Appears in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 748-753, Seattle, WA, July, 1994

Inducing Deterministic Prolog Parsers from Treebanks: A Machine Learning Approach *

John M. Zelle and Raymond J. Mooney
Department of Computer Sciences
University of Texas
Austin, TX 78712
zelle@cs.utexas.edu, mooney@cs.utexas.edu

Abstract

This paper presents a method for constructing deterministic Prolog parsers from corpora of parsed sentences. Our approach uses recent machine learning methods for inducing Prolog rules from examples (inductive logic programming). We discuss several advantages of this method compared to recent statistical methods and present results on learning complete parsers from portions of the ATIS corpus.

Introduction

Recent approaches to constructing robust parsers from corpora primarily use statistical and probabilistic methods such as stochastic context-free grammars (Black et al., 1992; Pereira and Schabes, 1992). Although several current methods learn some symbolic structures such as decision trees (Black et al., 1993) and transformations (Brill, 1993), statistical methods still dominate. In this paper, we present a method that uses recent techniques in machine learning to construct symbolic, deterministic parsers from parsed corpora (treebanks). Specifically, our approach is implemented in a program called CHILL (Zelle and Mooney, 1993b) that uses *inductive logic programming* (ILP) (Muggleton, 1992) to learn heuristic rules for controlling a deterministic shift-reduce parser written in Prolog.

We believe our approach offers several potential advantages compared to current methods. First, it constructs deterministic shift-reduce parsers, which are very powerful and efficient (Tomita, 1986) and arguably more cognitively plausible (Marcus, 1980; Berwick, 1985). Second, it constructs complete parsers from scratch that produce full parse trees, as opposed to producing only bracketings (Pereira and Schabes, 1992; Brill, 1993) or requiring an existing, complex parser that over-generates (Black et al., 1992; Black et al., 1993). Third, the approach is more flexible in several ways. It can produce parsers from either tagged or untagged treebanks.[1] When trained on an untagged corpus, it constructs it's own syntactic and/or semantic classes of words and phrases that allow it to deterministically parse the corpus. It can also learn to produce case-role assignments instead of syntactic parse trees and can use learned lexical and semantic classes to resolve ambiguities such as prepositional phrase attachment and lexical ambiguity (Zelle and Mooney, 1993b). Fourth, it uses a single, uniform parsing framework to perform all of these tasks and a single, general learning method that has also been used to induce a range of diverse logic programs from examples (Zelle and Mooney, 1994).

The remainder of the paper is organized as follows. In section 2, we summarize our ILP method for learning deterministic parsers, and how this method was tailored to work with existing treebanks. In section 3, we present and discuss experimental results on learning parsers from the ATIS corpus of the Penn Treebank (Marcus et al., 1993). Section 4 covers related work, and section 5 presents our conclusions.

The Chill System

Overview

Our system, CHILL, (Constructive Heuristics Induction for Language Learning) is an approach to parser acquisition which utilizes a general learning mechanism. The input to the system is a set of training instances consisting of sentences paired with the desired parses. The output is a shift-reduce parser (in Prolog) which maps sentences into parse trees.

The CHILL algorithm consists of two distinct tasks. First, the training instances are used to formulate an overly-general shift-reduce parser that is capable of producing parses from sentences. The initial parser is overly-general in that it produces a great many spurious analyses for any given input sentence. The parser is then specialized by introducing search-control heuristics. These control heuristics limit the contexts in

*This research was partially supported by the NSF under grant IRI-9102926 and the Texas Advanced Research Program under grant 003658114.

[1] In an *untagged treebank*, parses are represented as phrase-level word groupings without lexical categories dominating the words (e.g. parsed text from Penn Treebank (Marcus et al., 1993))

which certain operations are performed, eliminating the spurious analyses.

Constructing the Overly-General Parser

The syntactic parse of a sentence is a labeled bracketing of the words in the sentence. For example, the noun phrase, "a trip to Dallas", might be decomposed into component noun and prepositional phrases as: $[_{np}[_{np}$a trip] $[_{pp}$to $[_{np}$ dallas]]]. We represent such an analysis as a Prolog term of the form: `np:[np:[a, trip], pp:[to, np:[dallas]]]`.

A shift-reduce parser to produce such analyses is easily implemented as a logic program. The state of the parse is reflected by the contents of the stack and input buffer. A new state is produced by either removing a single word from the buffer and pushing it onto the stack (a shift operation), or by popping the top one or two stack elements and combining them into a new constituent which is then pushed back onto the stack (a reduce operation).

Each operation can be represented by a single program clause with two arguments representing the current stack and input buffer, and two arguments to represent their state after applying the operator. For example, the operations and associated clauses required to parse the above example phrase are as follows (the notation, **reduce(*N*)** *Cat*, indicates that the top N stack elements are combined to form a constituent with label, *Cat*):

reduce(2) pp:
```
    op([S1,S2|Ss], Words, [pp:[S1,S2]|Ss], Words).
```
reduce(2) np:
```
    op([S1,S2|Ss], Words, [np:[S1,S2]|Ss], Words).
```
reduce(1) np:
```
    op([S1|Ss], Words, [np:[S1]|Ss], Words).
```
shift: `op(Stack, [Word|Words], [Word|Stack], Words).`

Building an overly-general parser from a set of training examples is accomplished by constructing clauses for the `op` predicate. Each clause is a direct translation of a required parsing action; there must be a reduce operation for each constituent structure as well as the general shift operator illustrated above. If the sentence analyses include empty categories (detectable as lexical tokens that appear in the analyses, but not in the sentence), each empty marker is introduced via its own shift operator which does not consume a word from the input buffer.

The first step in the CHILL system is to analyze the training examples to produce the set of general operators that will be used in the overly–general parser. Once the necessary operators have been inferred, they are ordered according to their frequency of occurrence in the training set.

Parser Specialization

The overly-general parser produces a great many spurious analyses for the training sentences because there are no conditions specifying when it is appropriate to use the various operators. The program must be specialized by including control heuristics that guide the application of operator clauses. This section outlines the basic approach used in CHILL. More detail on incorporating clause selection information in Prolog programs can be found in (Zelle and Mooney, 1993a).

Program specialization occurs in three phases. First, the training examples are analyzed to construct positive and negative *control examples* for each operator clause. Examples of correct operator applications are generated by finding the first correct parsing of each training pair with the overly-general parser; any subgoal to which an operator is applied in a successful parse becomes a positive control example for that operator. A positive control example for any operator is considered a negative example for all previous operators that do not have it as a positive example. Note that this assumes a deterministic framework in which each sentence will have a single preferred parsing. Once an operator is found to be applicable to a particular parser state, subsequent operators will not be tried. For example, in parsing the above phrase, when the **reduce(2) NP** operator is first applied, the call to `op` appears as: `op([trip,a],[to,dallas], A, B)` where `A` and `B` are as yet uninstantiated output variables. This subgoal would be stored as a positive control example for the **reduce(2) NP** operator, and as a negative control example for **reduce(2) PP**, assuming the order of operator clauses shown above.

In the second phase, a general first-order induction algorithm is employed to learn a *control rule* for each operator. This control rule comprises a Horn-clause definition that covers the positive control examples for the operator but not the negative. There is a growing body of research in inductive logic programming which addresses this problem. CHILL combines elements from bottom-up techniques found in systems such as CIGOL (Muggleton and Buntine, 1988) and GOLEM (Muggleton and Feng, 1992) and top-down methods from systems like FOIL (Quinlan, 1990), and is able to invent new predicates in a manner analogous to CHAMP (Kijsirikul et al., 1992). Details of the CHILL induction algorithm can be found in (Zelle and Mooney, 1993b; Zelle and Mooney, 1994).

The final step in program specialization is to "fold" the control information back into the overly-general parser. A control rule is easily incorporated into the overly-general program by unifying the head of an operator clause with the head of the control rule for the clause and adding the induced conditions to the clause body. The definitions of any invented predicates are simply appended to the program. As an example, the **reduce(2) pp** clause might be modified as:

```
op([np:A,B|Ss], Words, [pp:[np:A,B]|Ss],Words) :-
    preposition(B).
preposition(of). preposition(to). ...
```

Here, the induction algorithm invented a new predicate representing the category "preposition."[2] This new predicate has been incorporated to form the rule which may be roughly interpreted as stating: "If the stack contains an NP followed by a preposition, then reduce this pair to a PP." The actual control rule learned for this operator is more complex, but this simple example illustrates the basic process.

Parsing the Treebank

Training a program to do accurate parsing requires large corpora of parsed text for training. Fortunately, such treebanks are being compiled and becoming available. For the current experiments, we have used parsed text from a preliminary version of the Penn Treebank (Marcus et al., 1993). One complication in using this data is that sentences are parsed only to the "phrase level", leaving the internal structure of NPs unanalyzed and allowing arbitrary-arity constituents. Rather than forcing the parser to learn reductions for arbitrary length constituents, CHILL was restricted to learning binary-branching structures. This simplifies the parser and allows for a more direct comparison to previous bracketing experiments (e.g. (Brill, 1993; Pereira and Schabes, 1992)) which use binary bracketings.

Making the treebank analyses compatible with the binary parser required "completion" of the parses into binary-branching structures. This "binarization" was accomplished automatically by introducing special internal nodes in a right-linear fashion. For example, the noun-phrase, `np:[the,big,orange,cat]`, would be binarized to create: `np:[the,int(np):[big, int(np):[orange, cat]]]`. The special labeling (int(np) for noun phrases, int(s) for sentences, etc.) permits restoration of the original structure by merging internal nodes. Using this technique, the resulting parses can be compared directly with treebank parses. All of the experiments reported below were done with automatically binarized training examples; control rules for the artificial internal nodes were learned in exactly the same way as for the original constituents.

Experimental Results

The Data

The purpose of our experiments was to investigate whether the mechanisms in CHILL are sufficiently robust for application to real-world parsing problems. There are two facets to this question, the first is whether the parsers learned by CHILL generalize well to new text. An additional issue is whether the induction mechanism can handle the large numbers of examples that would be necessary to achieve adequate performance on relatively large corpora.

We selected as our test corpus a portion of the ATIS dataset from a preliminary version of the Penn Treebank (specifically, the sentences in the file `ti_tb`). We chose this particular data because it represents realistic input from human-computer interaction, and because it has been used in a number of other studies on automated grammar acquisition (Brill, 1993; Pereira and Schabes, 1992) that can serve as a basis for comparison to CHILL.

Experiments were actually carried out on four different variations of the corpus. A subset of the corpus comprising sentences of length less than 13 words was used to form a more tractable corpus for systematic evaluation and to test the effect of sentence length on performance. The entire corpus contained 729 sentences with an average length of 10.3 words. The restricted set contains 536 sentences averaging 7.9 words in length. A second dimension of variation is the form of the input sentences and analyses. Since CHILL has the ability to create its own categories, it can use untagged parse trees. In order to test the advantage gained by tagging, we also ran experiments using lexical tags instead of words on both the full and restricted corpus.

Experimental Method

Training and testing followed the standard paradigm of first choosing a random set of test examples and then creating parsers using increasingly larger subsets of the remaining examples. The performance of these parsers was then determined by parsing the test examples. Obviously, the most stringent measure of accuracy is the proportion of test sentences for which the produced parse tree exactly matches the treebanked parse for the sentence. Sometimes, however, a parse can be useful even if it is not perfectly accurate; the treebank itself is not entirely consistent in the handling of various structures.

To better gauge the partial accuracy of the parser, we adopted a procedure for returning and scoring partial parses. If the parser runs into a "dead-end" while parsing a test sentence, the contents of the stack at the time of impasse is returned as a single, flat constituent labeled S. Since the parsing operators are ordered and the shift operator is invariably the most frequently used operator in the training set, shift serves as a sort of default when no reduction action applies. Therefore, at the time of impasse, all of the words of the sentence will be on the stack, and partial constituents will have been built. The contents of stack reflect the partial progress of the parser in finding constituents.

Partial scoring of trees is computed by determining the extent of overlap between the computed parse and the correct parse as recorded in the treebank. Two constituents are said to match if they span exactly the same words in the sentence. If constituents match and have the same label, then they are identical. The overlap between the computed parse and the correct parse is computed by trying to match each constituent of the computed parse with some

[2] Invented predicates actually have system generated names. They are renamed here for clarity.

Size	Correct	Partial	0-Cross	Crossing %
50	14.1	66.0	47.6	85.4
100	18.6	69.3	52.7	84.5
150	25.4	72.5	50.8	86.1
200	26.8	74.9	57.6	88.2
250	29.2	76.1	62.1	89.6
300	33.1	79.1	63.6	91.2

Table 1a: Lexical Tags

Size	Correct	Partial	0-Cross	Crossing %
50	11.4	60.0	43.1	80.3
100	10.3	61.5	45.2	81.5
150	12.4	62.0	44.2	80.4
200	17.5	67.8	52.1	83.5
250	18.1	67.7	51.6	83.7
300	17.4	69.6	53.6	85.2

Table 1b: Raw Text

Table 1: Results for restricted length corpus

constituent in the correct parse. If an identical constituent is found, the score is 1.0, a matching constituent with an incorrect label scores 0.5. The sum of the scores for all constituents is the overlap score for the parse. The accuracy of the parse is then computed as $Accuracy = (\frac{O}{Found} + \frac{O}{Correct})/2$ where O is the overlap score, $Found$ is the number of constituents in the computed parse, and $Correct$ is the number of constituents in the correct tree. The result is an average of the proportion of the computed parse that is correct and the proportion of the correct parse that was actually found.

Another accuracy measure, which has been used in evaluating systems that bracket the input sentence into unlabeled constituents, is the proportion of constituents in the parse that do not cross any constituent boundaries in the correct tree (Black, 1991). Of course, this measure only allows for direct comparison of systems that generate binary-branching parse trees.[3] By binarizing the output of the parser in a manner analogous to that described above, we can compute the number of sentences with parses containing no crossing constituents, as well as the proportion of constituents which are non-crossing over all test sentences. This gives a basis of comparison with previous bracketing results, although it should be emphasized that CHILL is designed for the harder task of actually producing labeled parses, and is not directly optimized for the bracketing task.

Results

The results of these experiments are summarized in Tables 1 and 2. The figures for the restricted length corpus in Table 1 reflect averages of three trials, while the results on the full corpus are averaged over two trials. The first column shows the size of the training set from which the parsers were derived, while the remaining columns present results for each of the four metrics outlined above. *Correct* is the percentage of test sentences with parses that matched the treebanked parse exactly. *Partial* is partial correctness using the overlap metric. The remaining columns reflect measures based on re-binarizing the parser output. *0-Cross* is the proportion of test sentences having no constituents that cross constituents in the correct parsing. The remaining column reports the percentage of (binarized) constituents that are consistent with the treebank (i.e. cross no constituents in the correct parse).

The results for the restricted corpus in Table 1 are encouraging. While we know of no other results for parsing accuracy of automatically constructed parsers on this corpus, the figures of 33% completely correct using the tagged input and 17% on the raw text seem quite good for a relatively modest training set of 300 sentences. The figures for 0-cross and crossing% are about the same as those reported in studies of automated bracketing for the unrestricted ATIS corpus (Brill (1993) reports 60% and 91.12%, respectively). However, our bracketing results for the unrestricted corpus are not as good.

A comparison of Tables 1a and 1b show that considerable advantage is gained by using word-class tags, rather than the actual words. This is to be expected as tagging significantly reduces the variety in the input. The results for raw-text use no special mechanism for handling previously unseen words occurring in the testing examples. Achieving 70% (partial) accuracy under these conditions seems quite good. Statistical approaches relying on n-grams or probabilistic context-free grammars would have difficulty due to the large number of terminal symbols (around 400) appearing in the modest-sized training corpus. The data for lexical selection would be too sparse to adequately train the pre-defined models. Likewise, the transformational approach of (Brill, 1993) is limited to bracketing strings of lexical classes, not words. A major advantage of our approach is the ability of the learning mechanism to automatically construct and attend to just those features of the input that are most useful in guiding parsing.

It should also be noted that the system created new categories in both situations. In the raw text experiments, CHILL regularly created categories for `preposition`, `verb`, `form-of-to-be`, etc.. With tagged input, various tags were grouped into classes such as the verb and noun forms. In both cases, the system also formed numerous categories and relations that seemed to defy any simple linguistic explanation. Nevertheless, these categories were helpful in parsing of new text. These results support our view that any

[3] A tree containing a single, flat constituent covering the entire sentence always produces a perfect (non)crossing score.

Size	Correct	Partial	0-Cross	Crossing %
50	9.2	58.9	35.3	72.5
100	16.3	65.2	41.9	74.3
150	21.0	70.1	44.7	76.8
200	23.5	71.4	46.2	77.9
250	25.6	72.5	47.1	77.9

Table 2a: Lexical Tags

Size	Correct	Partial	0-Cross	Crossing %
50	6.2	54.0	33.1	68.9
100	4.7	54.9	41.2	72.0
150	8.5	59.6	39.7	71.5

Table 2b: Raw Text

Table 2: Results for full corpus

practical acquisition system should be able to create its own categories, as it is unlikely that independently-crafted feature systems will capture all of the nuances necessary to do accurate parsing in a reasonably complex domain.

Table 2 shows results for the full corpus. As one might expect, the results are not as good as for the restricted set. There are a number of factors that could lead to diminishing performance as a function of increasing sentence length. One explanation might be that the longer sentences are simply more complicated and, thus harder to parse accurately. If the difficulty is inherent in the sentences, the only solution is larger training sets.

Another possible problem is the compounding of errors. If an operator is chosen incorrectly early in the parse, it might lead the parser into states that have not been encountered in training, leading to subsequent errors in the application of other operators. This factor might be mitigated by developing more robust training procedures. By providing control examples from erroneous states as well as correct ones, the parser might be trained to be somewhat self-correcting, choosing correct operators later on even in the face of previous errors.

A third possibility is that the additional sentence length is "swamping" the induction algorithm. Increasing the average sentence length significantly increases the number of control examples that must be handled by the induction mechanism. Training sizes of several hundred sentences give rise to induction over thousands of control examples. Additionally, longer sentences lend themselves to more conflicting analyses and may increase the amount of *noise* in the control data making it more difficult to spot useful generalizations. Additional progress here would require further improvement in the efficiency and noise-handling capabilities of the induction algorithm.

Clearly further experimentation is needed to pin down where the most improvement can be made. The results so far do indicate that the approach has potential. The current Prolog implementation running on a SPARC 2 was able to induce parsers from several hundred sentences in a few hours, producing over 700 lines of Prolog code. One trial was run using 400 tagged sentences from the full corpus; the resulting parser achieved 29.4% absolute accuracy and a partial scoring of 82%. Further improvements in efficiency may make it feasible to produce parsers from thousands of training sentences.

Related Work

As mentioned above, most recent work on automatically constructing parsers from corpora has focused on acquiring stochastic grammars rather than symbolic parsers. When learning in this framework, "one simply gathers statistics" to set the parameters of a predefined model (Charniak, 1993). However, there is a long tradition of research in AI and Machine Learning suggesting the utility of techniques that extract underlying structural models from the data. Earlier work in learning symbolic parsers (Anderson, 1977; Berwick, 1985) used fairly weak learning methods specific to language acquisition and were not tested on real corpora. CHILL represents the first serious application of modern, machine-learning methods to acquiring parsers from corpora.

Brill (1993), presents a technique for acquiring parsers that produce binary-branching syntax trees with unlabeled nonterminals. The technique, utilizing structural transformation rules based on lexical category information, has proven quite successful on real corpora. CHILL, which creates fully labeled parses, has a more general learning mechanism allowing it to make distinctions based on more subtle structural and lexical cues (e.g. creating semantic word classes for resolving attachment).

Our framework for learning deterministic, context-dependent parsers is very similar to that of (Simmons and Yu, 1992); however, there are two advantages of our ILP method compared to their exemplar matching method. First, ILP methods can handle unbounded, structured data so that the context does not need to be fixed to a limited window of the stack and the remaining sentence. The entire stack and remaining sentence is available as potential context for deciding which parsing operator to apply at each step. Second, the system is capable of creating its own syntactic and semantic word and phrase classes instead of relying on the user to provide part-of-speech tagging.

CHILL's ability to invent new classes of words and phrases specifically for resolving ambiguities such as prepositional phrase attachment makes it particularly interesting. There is some existing work on learning lexical classes from corpora (Schütze, 1992); however, the classes are based on word co-occurrence rather than the specific needs of parsing. There are also methods for learning to resolve attachments using lexical information (Hindle and Rooth, 1993); however, they do not create new lexical classes. CHILL uses a single learning algorithm to perform both of these tasks.

Conclusion

This paper has demonstrated that modern machine-learning methods are capable of inducing traditional shift-reduce parsers from corpora, complementing the results of recent statistical methods. We believe that the primary strength of corpus-based methods is not the particular approach or type of parser employed (e.g. statistical, connectionist, or symbolic), but the fact that large amounts of real data are used to automatically construct complex parsers that are intractable to build manually. However, we believe our approach based on a very general inductive-logic-programming method has several advantages such as efficient, deterministic parsing; production of complete labeled parse trees; and an ability to use untagged text and automatically create new, useful lexical and phrasal categories based directly on the needs of parsing.

References

Anderson, J. R. (1977). Induction of augmented transition networks. *Cognitive Science*, 1:125–157.

Berwick, B. (1985). *The Acquisition of Syntactic Knowledge*. Cambridge, MA: MIT Press.

Black, E., Jelineck, F., Lafferty, J., Magerman, D., Mercer, R., and Roukos, S. (1993). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 31–37. Columbus, Ohio.

Black, E., Lafferty, J., and Roukaos, S. (1992). Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 185–192. Newark, Delaware.

Black, E. et. al. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*, pages 306–311.

Brill, E. (1993). Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 259–265. Columbus, Ohio.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press.

Hindle, D. and Rooth, M. (1993). Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120.

Kijsirikul, B., Numao, M., and Shimura, M. (1992). Discrimination-based constructive induction of logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 44–49. San Jose, CA.

Marcus, M. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge, MA: MIT Press.

Marcus, M., Santorini, B., and Marcinkiewicz, M. (1993). Building a large annotated corpus of english: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

Muggleton, S. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Ann Arbor, MI.

Muggleton, S. and Feng, C. (1992). Efficient induction of logic programs. In Muggleton, S., editor, *Inductive Logic Programming*, pages 281–297. New York: Academic Press.

Muggleton, S. H., editor (1992). *Inductive Logic Programming*. New York, NY: Academic Press.

Pereira, F. and Schabes, Y. (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 128–135. Newark, Delaware.

Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239–266.

Schütze, H. (1992). Context space. In *Working Notes, AAAI Fall Symposium Series*, pages 113–120. AAAI-Press.

Simmons, R. F. and Yu, Y. (1992). The acquisition and use of context dependent grammars for English. *Computational Linguistics*, 18(4):391–418.

Tomita, M. (1986). *Efficient Parsing for Natural Language*. Boston: Kluwer Academic Publishers.

Zelle, J. M. and Mooney, R. J. (1993a). Combining FOIL and EBG to speed-up logic programs. In *Proceedings of the Thirteenth International Joint conference on Artificial intelligence*, pages 1106–1111. Chambery, France.

Zelle, J. M. and Mooney, R. J. (1993b). Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 817–822. Washington, D.C.

Zelle, J. M. and Mooney, R. J. (1994). Combining top-down and bottom-up methods in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*. New Brunswick, NJ.