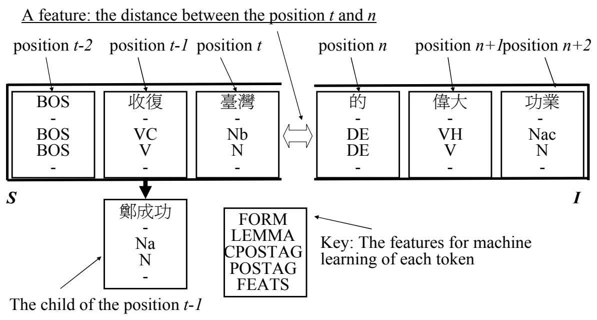

Fig. 2. The features for dependency analysis

the label. The architecture of the parser consists of four major procedures and as in **Fig.1**:

(i) Decide the neighboring dependency attachment between all adjacent words in the input sentence by SVM-based tagger (as a preprocessing)

(ii) Extract the surrounding features for the focused pair of nodes.

(iii) Estimate the dependency attachment operation of the focused pair of nodes by SVMs.

(iv) If there is a left or right attachment, estimate the label of dependency relation by MaxEnt.

We will explain the main procedures (steps (ii)-(iv)) in sections 2.1 and 2.2, and the preprocessing in section 2.3.

2.1 Word dependency analysis

In the algorithm, the state of the parser is represented by a triple $\langle S, I, A \rangle$. *S* and *I* are stacks, *S* keeps the words being in consideration, and *I* keeps the words to be processed. *A* is a list of dependency attachments decided in the algorithm. Given an input word sequence *W*, the parser is initialized by the triple $\langle nil, W, \phi \rangle$. The parser estimates the dependency attachment between two words (the top elements of stacks *S* and *I*). The algorithm iterates until the list *I* becomes empty. There are four possible operations (**Right**, **Left**, **Shift** and **Reduce**) for the configuration at hand.

Right or **Left:** If there is a dependency relation that the word *t* or *n* attaches to word *n* or *t*, add the new dependency relation $(t \rightarrow n)$ or $(n \rightarrow t)$ into *A*, remove *t* or *n* from *S* or *I*.

If there is no dependency relation between *n* and *t,* check the following conditions.

Reduce: If there is no word n' ($n' \in I$) which may depend on *t*, and *t* has a parent on its left side, the parser removes *t* from the stack *S*.

Shift: If there is no dependency between *n* and *t,* and the triple does not satisfy the conditions for **Reduce**, then push *n* onto the stack *S*.

In this work, we adopt SVMs for estimating the word dependency attachments. SVMs are binary classifiers based on the maximal margin strategy. We use the polynomial kernel: $K(\mathbf{x},\mathbf{z}) = (1+\mathbf{x}\cdot\mathbf{z})^d$ with d =2. The performance of SVMs is better than that of the maximum entropy method in our preceding work for Chinese dependency analysis (Cheng, 2005b). This is because that SVMs can combine features automatically (using the polynomial kernel), whereas the maximum entropy method cannot. To extend binary classifiers to multi-class classifiers, we use the pair-wise method, in which we make ${}_nC_2$ [1] binary classifiers between all pairs of the classes (Kreβel, 1998). We use Libsvm (Lin et al., 2001) in our experiments.

In our method, the parser considers the dependency attachment of two nodes (*n,t*). The features of a node are the word itself, the POS-tag and the information of its child node(s). The context features are 2 preceding nodes of node *t* (and *t* itself), 2 succeeding nodes of node *n* (and *n* itself), and their child nodes. The distance between nodes *n* and *t* is also used as a feature. The features are shown in **Fig.2**.

2.2 Label tagging

We adopt MaxEnt to estimate the label of dependency relations. We have tried to use linear-chain conditional random fields (CRFs) for estimating the labels after the dependency relation analysis. This means that the parser first analyzes the word dependency (head-modifier relation) of the input sentence, then the CRFs model analyzes the most suitable label set with the basic information of input sentence (FORM, LEMMA, POSTAG……etc) and the head information (FORM and POSTAG) of each word. However, as the number of possible labels in some languages is large, training a CRF model with these corpora (we use CRF++ (Kudo, 2005)) cost huge memory and time.

Instead, we combine the maximum entropy method in the word dependency analysis to tag the label of dependency relation. As shown in **Fig. 1**, the parser first gets the contextual features to estimate the word dependency. If the parsing operation

[1] To estimate the current operation (Left, Right, Shift and Reduce) by SVMs, we need to build 6 classifiers(Left-Right, Left-Shift, Left-Reduce, Right-Shift, Right-Reduce and Shift-Reduce).