Pattern-Based Disambiguation for Natural Language Processing

Eric Brill
Microsoft Research
One Microsoft Way
Redmond, Wa. 98052
brill@microsoft.com

Abstract

A wide range of natural language problems can be viewed as disambiguating between a small set of alternatives based upon the string context surrounding the ambiguity site. In this paper we demonstrate that classification accuracy can be improved by invoking a more descriptive feature set than what is typically used. We present a technique that disambiguates by learning regular expressions describing the string contexts in which the ambiguity sites appear.

Introduction

Many natural language tasks are essentially n-way classification problems, where classification decisions are made from a small set of choices, based upon the linguistic context in which the ambiguity site occurs. Examples of such tasks include: confusable word set disambiguation; word sense disambiguation; determining such lexical features as pronoun case and determiner number for machine translation; part of speech tagging; named entity labeling; spelling correction; and some formulations of skeletal parsing. Very similar feature sets have been used across machine learning algorithms and across classification problems. For example, in confusable word set disambiguation, systems typically use as features the occurrence of a particular word within a window of +/- n words of the target, and collocations based on the words and part of speech tags of up to two words to the left and two words to the right of the target.

Below we present a machine learning algorithm that learns from a much richer feature set than that typically used for classification in natural language. Our algorithm learns rule sequences for n-way classification, where the condition of a rule can be a restricted regular expression on the string context in which the ambiguity site appears. We demonstrate that using this more powerful feature space leads to an improvement in disambiguation performance on confusable words.

1 Previous Work: Richer Features

Most previous work applying machine learning to linguistic disambiguation has used as features very local collocational information as well as the presence of a word within a fixed window of an ambiguity site. Indeed, one of the great insights in both speech recognition and natural language processing is the realization that fixed local cues provide a great deal of useful information.

While the n-gram reins supreme in language modeling, there has been some interesting work done building language models based on linguistically richer features. Bahl, Brown et al. (1989) describe a language model that builds a decision tree that is allowed to ask questions about the history up to twenty words back. Saul and Pereira (1997) describe a language model that can in essence skip over uninformative words in the history. Della Pietra et al. (1994) discuss an approach to language modeling based on link grammars, where the model can look beyond the two previous words to condition on linguistically relevant words in the history. The language model described by Chelba and Jelinek (1998) similarly conditions on linguistically relevant words by assigning partial phrase structure to the history and percolating headwords.

Samuellson, Tapanainen et al. (1996) describe a method for learning a particular

useful type of pattern, which they call a *barrier*. Given two symbols X and Y, and a set of symbols S, they learn conditions of the form: take an action if there is an X preceded by a Y, with no intervening symbols from S. In their paper they demonstrate how such patterns can be useful for part of speech tagging. Even-Zohar and Roth (2000) show that by including linguistic features based on relations such as subject and object, they can better disambiguate between verb pairs.

2 Definitions

Below we provide the standard definition for regular expressions, and then define a less expressive language formalism, which we will refer to as *reduced regular expressions*. The learning method we describe herein can learn rules conditioned on any reduced regular expression.

Regular Expression (RE):[1] Given a finite alphabet Σ, the set of regular expressions over that alphabet is defined as (Hopcroft and Ullman 1979):
(1) $\forall a \in \Sigma$, a is a regular expression and denotes the set {a}
(2) if r and s are regular expressions denoting the languages R and S, respectively, then (r+s), (rs), and (r*) are regular expressions that denote the sets $R \cup S$, RS and R* respectively.

Reduced Regular Expression (RRE): Given a finite alphabet Σ, the set of reduced regular expressions over that alphabet is defined as:

(1) $\forall a \in \Sigma$:
a is an RRE and denotes the set {a}
a+ is an RRE and denotes the positive closure of the set {a}
a* is an RRE and denotes the Kleene closure of the set {a}
~a is an RRE and denotes the set Σ - a
~a+ is an RRE and denotes the positive closure of the set Σ - a
~a* is an RRE and denotes the Kleene closure of the set Σ - a
(2) . is an RRE denoting the set Σ
(3) .+ is an RRE denoting the positive closure of the set Σ
(4) .* is an RRE denoting the Kleene closure of the set Σ
(5) if r and s are RREs denoting the languages R and S, respectively, then rs is an RRE denoting the set RS.

Some examples of strings that are regular expressions but not reduced regular expressions include: (ab)*, a(b|c)d, (a (bc)+)*

Next, we need some definitions to allow us to make reference to particular positions in a collection of strings.

Corpus: A corpus is an ordered set of strings. We will notate the j^{th} string of a corpus C as C[j]. |C| is the number of strings in the corpus. |C[j]| is the number of symbols in the j^{th} string of the corpus.

Corpus Position: A corpus position for a corpus C is a tuple (j,k), meaning the k^{th} symbol in the j^{th} string in the corpus, with the restrictions: $1 \leq j \leq |C|$ and $0 \leq k \leq |C[j]|$.

A **Corpus Position Set** is a set of corpus positions.

Next, we define an RRE-Tree, the data structure we will use in learning RREs.

RRE-Tree: An RRE-Tree over Σ is a tree (V,E), where V is a set of tuples <v,S>, v being a unique vertex identifier and S being a Corpus Position Set, and E is a set of labeled directed edges $<v_i,v_j,label>$, where v_i and v_j are vertex identifiers, label $\in$ LABEL_SET and LABEL_SET = {dot, dot+, dot*} U {a,a+,a*,~a,~a+,~a* | $\forall a \in \Sigma$ }.[2]

3 Rule Sequence Learning

Our implemented learner is based upon the transformation-based learning paradigm (Brill 1995). In this section we briefly review transformation-based learning.

[1] In all of our formulations, we ignore expressions denoting the empty set Ø and the set {ε}.

[2] We use "dot" for "."

In string classification, the goal is to assign the proper label to a string, from a prespecified set of labels L. A transformation-based system consists of:

(1) A start-state annotator, which assigns an initial label to a string.
(2) A sequence of rules of the form: *Change the label of a string from m to n if C(string)*, where C is a predicate over strings and m,n ∈ L.

A string is labelled by first applying the start-state annotator to it, and then applying each rule, in order.

To learn a transformation sequence, the system begins with a properly labelled training set. It then removes the labels and applies the start-state annotator to each string. Then the learner iteratively does the following:

(1) Find the best rule to apply to the training set.
(2) Append that rule to the end of the learned transformation sequence.
(3) Apply that rule to the training set.

until the stopping criterion is met.

4 Learning RRE Rules

Below we will demonstrate how to learn transformation sequences where the predicate C(string) is of the form "Does RRE R apply to the string?" We will show this for the binary classification case (where |L| = 2).

In each learning iteration, we will construct an RRE-Tree in a particular way, find the best node in that RRE-Tree, and then return the edge labels on the path from root to best node as the learned RRE. The learner will learn a sequence of rules of the form:

Change the label of a string from l_i to l_j if the string matches reduced regular expression R.

Before proceeding, we need to specify two things: the start-state annotator and the goodness measure for determining what rule is best. The system will use a start-state annotator that initially labels all strings with the most frequent label in the training set, and the goodness measure will simply be the number of good label changes minus the number of bad label changes when a rule is applied to the training set.

Take the following training set:

String #	String	True Label	Init. Guess
1	a b c	0	1
2	a b b	1	1
3	b a a	1	1

Since 1 is the most frequent label in the training set, the start-state annotator would initially assign all three training set strings the label 1, meaning string 1 would be incorrectly labelled and strings 2 and 3 would be correct. Now we want to learn a rule whose application will best improve our labelling of the training set.

4.1 RRE-Tree Construction

We will first present an algorithm for constructing an RRE-Tree for a training corpus, and then trace through the application of this algorithm to our example training corpus above. To simplify the presentation, we will limit ourselves to learning rules for a weaker language type, which we call Very Reduced Regular Expressions (VRREs). The extension to RRE learning is straightforward.

Very Reduced Regular Expression (VRRE): Given a finite alphabet Σ, the set of very reduced regular expressions over that alphabet is defined as:

(1) $\forall a \in \Sigma$: a is a VRRE and denotes the set {a}
(2) . is a VRRE denoting the set Σ
(3) .* is a VRRE denoting the Kleene closure of the set Σ
(4) if r and s are VRREs denoting the languages R and S, respectively, then rs is a VRRE denoting the set RS.

Say we have a training corpus C. For every string C[j]∈ C, Truth[C[j]] ∈ {0,1} is the true label of C[j] and Guess[C[j]] is the current guess of the label of C[j]. The algorithm for one iteration of rule learning follows.

Main() {

(1) Create root node with corpus position set S = {(j,0) | j = 1 .. |C|). Push this node onto processing stack (STACK).
(2) While (STACK not empty) {
STATE = pop(STACK);
Push(dotexpand(STATE),STACK);
Push(dotstarexpand(STATE),STACK);
$\forall a \in \Sigma$
Push(atomexpand(a,STATE),STACK)
}
(3) Find best state S in the RRE-tree. Let R be the RRE obtained by following the edges from the root to S, outputting each edge label as the edge is traversed. Return either the rule "0→1 if R" or "1→0 if R" depending on which is appropriate for state S.
}

dotexpand(STATE) {
create new state STATE'
let P be the corpus position set of STATE
P' = {(j,k) | (j,k-1) ∈ P and k-1 ≠ |Corpus[j]|}
If (P' not empty) {
Make P' the corpus position set of STATE'
Add (STATE,STATE',DOT) to tree edges
return STATE'
}
Else return NULL
}

dotstarexpand(STATE) {
create new state STATE'
let P be the corpus position set of STATE
P' = {(j,k) | (j,m) ∈ P, m ≤ k, and k ≤ |Corpus[j]|}
If (P' ≠ P) {
Make P' the corpus position set of STATE'
Add (STATE,STATE',DOT*) to tree edges
return STATE'
}
Else return NULL
}

atomexpand(a,STATE) {
create new state STATE'
let P be the corpus position set of STATE
P' = {(j,k) | (j,k-1) ∈ P, k-1 ≠ |Corpus[j]|, and the k-1st symbol in Corpus[j] is a}
If (P' not empty) {
Make P' the corpus position set of STATE
Add (STATE,STATE',a) to tree edges
return STATE'
}
Else return NULL
}

Each state S in the RRE-tree represents the RRE corresponding to the edge labels on the path from root to S. For a state S with corpus position set P and corresponding RRE R, the goodness of the rule: 0→1 if R, is computed as:[3]

$$\text{Goodness\_0\_to\_1}(S) = \sum_{(j,k)\in P} \text{Score\_0\_to\_1}((j,k))$$

where

$$\text{Score\_0\_to\_1}((j,k)) = \begin{cases} 1 & \text{if } k = |C[j]| \\ & \wedge\, \text{Guess}[j] = 0 \\ & \wedge\, \text{Truth}[j] = 1 \\ -1 & \text{if } k = |C[j]| \\ & \wedge\, \text{Guess}[j] = 0 \\ & \wedge\, \text{Truth}[j] = 0 \\ 0 & \text{otherwise} \end{cases}$$

Similarly, we can compute the score for the rule : 1→0 if R. We then define Goodness(S) = max(Goodness_0_to_1(S),Goodness_1_to_0(S)) .

Returning to our example, for the 3 strings in this training corpus, the root node of the RRE-Tree would have the corpus position set: {(1,0),(2,0),(3,0)}. The root node corresponds to the null RRE, and so the position set consists of the beginning of each string in the training set. In figure 1 (at the end of the paper) we show a partial RRE-Tree. If we follow the edge labelled "dot" from the root node, we see it leads to a state with position set {(1,1),(2,1),(3,1)}, as a dot advances all positions by one.

[3] This assumes an RRE must match the entire string in order to accept it.

The square state in figure 1 represents the RRE: "dot* c" and the triangular state represents "a dot c". Both the square and triangular states have a corpus position set consisting of only one corpus position, namely the end of string 1, and both would have a goodness score of 1 for the corresponding 1→0 rule. If we prefer shorter rules, we will learn as our first rule in the rule list: 1→0 if dot* c. After applying this rule to the training corpus, all strings will be correctly labelled and training will terminate. If the stopping criterion were not met, we would apply the learned rule to change the values of our Guess array, then create a new RRE-tree, find the best state in that tree, and so on.

It is easy to extend the above algorithm to learn RREs instead of VRREs. Note, for instance, that the corpus position set for a state S with incoming edge labelled ~a can be found by taking the position set for the sibling of S with incoming edge labelled dot and deleting those corpus positions that are found in the position set for the sibling of S with incoming edge labelled a.

5 Optimizations

The algorithm above is exponential. There are some optimizations we can perform that make it feasible to apply the learning algorithm.

Optimization 1: Pruning states we know cannot be on the path from root to the best state. Define GoodPotential_0_to_1(S) as the number of sentences s in the training corpus for which Guess[s]=0, Truth[s]=1 and $\exists k : (s,k) \in \text{corpus\_position\_set}(S)$. We can similarly define GoodPotential_1_to_0(S), and then define

GoodPotential(S)=
    max(GoodPotential_0_to_1(S),
        GoodPotential_1_to_0(S))

As we construct the RRE-tree, we keep track of the largest Goodness(S) we have encountered. If that value is X, then for a state S', if GoodPotential(S') $\leq$ X, it is impossible for any path through S' to reach a state with a better goodness score than the best found thus far. We can check this condition when pushing states onto the stack, and when popping off the stack to be processed, and if the pruning condition is met, the state is discarded.

Optimization 2: Merging states with identical corpus position sets. If we are going to push a state onto the stack when a state already exists with an identical corpus position set, we do not need to retain both states. We may use heuristics to decide which of the states with identical corpus position sets we should keep (such as choosing the one with the shortest path to the root).

6 Experiments

To test whether learning RREs can improve disambiguation accuracy, we explored the task of confusion set disambiguation (Golding and Roth 1999). We trained and applied two different rule sequence learners, one which used the standard feature set for this problem (e.g. the identical feature set to that used in (Golding and Roth 1999) and (Mangu and Brill 1997) and described in the introduction, and one which learned RREs.[4] Because we wanted to determine what could be gained by using RREs, we ran an ablation study where we kept everything else constant across the two runs, and did not use performance enhancing techniques such as parameter tuning on held out data or classifier combination.

Both learners were given a window of +/- 5 words surrounding the ambiguity site. Context was not allowed to cross sentence boundaries. The training and test set were derived by finding all instances of the confusable words in the Brown Corpus, using the Penn Treebank parts of speech and tokenization (Marcus, Santorini et al. 1993), and then dividing this set into 80% for training and 20% for testing.

For the RRE-based system, we mapped the +/- 5 word window of context into a string as follows (where w_i is a word and t_i is a part of speech tag):

[4] The set of RREs is a superset of what can be learned using the standard feature set.

w_{i-5} t_{i-5} w_{i-4} t_{i-4} w_{i-3} t_{i-3} w_{i-2} t_{i-2} w_{i-1} t_{i-1} MIDDLE w_{i+1} t_{i+1} w_{i+2} t_{i+2} w_{i+3} t_{i+3} w_{i+4} t_{i+4} w_{i+5} t_{i+5}

where MIDDLE is the ambiguity site.

Both for execution time and space considerations for the learner and for fear of overtraining, we put a bound on the length of the RRE that could be learned.[5] We define an atomic RRE as any RRE derived without any concatenation operations. Then the length of an RRE is defined as the number of atomic RREs which that RRE is made up of. The atom "MIDDLE" is not counted in length.

Below we give two examples of rules that were learned for one confusion set:[6]

(1) past → passed if .* ~DT MIDDLE DOT IN
(2) past → passed if (~to)* NN MIDDLE

The first rule says to change the disambiguation guess to « passed » if the word before is not a determiner and the word after is a preposition. This matches contexts such as : « … they passed by … » while not matching contexts such as : « … made in the past by … » The second rule captures contexts such as : « … the hike passed the campground … » while not matching contexts such as : « … want to take a hike past the campground… »

In Table 1, we show test set results from running the rule sequence learner with both the standard set of features and with RRE-based features.[7] The results are sorted by training corpus size, with the raise/rise training corpus being the smallest and the then/than training corpus being the largest. Baseline accuracy is the accuracy attained on the test set by always picking the word that appears more frequently in the training set.

Conf. Pair	Baseline	Standard	RRE
Raise/Rise	53.6	75.0	**78.6**
Principal/Principle	64.5	80.6	**83.9**
Accept/Except	60.0	**94.5**	90.9
Affect/Effect	86.8	94.3	94.3
Lead/Led	53.6	89.3	89.3
Piece/Peace	51.1	83.0	83.0
Weather/Whether	79.7	84.6	**89.2**
Quiet/Quite	83.1	**100**	98.5
County/Country	75.6	78.2	**83.3**
Past/Passed	63.7	88.1	**89.3**
Amount/Number	74.1	83.3	**87.0**
Begin/Being	90.8	96.1	**96.7**
Among/Between	69.2	76.8	**80.8**
Then/Than	62.8	93.1	**93.4**

Table 1 Test Set Results: Standard vs RRE-Based Features

In Table 2 we see that the RRE-based system outperforms the standard system on 9 of the confusion sets, the standard system outperforms the RRE-based system on 2 and the two systems attain identical results on 3. We see that the relative performance of the RRE-based learner is better overall on the larger training sets than on the smaller sets. This is to be expected, as more data is needed to support learning the more expressive RRE-based rules.

	RRE Better	Standard Better	Identical
All Confusables	9	2	3
7 Smallest Sets	3	1	3
7 Largest Sets	6	1	0

Table 2 Performance Analysis Across Different Sets

Pooling all of the test sets into one big set, the RRE-based system achieves an overall accuracy of 89.9%, compared to 88.5% for the standard learner. Weighting each confusion pair equally, the RRE-based system achieves an

[5] Note that this does not imply a bound on the length of a string to which an RRE can apply.

[6] DT= determiner, IN = preposition, NN = singular noun.

[7] While these results look worse than those achieved by other systems, as reported in (Golding and Roth, 1999), we used different data splits and tokenization. Our baseline accuracies are significantly lower than the baselines for their test sets. If we account for this by instead measuring percent error reduction compared to baseline accuracy, then our average reduction is better than that reported for the BaySpell system, but worse than that of WinSpell. If we add voting to our system (WinSpell employs voting), then we attain results on par with WinSpell.

overall accuracy of 88.4%, compared to 86.9% for the standard learner.

Conclusions

The RRE-based rule sequence learner presented above is able to learn rules using more expressive conditions than what is typically used for disambiguation tasks in natural language processing. These regular-expression based conditions lead to higher accuracy than what is achieved when using the same learning paradigm with the traditionally used feature set. We hope that other learning algorithms can benefit from the ideas presented here and that the idea of learning RREs can be generalized to allow other learners to incorporate more powerful features as well.

References

Bahl, L., P. Brown, et al. (1989). "A Tree-Based Language Model for Natural Language Speech Recognition." IEEE Transactions on Acoustics, Speech and Signal Processing **37**: 1001-1008.

Brill, E. (1995). "Transformation-Based error-driven learning and natural language processing: a case study in part of speech tagging." Computational Linguistics.

Chelba, C. and F. Jelinek (1998). Exploiting Syntactic Structure for Language Modeling. Proceedings of Coling/ACL, Montreal, Canada.

Even-Zohar, Y. and D. Roth (2000). A Classification Approach to Word Prediction. Proceedings of NAACL, Seattle, Wa.

Golding, A. and D. Roth (1999). "A Winnow-Based Approach to Context-Sensitive Spelling Correction." Machine Learning.

Hopcroft, J. and J. Ullman (1979). Introduction to Automata Theory, Languages and Computation, Addison-Wesley.

Mangu, L. and E. Brill (1997). Automatic Rule Acquisition for Spelling Correction. Proceedings of the International Conference on Machine Learning, Nashville, Tn.

Marcus, M., B. Santorini, et al. (1993). "Building a large annotated corpus of English: the Penn Treebank." Computational Linguistics.

Pietra, S. D., V. D. Pietra, et al. (1994). Inference and Estimation of a Long-Range Trigram Model. Proceedings of the Second International Colloquium on Grammatical Inference, Alicante, Spain.

Samuellson, C., P. Tapanainen, et al. (1996). Inducing Constraint Grammars. Grammatical Inference: Learning Syntax from Sentences. L. Miclet and C. D. l. Huguera, Springer. **1147**.

Saul, L. and F. Pereira (1997). Aggregate and mixed-order Markov models for statistical language processing. Proceedings of the Second Conference on EMNLP.

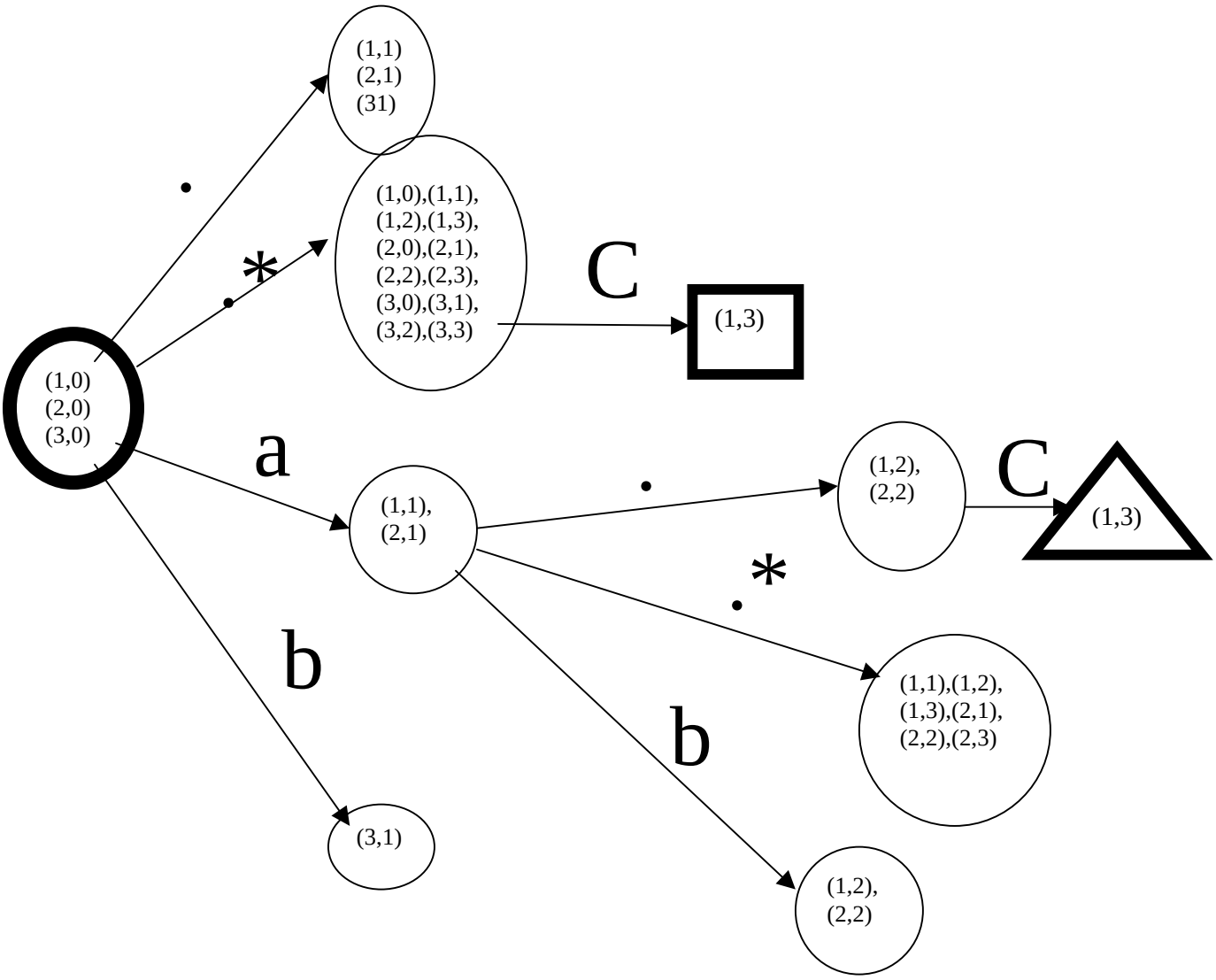

Figure 1 : A Partial RRE-Tree