# Dynamic Programming Algorithms for Transition-Based Dependency Parsers

**Marco Kuhlmann**
Dept. of Linguistics and Philology
Uppsala University, Sweden
marco.kuhlmann@lingfil.uu.se

**Carlos Gómez-Rodríguez**
Departamento de Computación
Universidade da Coruña, Spain
cgomezr@udc.es

**Giorgio Satta**
Dept. of Information Engineering
University of Padua, Italy
satta@dei.unipd.it

## Abstract

We develop a general dynamic programming technique for the tabulation of transition-based dependency parsers, and apply it to obtain novel, polynomial-time algorithms for parsing with the arc-standard and arc-eager models. We also show how to reverse our technique to obtain new transition-based dependency parsers from existing tabular methods. Additionally, we provide a detailed discussion of the conditions under which the feature models commonly used in transition-based parsing can be integrated into our algorithms.

## 1 Introduction

Dynamic programming algorithms, also known as tabular or chart-based algorithms, are at the core of many applications in natural language processing. When applied to formalisms such as context-free grammar, they provide polynomial-time parsing algorithms and polynomial-space representations of the resulting parse forests, even in cases where the size of the search space is exponential in the length of the input string. In combination with appropriate semirings, these packed representations can be exploited to compute many values of interest for machine learning, such as best parses and feature expectations (Goodman, 1999; Li and Eisner, 2009).

In this paper, we follow the line of investigation started by Huang and Sagae (2010) and apply dynamic programming to (projective) transition-based dependency parsing (Nivre, 2008). The basic idea, originally developed in the context of push-down automata (Lang, 1974; Tomita, 1986; Billot and Lang, 1989), is that while the number of computations of a transition-based parser may be exponential in the length of the input string, several portions of these computations, when appropriately represented, can be shared. This can be effectively implemented through dynamic programming, resulting in a packed representation of the set of all computations.

The contributions of this paper can be summarized as follows. We provide (declarative specifications of) novel, polynomial-time algorithms for two widely-used transition-based parsing models: arc-standard (Nivre, 2004; Huang and Sagae, 2010) and arc-eager (Nivre, 2003; Zhang and Clark, 2008). Our algorithm for the arc-eager model is the first tabular algorithm for this model that runs in polynomial time. Both algorithms are derived using the same general technique; in fact, we show that this technique is applicable to all transition-parsing models whose transitions can be classified into "shift" and "reduce" transitions. We also show how to reverse the tabulation to derive a new transition system from an existing tabular algorithm for dependency parsing, originally developed by Gómez-Rodríguez et al. (2008). Finally, we discuss in detail the role of feature information in our algorithms, and in particular the conditions under which the feature models traditionally used in transition-based dependency parsing can be integrated into our framework.

While our general approach is the same as the one of Huang and Sagae (2010), we depart from their framework by not representing the computations of a parser as a graph-structured stack in the sense of Tomita (1986). We instead simulate computations as in Lang (1974), which results in simpler algorithm specifications, and also reveals deep similarities between transition-based systems for dependency parsing and existing tabular methods for lexicalized context-free grammars.



*Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 673–682,
Portland, Oregon, June 19-24, 2011. 

## 2 Transition-Based Dependency Parsing

We start by briefly introducing the framework of transition-based dependency parsing; for details, we refer to Nivre (2008).

### 2.1 Dependency Graphs

Let $w = w_0 \cdots w_{n-1}$ be a string over some fixed alphabet, where $n \geq 1$ and $w_0$ is the special token ROOT. A *dependency graph* for $w$ is a directed graph $G = (V_w, A)$, where $V_w = \{0, \ldots, n-1\}$ is the set of nodes, and $A \subseteq V_w \times V_w$ is the set of arcs. Each node in $V_w$ encodes the position of a token in $w$, and each arc in $A$ encodes a dependency relation between two tokens. To denote an arc $(i, j) \in A$, we write $i \rightarrow j$; here, the node $i$ is the head, and the node $j$ is the dependent. A sample dependency graph is given in the left part of Figure 2.

### 2.2 Transition Systems

A *transition system* is a structure $S = (C, T, I, C_t)$, where $C$ is a set of *configurations*, $T$ is a finite set of *transitions*, which are partial functions $t : C \rightharpoonup C$, $I$ is a total *initialization function* mapping each input string to a unique initial configuration, and $C_t \subseteq C$ is a set of *terminal configurations*.

The transition systems that we investigate in this paper differ from each other only with respect to their sets of transitions, and are identical in all other aspects. In each of them, a configuration is defined relative to a string $w$ as above, and is a triple $c = (\sigma, \beta, A)$, where $\sigma$ and $\beta$ are disjoint lists of nodes from $V_w$, called *stack* and *buffer*, respectively, and $A \subseteq V_w \times V_w$ is a set of arcs. We denote the stack, buffer and arc set associated with $c$ by $\sigma(c)$, $\beta(c)$, and $A(c)$, respectively. We follow a standard convention and write the stack with its topmost element to the right, and the buffer with its first element to the left; furthermore, we indicate concatenation in the stack and in the buffer by a vertical bar. The initialization function maps each string $w$ to the initial configuration $([], [0, \ldots, |w|-1], \emptyset)$. The set of terminal configurations contains all configurations of the form $([0], [], A)$, where $A$ is some set of arcs.

Given an input string $w$, a parser based on $S$ processes $w$ from left to right, starting in the initial configuration $I(w)$. At each point, it applies one of the transitions, until at the end it reaches a terminal configuration; the dependency graph defined by the arc set associated with that configuration is then returned as the analysis for $w$. Formally, a *computation* of $S$ on $w$ is a sequence $\gamma = c_0, \ldots, c_m$, $m \geq 0$, of configurations (defined relative to $w$) in which each configuration is obtained as the value of the preceding one under some transition. It is called *complete* whenever $c_0 = I(w)$, and $c_m \in C_t$. We note that a computation can be uniquely specified by its initial configuration $c_0$ and the sequence of its transitions, understood as a string over $T$. Complete computations, where $c_0$ is fixed, can be specified by their transition sequences alone.

$$
\begin{aligned}
(\sigma, i|\beta, A) &\vdash (\sigma|i, \beta, A) && \text{(sh)} \\
(\sigma|i|j, \beta, A) &\vdash (\sigma|j, \beta, A \cup \{j \rightarrow i\}) && \text{(la)} \\
(\sigma|i|j, \beta, A) &\vdash (\sigma|i, \beta, A \cup \{i \rightarrow j\}) && \text{(ra)}
\end{aligned}
$$

Figure 1: Transitions in the arc-standard model.

## 3 Arc-Standard Model

To introduce the core concepts of the paper, we first look at a particularly simple model for transition-based dependency parsing, known as the *arc-standard model*. This model has been used, in slightly different variants, by a number of parsers (Nivre, 2004; Attardi, 2006; Huang and Sagae, 2010).

### 3.1 Transition System

The arc-standard model uses three types of transitions: SHIFT (sh) removes the first node in the buffer and pushes it to the stack. LEFT-ARC (la) creates a new arc with the topmost node on the stack as the head and the second-topmost node as the dependent, and removes the second-topmost node from the stack. RIGHT-ARC (ra) is symmetric to LEFT-ARC in that it creates an arc with the second-topmost node as the head and the topmost node as the dependent, and removes the topmost node.

The three transitions can be formally specified as in Figure 1. The right half of Figure 2 shows a complete computation of the arc-standard transition system, specified by its transition sequence. The picture also shows the contents of the stack over the course of the computation; more specifically, column $i$ shows the stack $\sigma(c_i)$ associated with the configuration $c_i$.

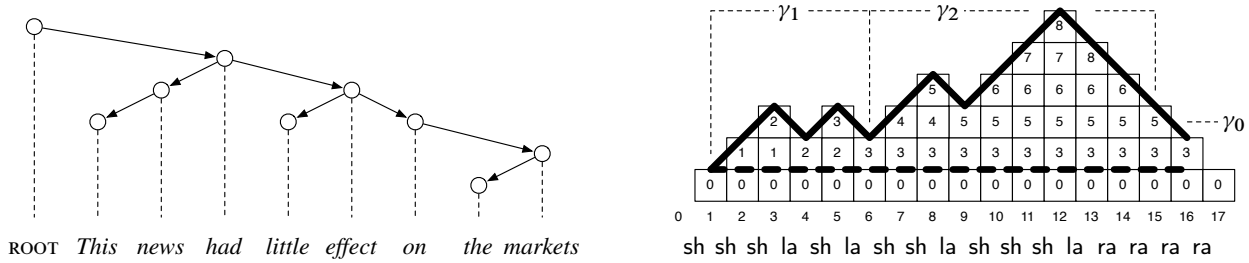


Figure 2: A dependency tree (left) and a computation generating this tree in the arc-standard system (right).

### 3.2 Push Computations

The key to the tabulation of transition-based dependency parsers is to find a way to decompose computations into smaller, shareable parts. For the arc-standard model, as well as for the other transition systems that we consider in this paper, we base our decomposition on the concept of *push computations*. By this, we mean computations

$$\gamma = c_0, \ldots, c_m, \quad m \geq 1,$$

on some input string $w$ with the following properties:

(P1) The initial stack $\sigma(c_0)$ is not modified during the computation, and is not even exposed after the first transition: For every $1 \leq i \leq m$, there exists a non-empty stack $\sigma_i$ such that $\sigma(c_i) = \sigma(c_0)|\sigma_i$.

(P2) The overall effect of the computation is to push a single node to the stack: The stack $\sigma(c_m)$ can be written as $\sigma(c_m) = \sigma(c_0)|h$, for some $h \in V_w$.

We can verify that the computation in Figure 2 is a push computation. We can also see that it contains shorter computations that are push computations; one example is the computation $\gamma_0 = c_1, \ldots, c_{16}$, whose overall effect is to push the node 3. In Figure 2, this computation is marked by the zig-zag path traced in bold. The dashed line delineates the stack $\sigma(c_1)$, which is not modified during $\gamma_0$.

Every computation that consists of a single sh transition is a push computation. Starting from these atoms, we can build larger push computations by means of two (partial) binary operations $f_{\mathsf{la}}$ and $f_{\mathsf{ra}}$, defined as follows. Let $\gamma_1 = c_{10}, \ldots, c_{1m_1}$ and $\gamma_2 = c_{20}, \ldots, c_{2m_2}$ be push computations on the same input string $w$ such that $c_{1m_1} = c_{20}$. Then

$$f_{\mathsf{ra}}(\gamma_1, \gamma_2) = c_{10}, \ldots, c_{1m_1}, c_{21}, \ldots, c_{2m_2}, c,$$

where $c$ is obtained from $c_{2m_2}$ by applying the ra transition. (The operation $f_{\mathsf{la}}$ is defined analogously.) We can verify that $f_{\mathsf{ra}}(\gamma_1, \gamma_2)$ is another push computation. For instance, with respect to Figure 2, $f_{\mathsf{ra}}(\gamma_1, \gamma_2) = \gamma_0$. Conversely, we say that the push computation $\gamma_0$ can be *decomposed* into the subcomputations $\gamma_1$ and $\gamma_2$, and the operation $f_{\mathsf{ra}}$.

### 3.3 Deduction System

Building on the compositional structure of push computations, we now construct a deduction system (in the sense of Shieber et al. (1995)) that tabulates the computations of the arc-standard model for a given input string $w = w_0 \cdots w_{n-1}$. For $0 \leq i \leq n$, we shall write $\beta_i$ to denote the buffer $[i, \ldots, n-1]$. Thus, $\beta_0$ denotes the full buffer, associated with the initial configuration $I(w)$, and $\beta_n$ denotes the empty buffer, associated with a terminal configuration $c \in C_t$.

*Item form.* The items of our deduction system take the form $[i, h, j]$, where $0 \leq i \leq h < j \leq n$. The intended interpretation of an item $[i, h, j]$ is: For every configuration $c_0$ with $\beta(c_0) = \beta_i$, there exists a push computation $\gamma = c_0, \ldots, c_m$ such that $\beta(c_m) = \beta_j$, and $\sigma(c_m) = \sigma(c_0)|h$.

*Goal.* The only goal item is $[0, 0, n]$, asserting that there exists a complete computation for $w$.

*Axioms.* For every stack $\sigma$, position $i < n$ and arc set $A$, by a single sh transition we obtain the push computation $(\sigma, \beta_i, A), (\sigma|i, \beta_{i+1}, A)$. Therefore we can take the set of all items of the form $[i, i, i+1]$ as the axioms of our system.

*Inference rules.* The inference rules parallel the composition operations $f_{\mathsf{la}}$ and $f_{\mathsf{ra}}$. Suppose that we have deduced the items $[i, h_1, k]$ and $[k, h_2, j]$, where $0 \leq i \leq h_1 < k \leq h_2 < j \leq n$. The item $[i, h_1, k]$ asserts that for every configuration $c_{10}$

$$\textbf{Item form: } [i,h,j]\,,\ 0 \le i \le h < j \le |w| \qquad \textbf{Goal: } [0,0,|w|] \qquad \textbf{Axioms: } [i,i,i+1]$$

$$\textbf{Inference rules: } \frac{[i,h_1,k] \quad [k,h_2,j]}{[i,h_2,j]}\ (\mathsf{la}; h_2 \to h_1) \qquad \frac{[i,h_1,k] \quad [k,h_2,j]}{[i,h_1,j]}\ (\mathsf{ra}; h_1 \to h_2)$$

Figure 3: Deduction system for the arc-standard model.

with $\beta(c_{10}) = \beta_i$, there exists a push computation $\gamma_1 = c_{10}, \ldots, c_{1m_1}$ such that $\beta(c_{1m_1}) = \beta_k$, and $\sigma(c_{1m_1}) = \sigma(c_{10})|h_1$. Using the item $[k, h_2, j]$, we deduce the existence of a second push computation $\gamma_2 = c_{20}, \ldots, c_{2m_2}$ such that $c_{20} = c_{1m_1}$, $\beta(c_{2m_2}) = \beta_j$, and $\sigma(c_{2m_2}) = \sigma(c_{10})|h_1|h_2$. By means of $f_{\mathsf{ra}}$, we can then compose $\gamma_1$ and $\gamma_2$ into a new push computation

$$f_{\mathsf{ra}}(\gamma_1, \gamma_2) \;=\; c_{10}, \ldots, c_{1m_1}, c_{21}, \ldots, c_{2m_2}, c\,.$$

Here, $\beta(c) = \beta_j$, and $\sigma(c) = \sigma(c_{10})|h_1$. Therefore, we may generate the item $[i, h_1, j]$. The inference rule for la can be derived analogously.

Figure 3 shows the complete deduction system.

### 3.4 Completeness and Non-Ambiguity

We have informally argued that our deduction system is sound. To show completeness, we prove the following lemma: For all $0 \le i \le h < j \le |w|$ and every push computation $\gamma = c_0, \ldots, c_m$ on $w$ with $\beta(c_0) = \beta_i$, $\beta(c_m) = \beta_j$ and $\sigma(c_m) = \sigma(c_0)|h$, the item $[i, h, j]$ is generated. The proof is by induction on $m$, and there are two cases:

$m = 1$. In this case, $\gamma$ consists of a single sh transition, $h = i$, $j = i + 1$, and we need to show that the item $[i, i, i + 1]$ is generated. This holds because this item is an axiom.

$m \ge 2$. In this case, $\gamma$ ends with either a la or a ra transition. Let $c$ be the rightmost configuration in $\gamma$ that is different from $c_m$ and whose stack size is one larger than the size of $\sigma(c_0)$. The computations

$$\gamma_1 = c_0, \ldots, c \quad \text{and} \quad \gamma_2 = c, \ldots, c_{m-1}$$

are both push computations with strictly fewer transitions than $\gamma$. Suppose that the last transition in $\gamma$ is ra. In this case, $\beta(c) = \beta_k$ for some $i < k < j$, $\sigma(c) = \sigma(c_0)|h$ with $h < k$, $\beta(c_{m-1}) = \beta_j$, and $\sigma(c_{m-1}) = \sigma(c_0)|h|h'$ for some $k \le h' < j$. By induction, we may assume that we have generated items $[i, h, k]$ and $[k, h', j]$. Applying the inference rule for ra, we deduce the item $[i, h, j]$. An analogous argument can be made for $f_{\mathsf{la}}$.

Apart from being sound and complete, our deduction system also has the property that it assigns at most one derivation to a given item. To see this, note that in the proof of the lemma, the choice of $c$ is uniquely determined: If we take any other configuration $c'$ that meets the selection criteria, then the computation $\gamma_2' = c', \ldots, c_{m-1}$ is not a push computation, as it contains $c$ as an intermediate configuration, and thereby violates property P1.

### 3.5 Discussion

Let us briefly take stock of what we have achieved so far. We have provided a deduction system capable of tabulating the set of all computations of an arc-standard parser on a given input string, and proved the correctness of this system relative to an interpretation based on push computations. Inspecting the system, we can see that its generic implementation takes space in $\mathcal{O}(|w|^3)$ and time in $\mathcal{O}(|w|^5)$.

Our deduction system is essentially the same as the one for the CKY algorithm for bilexicalized context-free grammar (Collins, 1996; Gómez-Rodríguez et al., 2008). This equivalence reveals a deep correspondence between the arc-standard model and bilexicalized context-free grammar, and, via results by Eisner and Satta (1999), to head automata. In particular, Eisner's and Satta's "hook trick" can be applied to our tabulation to reduce its runtime to $\mathcal{O}(|w|^4)$.

## 4 Adding Features

The main goal with the tabulation of transition-based dependency parsers is to obtain a representation based on which semiring values such as the highest-scoring computation for a given input (and with it, a dependency tree) can be calculated. Such computations involve the use of feature information. In this section, we discuss how our tabulation of the arc-standard system can be extended for this purpose.

$$\frac{[i, h_1, k; \langle x_2, x_1\rangle, \langle x_1, x_3\rangle] : v_1 \quad [k, h_2, j; \langle x_1, x_3\rangle, \langle x_3, x_4\rangle] : v_2}{[i, h_1, j; \langle x_2, x_1\rangle, \langle x_1, x_3\rangle] : v_1 + v_2 + \langle x_3, x_4\rangle \cdot \vec{\alpha}_{\mathsf{ra}}} \ (\mathsf{ra})$$

$$\frac{[i, h, j; \langle x_2, x_1\rangle, \langle x_1, x_3\rangle] : v}{[j, j, j+1; \langle x_1, x_3\rangle, \langle x_3, w_j\rangle] : \langle x_1, x_3\rangle \cdot \vec{\alpha}_{\mathsf{sh}}} \ (\mathsf{sh})$$

Figure 4: Extended inference rules under the feature model $\Phi = \langle \mathsf{s}_1.\mathsf{w}, \mathsf{s}_0.\mathsf{w}\rangle$. The annotations indicate how to calculate a candidate for an update of the Viterbi score of the conclusion using the Viterbi scores of the premises.

### 4.1 Scoring Computations

For the sake of concreteness, suppose that we want to score computations based on the following model, taken from Zhang and Clark (2008). The score of a computation $\gamma$ is broken down into a sum of scores $score(t, c_t)$ for combinations of a transition $t$ in the transition sequence associated with $\gamma$ and the configuration $c_t$ in which $t$ was taken:

$$score(\gamma) \ = \ \sum_{t \in \gamma} score(t, c_t) \qquad (1)$$

The score $score(t, c_t)$ is defined as the dot product of the feature representation of $c_t$ relative to a *feature model* $\Phi$ and a transition-specific weight vector $\vec{\alpha}_t$:

$$score(t, c_t) \ = \ \Phi(c_t) \cdot \vec{\alpha}_t$$

The feature model $\Phi$ is a vector $\langle \phi_1, \ldots, \phi_n\rangle$ of elementary *feature functions*, and the feature representation $\Phi(c)$ of a configuration $c$ is a vector $\vec{x} = \langle \phi_1(c), \ldots, \phi_n(c)\rangle$ of atomic values. Two examples of feature functions are the word form associated with the topmost and second-topmost node on the stack; adopting the notation of Huang and Sagae (2010), we will write these functions as $\mathsf{s}_0.\mathsf{w}$ and $\mathsf{s}_1.\mathsf{w}$, respectively. Feature functions like these have been used in several parsers (Nivre, 2006; Zhang and Clark, 2008; Huang et al., 2009).

### 4.2 Integration of Feature Models

To integrate feature models into our tabulation of the arc-standard system, we can use extended items of the form $[i, h, j; \vec{x}_L, \vec{x}_R]$ with the same intended interpretation as the old items $[i, h, j]$, except that the initial configuration of the asserted computations $\gamma = c_0, \ldots, c_m$ now is required to have the feature representation $\vec{x}_L$, and the final configuration is required to have the representation $\vec{x}_R$:

$$\Phi(c_0) = \vec{x}_L \quad \text{and} \quad \Phi(c_m) = \vec{x}_R$$

We shall refer to the vectors $\vec{x}_L$ and $\vec{x}_R$ as the *left-context vector* and the *right-context vector* of the computation $\gamma$, respectively.

We now need to change the deduction rules so that they become faithful to the extended interpretation. Intuitively speaking, we must ensure that the feature values can be computed along the inference rules. As a concrete example, consider the feature model $\Phi = \langle \mathsf{s}_1.\mathsf{w}, \mathsf{s}_0.\mathsf{w}\rangle$. In order to integrate this model into our tabulation, we change the rule for $\mathsf{ra}$ as in Figure 4, where $x_1, \ldots, x_4$ range over possible word forms. The shared variable occurrences in this rule capture the constraints that hold between the feature values of the subcomputations $\gamma_1$ and $\gamma_2$ asserted by the premises, and the computations $f_{\mathsf{ra}}(\gamma_1, \gamma_2)$ asserted by the conclusion. To illustrate this, suppose that $\gamma_1$ and $\gamma_2$ are as in Figure 2. Then the three occurrences of $x_3$ for instance encode that

$$[\mathsf{s}_0.\mathsf{w}](c_6) = [\mathsf{s}_1.\mathsf{w}](c_{15}) = [\mathsf{s}_0.\mathsf{w}](c_{16}) = w_3 \,.$$

We also need to extend the axioms, which correspond to computations consisting of a single $\mathsf{sh}$ transition. The most conservative way to do this is to use a generate-and-test technique: Extend the existing axioms by all valid choices of left-context and right-context vectors, that is, by all pairs $\vec{x}_L, \vec{x}_R$ such that there exists a configuration $c$ with $\Phi(c) = \vec{x}_L$ and $\Phi(\mathsf{sh}(c)) = \vec{x}_R$. The task of filtering out useless guesses can then be delegated to the deduction system.

A more efficient way is to only have one axiom, for the case where $c = I(w)$, and to add to the deduction system a new, unary inference rule for $\mathsf{sh}$ as in Figure 4. This rule only creates items whose left-context vector is the right-context vector of some other item, which prevents the generation of useless items. In the following, we take this second approach, which is also the approach of Huang and Sagae (2010).

$$\frac{[i, h, j; \langle x_2, x_1\rangle, \langle x_1, x_3\rangle] : (p, v)}{[j, j, j+1; \langle x_1, x_3\rangle, \langle x_3, w_j\rangle] : (p+\sigma, \sigma)}\ (\mathsf{sh})\,,\ \text{where } \sigma = \langle x_1, x_3\rangle \cdot \vec{\alpha}_{\mathsf{sh}}$$

$$\frac{[i, h_1, k; \langle x_2, x_1\rangle, \langle x_1, x_3\rangle] : (p_1, v_1) \quad [k, h_2, j; \langle x_1, x_3\rangle, \langle x_3, x_4\rangle] : (p_2, v_2)}{[i, h_1, j; \langle x_2, x_1\rangle, \langle x_1, x_3\rangle] : (p_1 + v_2 + \rho, v_1 + v_2 + \rho)}\ (\mathsf{ra})\,,\ \text{where } \rho = \langle x_3, x_4\rangle \cdot \vec{\alpha}_{\mathsf{ra}}$$

Figure 5: Extended inference rules under the feature model $\Phi = \langle \mathsf{s}_0.\mathsf{w}, \mathsf{s}_1.\mathsf{w}\rangle$. The annotations indicate how to calculate a candidate for an update of the prefix score and Viterbi score of the conclusion.

### 4.3 Computing Viterbi Scores

Once we have extended our deduction system with feature information, many values of interest can be computed. One simple example is the *Viterbi score* for an input $w$, defined as

$$\arg\max_{\gamma \in \Gamma(w)} \mathit{score}(\gamma)\,, \tag{2}$$

where $\Gamma(w)$ denotes the set of all complete computations for $w$. The score of a complex computation $f_t(\gamma_1, \gamma_2)$ is the sum of the scores of its subcomputations $\gamma_1, \gamma_2$, plus the transition-specific dot product. Since this dot product only depends on the feature representation of the final configuration of $\gamma_2$, the Viterbi score can be computed on top of the inference rules using standard techniques. The crucial calculation is indicated in Figure 4.

### 4.4 Computing Prefix Scores

Another interesting value is the *prefix score* of an item, which, apart from the Viterbi score, also includes the cost of the best search path leading to the item. Huang and Sagae (2010) use this quantity to order the items in a beam search on top of their dynamic programming method. In our framework, prefix scores can be computed as indicated in Figure 5. Alternatively, we can also use the more involved calculation employed by Huang and Sagae (2010), which allows them to get rid of the left-context vector from their items.[1]

### 4.5 Compatibility

So far we have restricted our attention to a concrete and extremely simplistic feature model. The feature models that are used in practical systems are considerably more complex, and not all of them are compatible with our framework in the sense that they can be integrated into our deduction system in the way described in Section 4.2.

For a simple example of a feature model that is incompatible with our tabulation, consider the model $\Phi' = \langle \mathsf{s}_0.\mathsf{rc}.\mathsf{w}\rangle$, whose single feature function extracts the word form of the right child (rc) of the topmost node on the stack. Even if we know the values of this feature for two computations $\gamma_1, \gamma_2$, we have no way to compute its value for the composed computation $f_{\mathsf{ra}}(\gamma_1, \gamma_2)$: This value coincides with the word form of the topmost node on the stack associated with $\gamma_2$, but in order to have access to it in the context of the ra rule, our feature model would need to also include the feature function $\mathsf{s}_0.\mathsf{w}$.

The example just given raises the question whether there is a general criterion based on which we can decide if a given feature model is compatible with our tabulation. An attempt to provide such a criterion has been made by Huang and Sagae (2010), who define a constraint on feature models called "monotonicity" and claim that this constraint guarantees that feature values can be computed using their dynamic programming approach. Unfortunately, this claim is wrong. In particular, the feature model $\Phi'$ given above is "monotonic", but cannot be tabulated, neither in our nor in their framework. In general, it seems clear that the question of compatibility is a question about the *relation* between the tabulation and the feature model, and not about the feature model alone. To find practically useful characterizations of compatibility is an interesting avenue for future research.

## 5 Arc-Eager Model

Up to now, we have only discussed the arc-standard model. In this section, we show that the framework of push computations also provides a tabulation of another widely-used model for dependency parsing, the *arc-eager model* (Nivre, 2003).

[1] The essential idea in the calculation by Huang and Sagae (2010) is to delegate (in the computation of the Viterbi score) the scoring of sh transitions to the inference rules for la/ra.

$$(\sigma, i|\beta, A) \vdash (\sigma|i, \beta, A) \quad (\mathsf{sh})$$
$$(\sigma|i, j|\beta, A) \vdash (\sigma, j|\beta, A \cup \{j \rightarrow i\}) \quad (\mathsf{la_e})$$
only if $i$ does not have an incoming arc
$$(\sigma|i, j|\beta, A) \vdash (\sigma|i|j, \beta, A \cup \{i \rightarrow j\}) \quad (\mathsf{ra_e})$$
$$(\sigma|i, \beta, A) \vdash (\sigma, \beta, A) \quad (\mathsf{re})$$
only if $i$ has an incoming arc

Figure 6: Transitions in the arc-eager model.

### 5.1 Transition System

The arc-eager model has three types of transitions, shown in Figure 6: SHIFT (sh) works just like in arc-standard, moving the first node in the buffer to the stack. LEFT-ARC ($\mathsf{la_e}$) creates a new arc with the first node in the buffer as the head and the topmost node on the stack as the dependent, and pops the stack. It can only be applied if the topmost node on the stack has not already been assigned a head, so as to preserve the single-head constraint. RIGHT-ARC ($\mathsf{ra_e}$) creates an arc in the opposite direction as LEFT-ARC, and moves the first node in the buffer to the stack. Finally, REDUCE (re) simply pops the stack; it can only be applied if the topmost node on the stack has already been assigned a head.

Note that, unlike in the case of arc-standard, the parsing process in the arc-eager model is not bottom-up: the right dependents of a node are attached before they have been assigned their own right dependents.

### 5.2 Shift-Reduce Parsing

If we look at the specification of the transitions of the arc-standard and the arc-eager model and restrict our attention to the effect that they have on the stack and the buffer, then we can see that all seven transitions fall into one of three types:

$$(\sigma, i|\beta) \vdash (\sigma|i, \beta) \quad \mathsf{sh}, \mathsf{ra_e} \quad \text{(T1)}$$
$$(\sigma|i|j, \beta) \vdash (\sigma|j, \beta) \quad \mathsf{la} \quad \text{(T2)}$$
$$(\sigma|i, \beta) \vdash (\sigma, \beta) \quad \mathsf{ra}, \mathsf{la_e}, \mathsf{re} \quad \text{(T3)}$$

We refer to transitions of type T1 as *shift* and to transitions of type T2 and T3 as *reduce transitions*.

The crucial observation now is that the concept of push computations and the approach to their tabulation that we have taken for the arc-standard system can easily be generalized to other transition systems whose transitions are of the type *shift* or *reduce*. In particular, the proof of the correctness of our deduction system that we gave in Section 3 still goes through if instead of sh we write "shift" and instead of la and ra we write "reduce".

### 5.3 Deduction System

Generalizing our construction for the arc-standard model along these lines, we obtain a tabulation of the arc-eager model. Just like in the case of arc-standard, each single shift transition in that model (be it sh or $\mathsf{ra_e}$) constitutes a push computation, while the reduce transitions induce operations $f_{\mathsf{la_e}}$ and $f_{\mathsf{re}}$. The only difference is that the preconditions of $\mathsf{la_e}$ and re must be met. Therefore, $f_{\mathsf{la_e}}(\gamma_1, \gamma_2)$ is only defined if the topmost node on the stack in the final configuration of $\gamma_2$ has not yet been assigned a head, and $f_{\mathsf{re}}(\gamma_1, \gamma_2)$ is only defined in the opposite case.

*Item form.* In our deduction system for the arc-eager model we use items of the form $[i, h^b, j]$, where $0 \leq i \leq h < j \leq |w|$, and $b \in \{0, 1\}$. An item $[i, h^b, j]$ has the same meaning as the corresponding item in our deduction system for arc-standard, but also keeps record of whether the node $h$ has been assigned a head ($b = 1$) or not ($b = 0$).

*Goal.* The only goal item is $[0, 0^0, |w|]$. (The item $[0, 0^1, |w|]$ asserts that the node 0 has a head, which never happens in a complete computation.)

*Axioms.* Reasoning as in arc-standard, the axioms of the deduction system for the arc-eager model are the items of the form $[i, i^0, i + 1]$ and $[j, j^1, j + 1]$, where $j > 0$: the former correspond to the push computations obtained from a single sh, the latter to those obtained from a single $\mathsf{ra_e}$, which apart from shifting a node also assigns it a head.

*Inference rules.* Also analogously to arc-standard, if we know that there exists a push computation $\gamma_1$ of the form asserted by the item $[i, h^b, k]$, and a push computation $\gamma_2$ of the form asserted by $[k, g^0, j]$, where $j < |w|$, then we can build the push computation $f_{\mathsf{la_e}}(\gamma_1, \gamma_2)$ of the form asserted by the item $[i, h^b, j]$. Similarly, if $\gamma_2$ is of the form asserted by $[k, g^1, j]$, then we can build $f_{\mathsf{re}}(\gamma_1, \gamma_2)$, which again is of the form by asserted $[i, h^b, j]$. Thus:

$$\frac{[i, i^b, k] \quad [k, k^0, j]}{[i, i^b, j]} \ (\mathsf{la_e}), \qquad \frac{[i, i^b, k] \quad [k, k^1, j]}{[i, i^b, j]} \ (\mathsf{re}).$$

**Item form:** $[i^b, j]\,, 0 \leq i < j \leq |w|\,, b \in \{0,1\}$ **Goal:** $[0^0, |w|]$ **Axioms:** $[0^0, 1]$

$$\frac{[i^b, j]}{[j^0, j+1]}\ (\mathsf{sh}) \qquad \frac{[i^b, k]\ \ [k^0, j]}{[i^b, j]}\ (\mathsf{la_e}; j \rightarrow k)\,, j < |w| \qquad \frac{[i^b, j]}{[j^1, j+1]}\ (\mathsf{ra_e}; i \rightarrow j) \qquad \frac{[i^b, k]\ \ [k^1, j]}{[i^b, j]}\ (\mathsf{re})$$

Figure 7: Deduction system for the arc-eager model.

As mentioned above, the correctness and non-ambiguity of the system can be proved as in Section 3. Features can be added in the same way as discussed in Section 4.

### 5.4 Computational Complexity

Looking at the inference rules, it is clear that an implementation of the deduction system for arc-eager takes space in $\mathcal{O}(|w|^3)$ and time in $\mathcal{O}(|w|^5)$, just like in the case of arc-standard. However, a closer inspection reveals that we can give even tighter bounds.

In all derivable items $[i, h^b, j]$, it holds that $i = h$. This can easily be shown by induction: The property holds for the axioms, and the first two indexes of a consequent of a deduction rule coincide with the first two indexes of the left antecedent. Thus, if we use the notation $[i^b, k]$ as a shorthand for $[i, i^b, k]$, then we can rewrite the inference rules for the arc-eager system as in Figure 7, where, additionally, we have added unary rules for sh and ra and restricted the set of axioms along the lines set out in Section 4.2. With this formulation, it is apparent that the space complexity of the generic implementation of the deduction system is in fact even in $\mathcal{O}(|w|^2)$, and its time complexity is in $\mathcal{O}(|w|^3)$.

## 6 Hybrid Model

We now reverse the approach that we have taken in the previous sections: Instead of tabulating a transition system in order to get a dynamic-programming parser that simulates its computations, we start with a tabular parser and derive a transition system from it. In the new model, dependency trees are built bottom-up as in the arc-standard model, but the set of all computations in the system can be tabulated in space $\mathcal{O}(|w|^2)$ and time $\mathcal{O}(|w|^3)$, as in arc-eager.

### 6.1 Deduction System

Gómez-Rodríguez et al. (2008) present a deductive version of the dependency parser of Yamada and Matsumoto (2003); their deduction system is given in Figure 8. The generic implementation of the deduction system takes space $\mathcal{O}(|w|^2)$ and time $\mathcal{O}(|w|^3)$.

In the original interpretation of the deduction system, an item $[i, j]$ asserts the existence of a pair of (projective) dependency trees: the first tree rooted at token $w_i$, having all nodes in the substring $w_i \cdots w_{k-1}$ as descendants, where $i < k \leq j$; and the second tree rooted at token $w_j$, having all nodes in the substring $w_k \cdots w_j$ as descendants. (Note that we use fencepost indexes, while Gómez-Rodríguez et al. (2008) indexes positions.)

### 6.2 Transition System

In the context of our tabulation framework, we adopt a new interpretation of items: An item $[i, j]$ has the same meaning as an item $[i, i, j]$ in the tabulation of the arc-standard model; for every configuration $c$ with $\beta(c) = \beta_i$, it asserts the existence of a push computation that starts with $c$ and ends with a configuration $c'$ for which $\beta(c') = \beta_j$ and $\sigma(c') = \sigma(c)|i$.

If we interpret the inference rules of the system in terms of composition operations on push computations as usual, and also take the intended direction of the dependency arcs into account, then this induces a transition system with three transitions:

$$\begin{aligned}
(\sigma, i\,|\beta, A) &\vdash (\sigma|i, \beta, A) && (\mathsf{sh})\\
(\sigma|i, j\,|\beta, A) &\vdash (\sigma, j\,|\beta, A \cup \{j \rightarrow i\}) && (\mathsf{la_h})\\
(\sigma|i\,|j, \beta, A) &\vdash (\sigma|i, \beta, A \cup \{i \rightarrow j\}) && (\mathsf{ra})
\end{aligned}$$

We call this transition system the *hybrid model*, as sh and ra are just like in arc-standard, while $\mathsf{la_h}$ is like the LEFT-ARC transition in the arc-eager model ($\mathsf{la_e}$), except that it does not have the precondition. Like the arc-standard but unlike the arc-eager model, the hybrid model builds dependencies bottom-up.

## 7 Conclusion

In this paper, we have provided a general technique for the tabulation of transition-based dependency parsers, and applied it to obtain dynamic programming algorithms for two widely-used parsing models,

**Item form:** $[i, j]\,, 0 \leq i < j \leq |w|$ **Goal:** $[0, |w|]$ **Axioms:** $[0, 1]$

**Inference rules:** $\dfrac{[i, j]}{[j, j+1]}$ (sh) $\quad \dfrac{[i, k] \quad [k, j]}{[i, j]}$ $(\mathsf{la_h}; j \rightarrow k)\,, j < |w|$ $\quad \dfrac{[i, k] \quad [k, j]}{[i, j]}$ $(\mathsf{ra}; i \rightarrow k)$

Figure 8: Deduction system for the hybrid model.

arc-standard and (for the first time) arc-eager. The basic idea behind our technique is the same as the one implemented by Huang and Sagae (2010) for the special case of the arc-standard model, but instead of their graph-structured stack representation we use a tabulation akin to Lang's approach to the simulation of pushdown automata (Lang, 1974). This considerably simplifies both the presentation and the implementation of parsing algorithms. It has also enabled us to give simple proofs of correctness and establish relations between transition-based parsers and existing parsers based on dynamic programming.

While this paper has focused on the theoretical aspects and the analysis of dynamic programming versions of transition-based parsers, an obvious avenue for future work is the evaluation of the empirical performance and efficiency of these algorithms in connection with specific feature models. The feature models used in transition-based dependency parsing are typically very expressive, and exhaustive search with them quickly becomes impractical even for our cubic-time algorithms of the arc-eager and hybrid model. However, Huang and Sagae (2010) have provided evidence that the use of dynamic programming on top of a transition-based dependency parser can improve accuracy even without exhaustive search. The tradeoff between expressivity of the feature models on the one hand and the efficiency of the search on the other is a topic that we find worth investigating. Another interesting observation is that dynamic programming makes it possible to use predictive features, which cannot easily be integrated into a non-tabular transition-based parser. This could lead to the development of parsing models that cross the border between transition-based and tabular parsing.

## Acknowledgments

All authors contributed equally to the work presented in this paper. M. K. wrote most of the manuscript. C. G.-R. has been partially supported by Ministerio de Educación y Ciencia and FEDER (HUM2007-66607-C04) and Xunta de Galicia (PGIDIT07SIN005206PR, Rede Galega de Procesamento da Linguaxe e Recuperación de Información, Rede Galega de Lingüística de Corpus, Bolsas Estadías INCITE/FSE cofinanced).

## References

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170, New York, USA.

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 143–151, Vancouver, Canada.

Michael Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 184–191, Santa Cruz, CA, USA.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and Head Automaton Grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 457–464, College Park, MD, USA.

Carlos Gómez-Rodríguez, John Carroll, and David J. Weir. 2008. A deductive approach to dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL): Human Language Technologies*, pages 968–976, Columbus, OH, USA.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1077–1086, Uppsala, Sweden.

Liang Huang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1222–1231, Singapore.

Bernard Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In Jacques Loecx,

editor, *Automata, Languages and Programming, 2nd Colloquium, University of Saarbrücken, July 29–August 2, 1974*, number 14 in Lecture Notes in Computer Science, pages 255–269. Springer.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 40–51, Singapore.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT)*, pages 149–160, Nancy, France.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain.

Joakim Nivre. 2006. *Inductive Dependency Parsing*, volume 34 of *Text, Speech and Language Technology*. Springer.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Stuart M. Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.

Masaru Tomita. 1986. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Springer.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT)*, pages 195–206, Nancy, France.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562—571, Honolulu, HI, USA.