On the Intersection of Finite State Automata and Definite Clause Grammars

Gertjan van Noord
Vakgroep Alfa-informatica
Rijksuniversiteit Groningen

Abstract

An elegant framework for robust parsing was presented by Bernard Lang on last year's TWLT. In Lang's approach parsing is viewed as the calculation of the intersection of a FSA (the input) and a CFG. Viewing the input for parsing as a FSA rather than as a string combines well with standard approaches in speech understanding systems in which typically a word lattice is produced as an intermediate result. Furthermore, certain techniques for robust parsing can be modelled as finite state transducers.

In this paper we investigate how we can generalize this approach for unification grammars. In particular we will concentrate how we might calculate the intersection of a FSA and a DCG. It is shown that existing parsing algorithms can be easily extended for FSA inputs. However, we also show that the termination properties change drastically: we show that it is undecidable whether the intersection of a FSA and an off-line parsable DCG is empty.

Furthermore we discuss techniques to cope with the problem.

1 Introduction

In this paper we are concerned with the syntactic analysis phase of a speech analysis system. The input for the syntactic analysis phase in such a system is a *word lattice*, and the output is a packed representation of all possible parse trees. Syntactic analysis is performed on the basis of some grammar G. Following Bernard Lang we can thus characterize the syntactic analysis phase as the computation of the intersection of a finite state automaton (the word-lattice) and the grammar G. This intersection is a grammar that derives exactly all parse trees for the input.

Note that we allow the input to be a full FSA (possibly including cycles, etc.) in order for certain techniques to handle ill-formed input to be applied. Whereas an ordinary word-graph always defines a finite language, a FSA of course can easily define an infinite number of sentences.

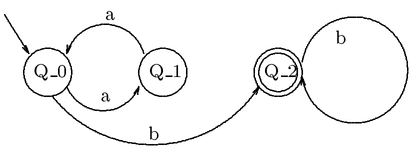
It can be shown that the computation of the intersection of a FSA and a CFG requires only a minimal generalization of existing parsing algorithms. We simply replace the usual string positions with the names of the states in the FSA. It is therefore straightforward to show that the complexity of this process is cubic in the number of states of the FSA (in the case of ordinary parsing the number of states equals $n + 1$) (Lang, 1974; Billot and Lang, 1989) (assuming the grammar is in bilinear format).

In this paper we investigate whether the same techniques can be applied in case the grammar G is a constraint-based grammar rather than a CFG. For specificity we will take G to be a *Definite Clause Grammar* (DCG) Pereira and Warren (1980). A DCG is a simple example of a family of grammar formalisms that are widely used in natural language analysis (and generation). The main findings of this paper can be extended to the other members of that family of constraint-based grammar formalisms.

2 The intersection of a CFG and a FSA

The calculation of the intersection of a CFG and a FSA is very simple (Bar-Hillel, Perles, and Shamir, 1961). However this construction yields an enormous amount of rules that are 'useless'. Furthermore the resulting parse forest grammar might define an empty language (if the intersection was empty). Luckily 'ordinary' recognizers/parsers for CFG can be easily generalized to construct this intersection (usually yielding a much smaller grammar than in the construction of Bar-Hillel, Perles, and Shamir (1961)). Checking whether the intersection is empty or not is then usually very simple as well: only in the latter case will the parser terminate succesfully.

Consider the following definite clause specification of a FSA. The following conventions apply. We define the transition relation using the relation `trans/3`. For start states, the relation start/1 should hold, and for final states the relation final/1 should hold. Thus the following FSA, defining the regular language $L = (aa)^*b^+$ (i.e. an even number of a's followed by at least one b) is given as:




```
start(q0).  final(q2).

trans(q0,a,q1).  trans(q1,a,q0).
trans(q0,b,q2).  trans(q2,b,q2).
```

A context-free grammar is represented as a definite-clause specification as follows. We do not wish to define the sets of terminal and non-terminal symbols explicitly, these can be understood from the rules that are defined using the relation `rule/2`, and where symbols of the rhs are prefixed with '-' in the case of terminals and '+' in the case of non-terminals. The relation `top/1` should hold for the start symbol. The language $L' = a^n b^n$ is defined as:

```
top(s).

rule(s,[-a,+s,-b]).  rule(s,[]).
```

In order to illustrate how ordinary parsers can be used to compute the intersection of a FSA and a CFG consider the definite-clause specification of a top-down parser. This parser runs in cubic time if implemented using Earley deduction or XOLDT resolution. It is assumed that the input string is represented by the `trans/3` predicate.

```
parse(P0,P) :-
    top(Cat), parse(+Cat,P0,P).

parse(-Cat,P0,P) :-
    trans(P0,Cat,P),
    side_effect(p(Cat,P0,P) --> Cat).
parse(+Cat,P0,P) :-
    rule(Cat,Ds),
    parse_ds(Ds,P0,P,His),
    side_effect(p(Cat,P0,P) --> His).

parse_ds([],P,P,[]).
parse_ds([H|T],P0,P,[p(H,P0,P1)|His]) :-
    parse(H,P0,P1),
    parse_ds(T,P1,P,His).
```

The predicate `side_effect` is used to construct the parse forest grammar. The predicate always succeeds, and as a side-effect asserts that its argument is a rule of the parse forest grammar. For the sentence 'a a b b' we obtain the parse forest grammar:

```
p(s,2,2) --> [].
p(s,1,3) -->
    [p(-a,1,2),p(+s,2,2),p(-b,2,3)].
p(s,0,4) -->
    [p(-a,0,1),p(+s,1,3),p(-b,3,4)].
p(a,1,2) --> a.
p(a,0,1) --> a.
p(b,2,3) --> b.
p(b,3,4) --> b.
```

The reader easily verifies that indeed this grammar generates (a isomorphism of) the single parse tree of this example, assuming of course that the start symbol is `p(s,0,4)`. In the parse-forest grammar, complex symbols are non-terminals, atomic symbols are terminals.

Interestingly, nothing needs to be changed to use the same parser for the computation of the intersection of a FSA and a CFG. If our input 'sentence' now is the definition of `trans/3` as given above, we obtain the following parse forest grammar (where the start symbol is `p(s,q0,q2)`):

```
p(s,q0,q0) --> [].
```

```
p(s,q1,q1) --> [].
p(s,q1,q2) -->
    [p(-a,q1,q0),p(+s,q0,q0),p(-b,q0,q2)].
p(s,q0,q2) -->
    [p(-a,q0,q1),p(+s,q1,q2),p(-b,q2,q2)].
p(s,q1,q2) -->
    [p(-a,q1,q0),p(+s,q0,q2),p(-b,q2,q2)].
p(a,q0,q1) --> a.
p(a,q1,q0) --> a.
p(b,q0,q2) --> b.
p(b,q2,q2) --> b.
```

Thus, even though we now use the same parser for an infinite set of input sentences (represented by the FSA) the parser still is able to come up with a parse forest grammar. A possible derivation for this grammar is the following (abbreviated) parse tree in figure 1.

3 The intersection of a DCG and a FSA

First note that the problem of calculating the intersection of a DCG and a FSA can be solved trivially by a generalization of the construction used by Bar-Hillel, Perles, and Shamir (1961). However, if we use that method we will end up with an enormously large forest grammar that is not even guaranteed to contain solutions. Therefore, we are interested in methods that only generate a small subset of this; if the intersection is empty we want an empty parse-forest grammar. The straightforward approach is to generalize existing recognition algorithms.

The same techniques that are used for calculating the intersection of a FSA and a CFG can be applied in the case of DCGs. In order to compute the intersection of a DCG and a FSA we assume that FSA's are represented as before. DCGs are represented using the same notation we used for context-free grammars, but now of course the category symbols can be first-order terms of arbitrary complexity (note that without loss of generality we don't take into account DCGs having external actions defined in curly braces).

3.1 Undecidability

This confronts us with an undecidability problem: the recognition problem for DCGs is undecidable. A fortiori the problem of deciding whether the intersection of a FSA and a DCG is empty or not is undecidable.

In the case of parsing, this undecidability result is usually circumvented by considering subsets of DCGs which can be recognized effectively. For example, we can restrict the attention to DCGs of which the context-free skeleton does not contain cycles. Recognition for such 'off-line parsable' grammars is decidable.

Most existing constraint-based parsing algorithms will terminate for grammars that exhibit the property that for each string there is only a finite number of possible derivations. Note that off-line parsability is one possible way of ensuring that this is the case.

This observation is not very helpful in establishing insights concerning interesting subclasses of DCGs for which termination can be guaranteed (in the case of FSA input). The reason is that there are now two sources of recursion: in the DCG and in the FSA (cycles). As we saw earlier: even for CFG it holds that there can be an infinite number of analyses for a given FSA (but in the CFG this of course does not imply undecidability). In the appendix we give a simple proof that the question whether the intersection of a FSA and an off-line parsable DCG is empty or not is undecidable.

3.2 What to do?

The following approaches towards the undecidability problem can be taken:

- limit the power of the FSA
- limit the power of the DCG
- compromise completeness
- compromise soundness

These approaches are discussed now in turn.

Limit the FSA Rather than assuming the input for parsing is a FSA in its full generality, we might assume that the input is an ordinary word graph (a FSA without cycles).

Thus the techniques for robust processing that give rise to such cycles cannot be used. One example is the processing of an unknown sequence of

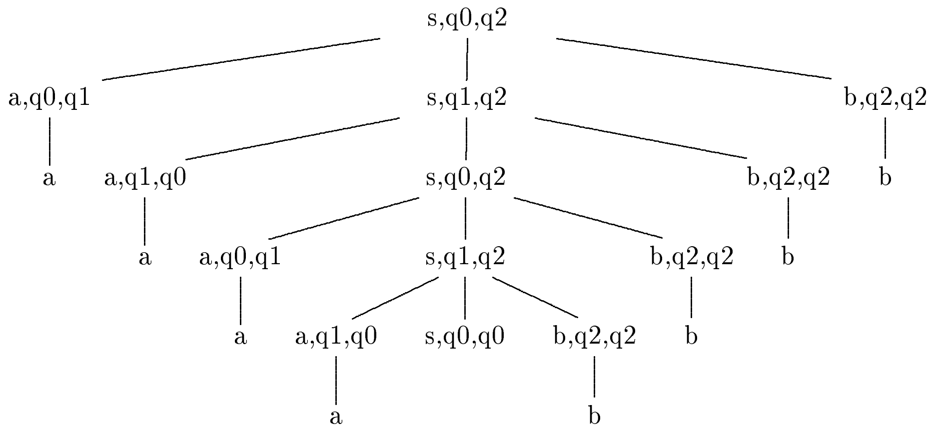


Figure 1: A parse-tree extracted from the parse forest grammar

words, e.g. in case there is noise in the input and it is not clear how many words have been uttered during this noise.

It is not clear to me right now what we loose (in practical terms) if we give up such cycles.

Note that it is easy to verify that parsing is decidable on the basis of a FSA without cycles and a DCG that is off-line parsable.

Limit the DCG Another approach is to limit the size of the categories that are being employed. This is the GPSG and F-TAG approach. In that case we are not longer dealing with DCGs but rather with CFGs (which have been shown to be insufficient in general for the description of natural languages).

Compromise completeness Completeness in this context means: the parse forest grammar contains all possible parses. It is possible to compromise here, in such a way that the parser is guaranteed to terminate.

For example, if we assume that each edge in the FSA is associated with a probability it is possible to define a threshold such that each partial result that is derived has a probability higher than the threshold. Thus, it is still possible to have cycles in the FSA, but anytime the cycle is 'used' the probability decreases and if too many cycles are encountered the threshold will cut of that derivation.

For any threshold it is the case that the intersection problem of off-line parsable DCGs and FSA is decidable.

Compromise soundness Soundness in this context should be understood as the property that all parse trees in the parse forest grammar are valid parse trees. A possible way to ensure termination is to remove all constraints from the DCG and parse according to this context-free skeleton. The resulting parse-forest grammar will be too general most of the times.

A practical variation can be conceived as follows. From the DCG we take its context-free skeleton. This skeleton is obtained by removing the constraints from each of the grammar rules. Then we compute the intersection of the skeleton with the input FSA. This results in a parse forest grammar. Finally, we add the corresponding constraints from the DCG to the grammar rules of the parse forest grammar.

This has the advantage that the result is still sound, although the size of the parse forest grammar is not optimal (as a consequence it is not guaranteed that the parse forest grammar contains a parse tree).

References

Bar-Hillel, Y., M. Perles, and E. Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprach-Wissenschaft und Kommunicationsforschung*, 14:143–172. Reprinted in Bar-Hillel's Language and Information – Selected Essays on their Theory and Application, Addison Wesley series in Logic, 1964, pp. 116-150.

Billot, S. and B. Lang. 1989. The structure of shared parse forests in ambiguous parsing. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver.

Grosz, Barbara, Karen Sparck Jones, and Bonny Lynn Webber, editors. 1986. *Readings in Natural Language Processing*. Morgan Kaufmann.

Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley.

Lang, Bernard. 1974. Deterministic techniques for efficient non-deterministic parsers. In J. Loeckx, editor, *Proceedings of the Second Colloquium on Automata, Languages and Programming.* Also: Rapport de Recherche 72, IRIA-Laboria, Rocquencourt (France).

Pereira, Fernando C.N. and David Warren. 1980. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13. reprinted in (Grosz, Jones, and Webber, 1986).

A Intersection of FSA and off-line parsable DCG is undecidable

Here I show that the question whether the intersection of a FSA and an off-line parsable DCG is empty is undecidable. A yes-no problem is *undecidable* (cf. Hopcroft and Ullman (1979, pp.178-179)) if there is no algorithm that takes as its input an *instance* of the problem and determines whether the answer to that instance is 'yes' or 'no'. An instance of a problem consists of a particular choice of the *parameters* of that problem.

I use Post's Correspondence Problem (PCP) as a well-known undecidable problem. I show that if the above mentioned intersection problem were decidable, then we could solve the PCP too. The following definition and example of a PCP are taken from Hopcroft and Ullman (1979)[chapter 8.5].

An instance of PCP consists of two lists, $A = v_1 \dots v_k$ and $B = w_1 \dots w_k$ of strings over some alphabet Σ. This instance has *a solution* if there is any sequence of integers $i_1 \dots i_m$, with $m \geq 1$, such that

$$v_{i_1}, v_{i_2}, \dots, v_{i_m} = w_{i_1}, w_{i_2}, \dots, w_{i_m}.$$

The sequence $i_1, \dots, i_m$ is a solution to this instance of PCP. As an example, assume that $\Sigma = \{0, 1\}$. Furthermore, let $A = \langle 1, 10111, 10 \rangle$ and $B = \langle 111, 10, 0 \rangle$. A solution to this instance of PCP is the sequence 2,1,1,3 (obtaining the sequence 101111110). For an illustration, cf. figure 2.

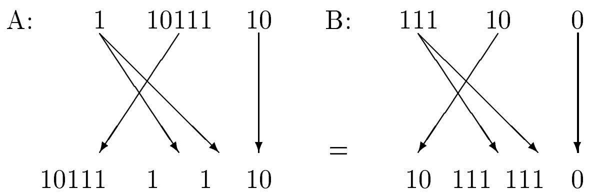


Figure 2: Illustration of a solution of a PCP problem.

Clearly there are PCP's that do not have a solution. Assume again that $\Sigma = \{0, 1\}$. Furthermore let $A = \langle 1 \rangle$ and $B = \langle 0 \rangle$. Clearly this PCP does not have a solution. In general, however, the problem whether some PCP has a solution or not is not decidable. This result is proved by Hopcroft and Ullman (1979) by showing that the halting problem for Turing Machines can be encoded as an instance of Post's Correspondence Problem.

First I give a simple algorithm to encode any instance of a PCP as a pair, consisting of a FSA and an off-line parsable DCG, in such a way that the question whether there is a solution to this PCP is equivalent to the question whether the intersection of this FSA and DCG is empty.

Encoding of PCP.

1. For each $1 \leq i \leq k$ (k the length of lists A and B) define a DCG rule (the $i-th$ member of A is $a_1 \dots a_m$, and the i–th member of B is $b_1 \dots b_n$): $r([a_1 \dots a_m|A], A, [b_1 \dots b_n|B], B) \rightarrow [x]$.

2. Furthermore, there is a rule $r(A_0, A, B_0, B) \rightarrow r(A_0, A_1, B_0, B_1), r(A_1, A, B_1, B)$.

3. Furthermore, there is a rule $s \rightarrow r(X, [\,], X, [\,])$. Also, s is the start category of the DCG.

4. Finally, the FSA consists of a single state q which is both the start state and the final state, and a single transition $\delta(q, x) = q$.

Observe that the DCG is off-line parsable.

The underlying idea of the algorithm is really very simple. For each pair of strings from the lists A and B there will be one lexical entry where these strings are represented by a difference-list encoding. Furthermore there is a general combination rule that simply concatenates A-strings and concatenates B-strings. Finally the rule for s states that in order to construct a succesful top category the A and B lists must match.

The resulting DCG, FSA pair for the example PCP is:

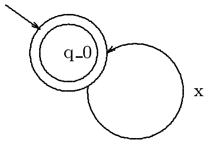
```
s --> r(X,[],X,[]).

r(A0,A,B0,B) -->
    r(A0,A1,B0,B1),
    r(A1,A, B1,B ).

r([1|A],A,[1,1,1|B],B) --> [x].

r([1,0,1,1,1|A],A,[1,0|B],B) --> [x].

r([1,0|A],A,[0|B],B) --> [x].
```



Proposition The question whether the intersection of a FSA and an off-line parsable DCG is empty is undecidable.

Proof. Suppose the problem *was* decidable. In that case there would exist an algorithm for solving the problem. This algorithm could then be used to solve the PCP, because a PCP π has a solution if and only if its encoding given above as a FSA and an off-line parsable DCG is not empty. The PCP problem however is known to be undecidable. Hence the intersection question is undecidable too.