Bottom-Up Earley Deduction

Gregor Erbach*
University of the Saarland
Computational Linguistics
D-66041 Saarbrücken, Germany
erbach@coli.uni-sb.de

Abstract

We propose a bottom-up variant of Earley deduction. Bottom-up deduction is preferable to top-down deduction because it allows incremental processing (even for head-driven grammars), it is data-driven, no subsumption check is needed, and preference values attached to lexical items can be used to guide best-first search. We discuss the scanning step for bottom-up Earley deduction and indexing schemes that help avoid useless deduction steps.

1 Introduction

Recently, there has been a lot of interest in Earley deduction [10] with applications to parsing and generation [13, 6, 7, 3].

Earley deduction is a very attractive framwork for natural language processing because it has the following properties and applications.

- Memoization and reuse of partial results

- Incremental processing by addition of new items

- Hypothetical reasoning by keeping track of dependencies between items

- Best-first search by means of an agenda

Like Earley's algorithm, all of these approaches operate top-down (backward chaining). The interest has naturally focussed on top-down methods because they are at least to a certain degree goal-directed.

In this paper, we present a bottom-up variant of Earley deduction, which we find advantageous for the following reasons:

Incrementality: Portions of an input string can be analysed as soon as they are produced (or generated as soon as the what-to-say component has decided to verbalize them), even for grammars where one cannot assume that the left-corner has been predicted before it is scanned.

Data-Driven Processing: Top-down algorithms are not well suited for processing grammatical theories like Categorial Grammar or HPSG that would only allow very general predictions because they make use of general schemata instead of construction-specific rules. For these grammars data-driven bottom-up processing is more appropriate. The same is true for large-coverage rule-based grammars which lead to the creation of very many predictions.

Subsumption Checking: Since the bottom-up algorithm does not have a prediction step, there is no need for the costly operation of subsumption checking.[1]

Search Strategy: In the case where lexical entries have been associated with preference in-

---

[1]Subsumption checking may still be needed to filter out spurious ambiguities.

formation, this information can be exploited to guide the heuristic search.

2 Bottom-up Earley Deduction

Earley deduction [10] is based on grammars encoded as definite clauses. The instantiation (prediction) rule of top-down Earley deduction is not needed in bottom-up Earley deduction, because there is no prediction. There is only one inference rule, namely the reduction rule (1).[2] In (1), X, G and G' are literals, Ω is a (possibly empty) sequence of literals, and σ is the most general unifier of G and G'. The leftmost literal in the body of a non-unit clause is always the selected literal.

$$\frac{\begin{array}{c} X \leftarrow G \wedge \Omega \\ G' \leftarrow \end{array}}{\sigma(X \leftarrow \Omega)} \quad (1)$$

In principle, this rule can be applied to any pair of unit clauses and non-unit clauses of the program to derive any consequences of the program. In order to reduce this search space and achieve a more goal-directed behaviour, the rule is not applied to any pair of clauses, but clauses are only selected if they can contribute to a proof of the goal. The set of selected clauses is called the *chart*.[3] The selection of clauses is guided by a scanning step (section 2.1) and indexing of clauses (section 2.2).

2.1 Scanning

The purpose of the scanning step, which corresponds to lexical lookup in chart parsers, is to look up base cases of recursive definitions to serve as a starting point for bottom-up processing. The scanning step selects clauses that can appear as leaves in the proof tree for a given goal G.

Consider the following simple definition of an HPSG, with the recursive definition of the predicate sign/1.[4]

[2]This rule is called combine by Earley, and is also referred to as the fundamental rule in the literature on chart parsing.

[3]The chart differs from the *state* of [10] in that clauses in the chart are indexed (cf. section 2.2).

[4]We use feature terms in definite clauses in addition to Prolog terms. f:X denotes a feature structure where X is the value of feature f, and X & Y denotes the conjunction

```
sign(X) <- phrasal_sign(X).
sign(X) <- lexical_sign(X).

phrasal_sign(X & dtrs:(head_dtr:HD &
                       comp_dtr:CD)  ) <-
    sign(HD),
    sign(CD),
    principles(X,HD,CD).

principles(X,HD,CD) <-
    constituent_order_principle(X,HD,CD),
    head_feature_principle(X,HD),
    ...

constituent_order_principle(phon:X_Ph,
                            phon:HD_Ph,
                            phon:CD_Ph) <-
    sequence_union(CD_Ph,HD_Ph,X_Ph).
```

The predicate sign/1 is defined recursively, and the base case is the predicate lexical_sign/1. But, clearly it is not restrictive enough to find only the predicate name of the base case for a given goal. The base cases must also be instantiated in order to find those that are useful for proving a given goal. In the case of parsing, the lookup of base cases (lexical items) will depend on the words that are present in the input string. This is implied by the first goal of the predicate principles/3, the constituent order principle, which determines how the PHON value of a constituent is constructed from the PHON values of its daughters. In general, we assume that the constituent order principle makes use of a linear and non-erasing operation for combining strings.[5] If this is the case, then all the words contained in the PHON value of the goal can have their lexical items selected as unit clauses to start bottom-up processing.

For generation, an analogous condition on logical forms has been proposed by Shieber [13] as the "semantic monotonicity condition," which requires that the logical form of every base case must subsume some portion of the goal's logical form.

Base case lookup must be defined specifically for different grammatical theories and directions of processing by the predicate lookup/2, whose first argument is the goal and whose second argument is the selected base case. The following

of the feature terms X and Y.

[5]There is an obvious connection to the Linear Context-Free Rewriting Systems (LCFRS) [15, 16].

clause defines the lookup relation for parsing with
HPSG.

```
% lookup(+Goal,-BaseCase)
lookup(phon:PhonList,
        lexical_sign(phon:[Word] & synsem:X)
        ) <-
      member(Word,PhonList),
      lexicon(Word,X).
```

Note that the base case clauses can become
further instantiated in this step. If concatena-
tion (of difference lists) is used as the operation
on strings, then each base case clause can be in-
stantiated with the string that follows it. This
avoids combination of items that are not adjacent
in the input string.

```
lookup(phon:PhonList,
        lexical_sign(phon:[Word|Suf]-Suf &
                      synsem: Synsem)
        ) <-
      append(_,[Word|Suf],PhonList),
      lexicon(Word,Synsem).
```

In bottom-up Earley deduction, the first step
towards proving a goal is perform lookup for the
goal, and to add all the resulting (unit) clauses to
the chart. Also, all non-unit clauses of the pro-
gram, which can appear as internal nodes in the
proof tree of the goal, are added to the chart.

The scanning step achieves a certain degree of
goal-directedness for bottom-up algorithms bec-
ause only those clauses which can appear as lea-
ves in the proof tree of the goal are added to the
chart.

2.2 Indexing

An item in normal context-free chart parsing can
be regarded as a pair $\langle R,S \rangle$ consisting of a dotted
rule R and the substring S that the item covers
(a pair of starting and ending position). The fun-
damental rule of chart parsing makes use of these
string positions to ensure that only adjacent sub-
strings are combined and that the result is the
concatenation of the substrings.

In grammar formalisms like DCG or HPSG, the
complex nonterminals have an argument or a fea-
ture (PHON) that represents the covered substring
explicitly. The combination of the substrings is

explicit in the rules of the grammar. As a conse-
quence, Earley deduction does not need to make
use of string positions for its clauses, as Pereira
and Warren [10] point out.

Moreover, the use of string positions known
from chart parsing is too inflexible because it al-
lows only concatenation of adjacent contiguous
substrings. In linguistic theory, the interest has
shifted from phrase structure rules that combine
adjacent and contiguous constituents to

- principle-based approaches to grammar that
  state general well-formedness conditions in-
  stead of describing particular constructions
  (e.g. HPSG)

- operations on strings that go beyond conca-
  tenation (head wrapping [11], tree adjoining
  [15], sequence union [12]).

The string positions known from chart parsing
are also inadequate for generation, as pointed out
by Shieber [13] in whose generator all items go
from position 0 to 0 so that any item can be com-
bined with any item.

However, the string positions are useful as an
indexing of the items so that it can be easily detec-
ted whether their combination can contribute to a
proof of the goal. This is especially important for
a bottom-up algorithm which is not goal-directed
like top-down processing. Without indexing, there
are too many combinations of items which are use-
less for a proof of the goal, in fact there may be in-
finitely many items so that termination problems
can arise.

For example, in an order-monotonic grammar
formalism that uses sequence union as the opera-
tion for combining strings, a combination of items
would be useless which results in a sign in which
the words are not in the same order as in the input
string [14].

We generalize the indexing scheme from chart
parsing in order to allow different operations for
the combination of strings. Indexing improves ef-
ficiency by detecting combinations that would fail
anyway and by avoiding combinations of items
that are useless for a proof of the goal.

We define an item as a pair of a clause Cl and
an index Idx, written as $\langle Cl, Idx \rangle$.

Below, we give some examples of possible indexing schemes. Other indexing schemes can be used if they are needed.

1. **Non-reuse of Items:** This is useful for LCFRS, where no word of the input string can be used twice in a proof, or for generation where no part of the goal logical form should be verbalized twice in a derivation.

2. **Non-adjacent combination:** This indexing scheme is useful for order-monotonic grammars.

3. **Non-directional adjacent combination:** This indexing is used if only adjacent constituents can be combined, but the order of combination is not prescribed (e.g. non-directional basic categorial grammars).

4. **Directional adjacent combination:** This is used for grammars with a "context-free backbone."

5. **Free combination:** Allows an item to be used several times in a proof, for example for the non-unit clauses of the program, which would be represented as items of the form $\langle X \leftarrow G_1 \wedge \ldots \wedge G_n, \text{free} \rangle$.

The following table summarizes the properties of these five combination schemes. *Index 1 (I1)* is the index associated with the non-unit clause, *Index 2 (I2)* is associated with the unit clause, and $I1 \star I2$ is the result of combining the indices.

	Index 1 I1	Index 2 I2	Result $I1 \star I2$	Note
1.	X	Y	$X \cup Y$	$X \cap Y = \emptyset$
2.	X	Y	$X \odot Y$	
3.	$X + Y$	$Y + Z$	$X + Z$	
	$Y + Z$	$X + Y$	$X + Z$	
4.	$X - Y$	$Y - Z$	$X - Z$	
5.	X	'free'	X	
	'free'	X	X	

In case 2 ("non-adjacent combination"), the indices X and Y consist of a set of string positions, and the operation $\odot$ is the union of these string positions, provided that no two string positions from X and Y do overlap.

In (2), the reduction rule is augmented to handle indices. $X \star Y$ denotes the combination of the indices X and Y.

$$\frac{\langle X \leftarrow G \wedge \Omega, I1 \rangle \quad \langle G' \leftarrow, I2 \rangle}{\langle \sigma(X \leftarrow \Omega), I1 \star I2 \rangle} \quad (2)$$

With the use of indices, the lookup relation becomes a relation between goals and *items*. The following specification of the lookup relation provides indexing according to string positions as in a chart parser (usable for combination schemes 2, 3, and 4).

```
lookup(phon:PhonList,
       item(lexical_sign(phon:[Word] &
                         synsem:X),
            Begin-End)
      ) <-
nth_member(Word,Begin,End,PhonList),
lexicon(Word,X).

nth_member(X,0,1,[X|_]).
nth_member(X,N1,N2,[_|R]) <-
    nth_member(X,N0,N1,R),
    N2 is N1 + 1.
```

2.3 Goal Types

In constraint-based grammars there are some predicates that are not adequately dealt with by bottom-up Earley deduction, for example the Head Feature Principle and the Subcategorization Principle of HPSG. The Head Feature Principle just unifies two variables, so that it can be executed at compile time and need not be called as a goal at runtime. The Subcategorization Principle involves an operation on lists (append/3 or delete/3 in different formalizations) that does not need bottom-up processing, but can better be evaluated by top-down resolution if its arguments are sufficiently instantiated. Creating and managing items for these proofs is too much of a computational overhead, and, moreover, a proof may not terminate in the bottom-up case because infinitely many consequences may be derived from the base case of a recursively defined relation.

In order to deal with such goals, we associate the goals in the body of a clause with goal types. The goals that are relevant for bottom-up Earley deduction are called *waiting goals* because they wait until they are activated by a unit clause that unifies with the goal.[6] Whenever a unit clause is

---

[6]The other goal types are top-down goals (top-down

combined with a non-unit clause all goals up to the first waiting goal of the resulting clause are proved according to their goal type, and then a new clause is added whose selected goal is the first waiting goal.

In the following inference rule for clauses with mixed goal types, Ξ is a (possibly empty) sequence of goals without any waiting goals, and Ω is a (possibly empty) sequence of goals starting with a waiting goal. σ is the most general unifier of G and G', and the substitution τ is the solution which results from proving the sequence of goals Ξ.

$$\frac{\langle X \leftarrow G \wedge \Xi \wedge \Omega, I1 \rangle \quad \langle G' \leftarrow, I2 \rangle}{\langle \tau\sigma(X \leftarrow \Omega), I1 \star I2 \rangle} \tag{3}$$

2.4 Correctness and Completeness

In order to show the correctness of the system, we must show that the scanning step only adds consequences of the program to the chart, and that any items derived by the inference rule are consequences of the program clauses. The former is easy to show because all clauses added by the scanning step are *instances* of program clauses, and the inference rule performs a resolution step whose correctness is well-known in logic programming. The other goal types are also proved by resolution.

There are two potential sources of incompleteness in the algorithm. One is that the scanning step may not add all the program clauses to the chart that are needed for proving a goal, and the other is that the indexing may prevent the derivation of a clause that is needed to prove the goal.

In order to avoid incompleteness, the scanning step must add *all* program clauses that are needed for a proof of the goal to the chart, and the combination of indices may only fail for inference steps which are useless for a proof of the goal. That

depth-first search), x-corner goals (which combine bottom-up and top-down processing like left-corner or head-corner algorithms), Prolog goals (which are directly executed by Prolog for efficiency or side-effects), and chart goals which create a new, independent chart for the proof of the goal. Dörre [3] proposes a system with two goal types, namely *trigger goals*, which lead to the creation of items and other goals which don't.

the lookup relation and the indexing scheme satisfy this property must be shown for particular grammar formalisms.

In order to keep the search space small (and finite to ensure termination) the scanning step should (ideally) add *only* those items that are needed for proving the goal to the chart, and the indexing should be chosen in such a way that it excludes derived items that are useless for a proof of the goal.

3 Best-First Search

For practical NL applications, it is desirable to have a best-first search strategy, which follows the most promising paths in the search space first, and finds preferred solutions before the less preferred ones.

There are often situations where the criteria to guide the search are available only for the base cases, for example

- weighted word hypotheses from a speech recognizer

- readings for ambigous words with probabilities, possibly assigned by a stochastic tagger (cf. [2])

- hypotheses for correction of string errors which should be delayed [5]

Goals and clauses are associated with preference values that are intended to model the degree of confidence that a particular solution is the 'correct' one. Unit clauses are associated with a numerical preference value, and non-unit clauses with a formula that determines how its preference value is computed from the preference values of the goals in the body of the clause. Preference values can (but need not) be interpreted as probabilities.[7]

The preference values are the basis for giving priorities to items. For unit clauses, the priority is identified with the preference value. For non-unit clauses, where the preference formula may contain uninstantiated variables, the priority is the value of the formula with the free variables instantiated to the highest possible preference value (in case

---

[7]For further details and examples see [4] and [5].

of an interpretation as probabilities: 1), so that the priority is equal to the maximal possible preference value for the clause.[8]

The implementation of best-first search does not combine new items with the chart immediately, but makes use of an agenda [8], on which new items are ordered in order of descending priority. The following is the algorithm for bottom-up best-first Earley deduction.

procedure prove($Goal$):
– initialize-agenda($Goal$)
– consume-agenda
– for any item $\langle G, I \rangle$
  – return mgu($Goal, G$) as solution if it exists

procedure initialize-agenda($Goal$):
– for every unit clause UC in lookup($Goal, UC$)
  – create the index I for UC
  – add item $\langle UC, I \rangle$ to agenda
– for every non-unit program clause $H \leftarrow Body$
  – add item $\langle H \leftarrow Body, \text{free} \rangle$ to agenda

procedure add item I to agenda
– compute the priority of I
– agenda := agenda $\cup \{I\}$

procedure consume-agenda
– while agenda is not empty
  – remove item I with highest priority from agenda
  – add item I to chart

procedure add item $\langle C, I1 \rangle$ to chart
– chart := chart $\cup \{\langle C, I1 \rangle\}$
– if C is a unit clause
  – for all items $\langle H \leftarrow G \wedge \Xi \wedge \Omega, I2 \rangle$
    – if $I = I2 \star I1$ exists
    and $\sigma = $ mgu(C, G) exists
    and goals Ξ are provable with solution τ
    then add item $\langle \tau\sigma(H \leftarrow \Omega), I \rangle$ to agenda
– if $C = H \leftarrow G \wedge \Xi \wedge \Omega$ is a non-unit clause
  – for all items $\langle G' \leftarrow, I2 \rangle$
    – if $I = I1 \star I2$ exists
    and $\sigma = $ mgu(G, G') exists
    and goals Ξ are provable with solution τ
    then add item $\langle \tau\sigma(H \leftarrow \Omega), I \rangle$ to agenda

The algorithm is parametrized with respect to the relation lookup/2 and the choice of the indexing scheme, which are specific for different grammatical theories and directions of processing.

---

[8] There are also other methods for assigning priorities to items.

4 Implementation

The bottom-up Earley deduction algorithm described here has been implemented in Quintus Prolog as part of the GeLD system. GeLD (Generalized Linguistic Deduction) is an extension of Prolog which provides typed feature descriptions and preference values as additions to the expressivity of the language, and partial evaluation, top-down, head-driven, and bottom-up Earley deduction as processing strategies. Tests of the system with small grammars have shown promising results, and a medium-scale HPSG for German is presently being implemented in GeLD. The lookup relation and the choice of an indexing scheme must be specified by the user of the system.

5 Conclusion and Future Work

We have proposed bottom-up Earley deduction as a useful alternative to the top-down methods which require subsumption checking and restriction to avoid prediction loops.

The proposed method should be improved in two directions. The first is that the lookup predicate should not have to be specified by the user, but automatically inferred from the program.

The second problem is that all non-unit clauses of the program are added to the chart. The addition of non-unit clauses should be made dependent on the goal and the base cases in order to go from a purely bottom-up algorithm to a directed algorithm that combines the advantages of top-down and bottom-up processing. It has been repeatedly noted [8, 17, 1] that directed methods are more efficient than pure top-down or bottom-up methods. However, it is not clear how well the directed methods are applicable to grammars which do not depend on concatenation and have no unique 'left corner' which should be connected to the start symbol.

It remains to be seen how bottom-up Earley deduction compares with (and can be combined with) the improved top-down Earley deduction of Dörre [3], Johnson [7] and Neumann [9], and to head-driven methods with well-formed substring tables [1], and which methods are best suited for which kinds of problems (e.g. parsing, generation, noisy input, incremental processing etc.).

References

[1] Gosse Bouma and Gertjan van Noord. Head-driven parsing for lexicalist grammars: Experimental results. In *EACL93*, pages 71 – 80, Utrecht, NL, 1993.

[2] Chris Brew. Adding preferences to CUF. In Jochen Dörre, editor, *DYANA-2 Deliverable R1.2.A: Computational Aspects of Constraint-Based Linguistic Description I*, pages 57 – 69. Esprit Basic Research Project 6852, 1993.

[3] Jochen Dörre. Generalizing Earley deduction for constraint-based grammars. In Jochen Dörre, editor, *DYANA-2 Deliverable R1.2.A: Computational Aspects of Constraint-Based Linguistic Description I*, pages 23 – 41. Esprit Basic Research Project 6852, 1993.

[4] Gregor Erbach. Using preference values in typed feature structures to exploit non-absolute constraints for disambiguation. In Harald Trost, editor, *Feature Formalisms and Linguistic Ambiguity*. Ellis-Horwood, 1993.

[5] Gregor Erbach. Towards a theory of degrees of grammaticality. In Carlos Martín-Vide, editor, *Current Issues in Mathematical Linguistics*. North-Holland, Amsterdam, to appear. Also published as CLAUS Report 34, Universität des Saarlandes, 1993.

[6] Dale Douglas Gerdemann. *Parsing and Generation of Unification Grammars*. PhD thesis, University of Illinois at Urbana-Champaign, 1991. Cognitive Science technical report CS-91-06 (Language Series).

[7] Mark Johnson. Memoization in constraint logic programming. Department of Cognitive Science, Brown University. Presented at the 1st International Conference on Constraint Programming, Newport, Rhode Island; to appear in the proceedings, 1993.

[8] Martin Kay. Algorithm schemata and data structures in syntactic processing. Technical Report CSL-80-12, XEROX PARC, Palo Alto, CA, 1980.

[9] Günter Neumann. *A Uniform Tabular Algorithm for Natural Language Parsing and Generation and its Use within Performance-based Methods*. PhD thesis, University Saarbrücken. forthcoming.

[10] Fernando C.N. Pereira and David H.D. Warren. Parsing as deduction. In *ACL Proceedings, 21st Annual Meeting*, pages 137–144, 1983.

[11] Carl Pollard. *Generalized Context-Free Grammars, Head Grammars, and Natural Language*. PhD thesis, Stanford, 1984.

[12] Mike Reape. A theory of word order and discontinuous constituency in West Germanic. In E. Engdahl and M. Reape, editors, *Parametric Variation in Germanic and Romance: Preliminary Investigations*, pages 25–40. ESPRIT Basic Research Action 3175 DYANA, Deliverable R1.1.A, 1990.

[13] Stuart M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

[14] Gertjan van Noord. *Reversibility in Natural Language Processing*. PhD thesis, Rijksuniversiteit Utrecht, NL, 1993.

[15] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting*, pages 104–111, Stanford, CA, 1987. Association for Computational Linguistics.

[16] David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1988.

[17] Mats Wirén. A comparison of rule-invocation strategies in context-free chart parsing. In *ACL Proceedings, Third European Conference*, pages 226–235, 1987.