two top items in S (s_1 and s_2) are popped, and a new item is pushed onto S. This new item is a tree formed by making the root s_1 of a dependent of the root of s_2, or the root of s_2 a dependent of the root of s_1. Depending on which of these two cases occur, we call the action *reduce-left* or *reduce-right*, according to whether the head of the new tree is to the left or to the right its new dependent. In addition to deciding the direction of a reduce action, the label of the newly formed dependency arc must also be decided.

Parsing terminates successfully when Q is empty (all words in the input have been processed) and S contains only a single tree (the final dependency tree for the input sentence). If Q is empty, S contains two or more items, and no further reduce actions can be taken, parsing terminates and the input is rejected. In such cases, the remaining items in S contain partial analyses for contiguous segments of the input.

2.2 A Probabilistic LR Model for Dependency Parsing

In the traditional LR algorithm, parser states are placed onto the stack, and an LR table is consulted to determine the next parser action. In our case, the parser state is encoded as a set of features derived from the contents of the stack S and queue Q, and the next parser action is determined according to that set of features. In the deterministic case described above, the procedure used for determining parser actions (a classifier, in our case) returns a single action. If, instead, this procedure returns a list of several possible actions with corresponding probabilities, we can then parse with a model similar to the probabilistic LR models described by Briscoe and Carroll (1993), where the probability of a parse tree is the product of the probabilities of each of the actions taken in its derivation.

To find the most probable parse tree according to the probabilistic LR model, we use a best-first strategy. This involves an extension of the deterministic shift-reduce into a best-first shift-reduce algorithm. To describe this extension, we first introduce a new data structure T_i that represents a parser state, which includes a stack S_i, a queue Q_i, and a probability P_i. The deterministic algorithm is a special case of the probabilistic algorithm where we have a single parser state T_0 that contains S_0 and Q_0, and the probability of the parser state is 1. The best-first algorithm, on the other hand, keeps a heap H containing multiple parser states $T_0 ... T_m$. These states are ordered in the heap according to their probabilities, which are determined by multiplying the probabilities of each of the parser actions that resulted in that parser state. The heap H is initialized to contain a single parser state T_0, which contains a stack S_0, a queue Q_0 and probability $P_0 = 1.0$. S_0 and Q_0 are initialized in the same way as S and Q in the deterministic algorithm. The best-first algorithm then loops while H is non-empty. At each iteration, first a state $T_{current}$ is popped from the top of H. If $T_{current}$ corresponds to a final state ($Q_{current}$ is empty and $S_{current}$ contains a single item), we return the single item in $S_{current}$ as the dependency structure corresponding to the input sentence. Otherwise, we get a list of parser actions $act_0 ... act_n$ (with associated probabilities $Pact_0 ... Pact_n$) corresponding to state $T_{current}$. For each of these parser actions act_j, we create a new parser state T_{new} by applying act_j to $T_{current}$, and set the probability T_{new} to be $P_{new} = P_{currnet} * Pact_j$. Then, T_{new} is inserted into the heap H. Once new states have been inserted onto H for each of the n parser actions, we move on to the next iteration of the algorithm.

3 Multilingual Parsing Experiments

For each of the ten languages for which training data was provided in the multilingual track of the CoNLL 2007 shared task, we trained three LR models as follows. The first LR model for each language uses maximum entropy classification (Berger et al., 1996) to determine possible parser actions and their probabilities[4]. To control overfitting in the MaxEnt models, we used box-type inequality constraints (Kazama and Tsujii, 2003). The second LR model for each language also uses MaxEnt classification, but parsing is performed *backwards*, which is accomplished simply by reversing the input string before parsing starts. Sagae and Lavie (2006a) and Zeman and Žabokrtský (2005) have observed that reversing the direction of stepwise parsers can be beneficial in parser combinations. The third model uses support vector machines[5] (Vapnik, 1995) using the polynomial

[4] Implementation by Yoshimasa Tsuruoka, available at http://www-tsujii.is.s.u-tokyo.ac.jp/~tsuruoka/maxent/
[5] Implementation by Taku Kudo, available at http://chasen.org/~taku/software/TinySVM/ and all vs. all was used for multi-class classification.