J., and Kaplan, R. M. 1982. Introduction:grammars as mental representations of language. In J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*. Cambridge, MA: MIT Press.

Briscoe, T. 1993a. Introduction. In T. Briscoe, V. de Paiva and A. Copestake (eds.), *Inheritance, Defaults and the Lexicon*. Cambridge, UK: Cambridge University Press.

Briscoe, T., and Carroll, J. 1993b. Generalized probabilistic LR-parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics* **19**(1): 25–59.

Butt, M., King, T. H., Niño M.-E., and Segond, F. 1999. *A Grammar Writer's Cookbook*. Stanford, CA: CSLI Publications.

Carpenter, B. 1992. *The Logic of Typed Feature Structures*. Cambridge, UK: Cambridge University Press.

Carpenter, B. 1993. Skeptical and credulous default unification with applications to templates and inheritance. In T. Briscoe, V. de Paiva and A. Copestake (eds.), *Inheritance, Defaults and the Lexicon*. Cambridge, UK: Cambridge University Press.

Charniak, E., and Johnson, M. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, MI, pp. 173–180.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Clark, S., and Curran, J. R. 2004a The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, Geneva, Switzerland, pp. 282–288.

Clark, S., and Curran, J. R. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain, pp. 103–110.

Collins, M. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning*, Stanford, CA, pp.175–182.

Collins, M., and Roark, B. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain, pp. 111–118.

Copestake, A. 1993. Defaults in lexical representation. In T. Briscoe, V. de Paiva, and A. Copestake (eds.), *Inheritance, Defaults and the Lexicon*. Cambridge, UK: Cambridge University Press.

Daumé III, H., and Marcu, D. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *Proceedings of The 22nd International Conference on Machine Learning*, Bonn, Germany, pp. 169–176.

Frank, A., King, T. H., Kuhn, J., and Maxwell III, J. T. 2001. Optimality theory style constraint ranking in large-scale LFG grammars. In P. Sells (ed.), *Formal and Empirical Issues in Optimality-Theoretic Syntax*. Stanford, CA: CSLI, pp. 367–397.

Geman, S., and Johnson, M. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA, pp. 279–286.

Hockenmaier, J. 2003. Parsing with generative models of predicate–argument structure. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, pp. 359–366.

Hockenmaier, J., and Steedman, M. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas, Spain, pp. 1974–1981.

Huang, L. 2008. Forest reranking: discriminative parsing with non-local features. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Columbus, OH, pp. 586–594.

Johnson, M., Geman, S., Canon, S., Chi, Z., and Riezler, S. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, College Park, MD, pp. 535–541.

Kaplan, R. M., Riezler, S., King, T. H., Maxwell III, J. T., and Vasserman, A. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, Boston, MA, pp. 97–104.

Malouf, R. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning*, Taipei, Taiwan, pp. 49–55.

Malouf, R., and van Noord, G. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop: Beyond Shallow Analyses - Formalisms and Statistical Modeling for Deep Analyses*. Hainan Island, China.

Matsuzaki, T., Miyao, Y., and Tsujii, J. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, Hyderabad, India, pp. 1671–1676.

Miyao, Y., Ninomiya, T., and Tsujii, J. 2005. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the Penn Treebank. In K.-Y. Su, J. T., J.-H. Lee, and O. Y. Kwong (eds.), *Natural Language Processing - IJCNLP 2004*, pp. 684–693. Lecture Notes in Artificial Intelligence. Berlin, Germany: Springer-Verlag.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Miyao, Y., and Tsujii, J. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Second International Conference on Human Language Technology Research*, San Diego, CA, pp. 292–297.

Miyao, Y., and Tsujii, J. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, Ann Arbor, MI, pp. 83–90.

Nakagawa, T. 2007 Multilingual dependency parsing using global features. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, Prague, Czech Republic, pp. 952–956.

Ninomiya, T., Matsuzaki, T., Miyao, Y., and Tsujii, J. 2007. A log-linear model with an n-gram reference distribution for accurate HPSG parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, Prague, Czech Republic, pp. 60–68.

Ninomiya, T., Matsuzaki, T., Tsuruoka, Y., Miyao, Y., and Tsujii, J. 2006. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, pp. 155–163.

Ninomiya, T., Miyao, Y., and Tsujii, J. 2002. Lenient default unification for robust processing within unification-based grammar formalisms. In *Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan, pp. 744–750.

Nivre, J., and Scholz, M. 2004. Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics*, Geneva, Switzerland, pp. 64–70.

O'Donovan, R., Burke, M., Cahill, A., van Genabith, J., and Way, A. 2004 Large-scale induction and evaluation of lexical resources from the Penn-II Treebank. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain, pp. 367–374.

Perkins, S., Lacker, K., and Theiler, J. 2003. Grafting: fast, incremental feature selection by gradient decent in function space. *Journal of Machine Learning Research* **3**: 1333–1356.

Pollard, C., and Sag, I. A. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, IL and Stanford, CA: University of Chicago Press and CSLI Publications.

Ratnaparkhi, A. 1997, A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, Providence, RI, pp. 1–10.

Riezler, S., Prescher, D., Kuhn, J., and Johnson, M. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, China, pp. 480–487.

Sagae, K., and Lavie, A. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the 9th International Workshop on Parsing Technology*, Vancouver, Canada, pp. 125–132.

Sagae, K., and Lavie, A. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL on Main Conference Poster Sessions*, Sydney, Australia, pp. 691–698.

Sagae, K., Miyao, Y., and Tsujii, J. 2007. HPSG parsing with shallow dependency constraints. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic, pp. 624–631.

Steedman, M. 2000. *The Syntactic Process*. Cambridge, MA: The MIT Press.

Toutanova, K., Manning, C. D., Shieber, S. M., Flickinger, D., and Oepen, S. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of The First Workshop on Treebanks and Linguistic Theories (TLT2002)*, Sozopol, Bulgaria, pp. 253–263.

Tsuruoka, Y., and Tsujii, J. 2005. Iterative CKY parsing for probabilistic context-free grammars. In K.-Y. Su, J. Tsujii, J.-H. Lee, and O. Y. Kwong (eds.), *Natural Language Processing - IJCNLP 2004*, pp. 52–60. Lecture Notes in Artificial Intelligence. Berlin¡ Germany: Springer-Verlag.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press

Yamada, H., and Matsumoto, Y. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, Nancy, France, pp. 195–206.

Zhang, Y., and Clark, S. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, Paris, France, pp. 162–171.

Zhang, Y., Oepen, S., and Carroll, J. 2007. Efficiency in unification-based N-best parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, Prague, Czech, pp. 48–59.

https://doi.org/10.1017/S1351324910000240 Published online by Cambridge University Press