Analysis of Equation Structure using Least Cost Parsing

R. Nigel Horspool and John Aycock

E-mail: `{nigelh,aycock}@csr.uvic.ca`

Department of Computer Science, University of Victoria
P.O. Box 3055, Victoria, BC, Canada V8W 3P6

Abstract

Mathematical equations represented in markup languages such as LaTeX and MathML are difficult to parse because the particular mathematical notation may not be known in advance. Parsing would be, for example, the first step before translation to another word processing format. A promising approach is to specify the equation language with an ambiguous grammar whose rules have associated cost expressions. The parse which yields the least cost would be the preferred parse.

An implementation of least cost parsing, based on Earley's method, is described. If the cost expressions are restricted to functions that are monotonically non-decreasing with respect to each argument, the space of parsing possibilities may be pruned and the parsing method is efficient. If preprocessing of the grammar into a simplified form is permitted, the parser is as efficient as Earley's parsing method.

1 Introduction

LaTeX [5] is a text markup language for specifying the formatting of a document. It is widely used by mathematicians and scientists for preparation of published articles. For formatting of mathematical equations, LaTeX has few rivals. This is due to the variety of symbols and notations that are supported and to the degree of control offered over equation layout. The LaTeX tagging scheme for mathematics is, in the context of markup languages, known as a presentation tagging scheme. This is because the primary purpose of the tags is to specify how the equation components are to be formatted and not so much to specify what the equations mean. For example, the LaTeX input

```
    \[ a^2 \]
```

is used to produce the formatted result a^2. However, the '`^`' symbol must not be confused with an exponentiation operator. It is not an operator in a mathematical sense, it is simply a tag that specifies how to format the following group of symbols. The meaning of the '`^`' symbol as an operator to set text in superscript format perhaps becomes clearer if one observes that a nuclear physicist might write the LaTeX notation

```
    \[ ^3 He \]
```

to produce the formatted result ^3He.

Note that the '`\[`' and '`\]`' symbols delimit equations within LaTeX files. There are other delimiter pairs which also delimit equations. The analysis techniques described in this paper are meant to be applied only to the equations within LaTeX files, and these equations can be easily located and extracted.

XML (for Extensible Markup Language) [3] is a general notation for adding markup to text documents. Many special purpose tagging schemes have been defined under the XML aegis; one such scheme is MathML [6]. MathML makes a distinction between presentation tagging and its alternative, content tagging, by providing a set of tags for each. As an example of MathML, reference [6] shows how the equation

$$r \; = \; \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

would be specified using presentation tags, as shown in Figure 1, and using content tags, as shown in Figure 2. The main point to notice with Figure 1 is that a horizontal sequence of equation elements is laid out using the `<mrow>` tag, and that the sequence can be a free mixture of mathematical operators and operands. In contrast, the content tagging used in Figure 2 explicitly shows which operators apply to which operands, and the whole construct is tagged as a relation using the `<reln>` tag. As a comparison between LaTeX and MathML, the same equation can be rendered in LaTeX in the much more concise form

```
    \[    r ~ = ~ \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}   \]
```

but this is still presentation tagging.

An all too frequent requirement is to convert documents from one document formatting system to another. Authors may prepare their paper using one system but discover that the publisher uses another. A company may decide to standardize all its documentation in one word processing system only to find that it first has much legacy documentation to convert. The author has developed a conversion program (a filter) which translates LaTeX docu-

```
<!-- Presentation Tagging -->
<mrow>
    <mi>x</mi>
    <mo>=</mo>
    <mfrac>
        <mrow>
            <mo>-</mo> <mi>b</mi> <mo>&PlusMinus;</mo>
            <msqrt>
                <mrow>
                    <msup>
                        <mi>b</mi> <mn>2</mn>
                    </msup>
                    <mo>-</mo> <mn>4</mn> <mi>a</mi> <mi>c</mi>
                </mrow>
            </msqrt>
        </mrow>
        <mrow>
            <mn>2</mn> <mi>a</mi>
        </mrow>
    </mfrac>
</mrow>
```

Figure 1: Example of Presentation Tags

ments into FrameMaker™ format. (FrameMaker [4] is a desktop publishing system sold by Adobe Systems Corporation.) Much of the conversion is straightforward. However, conversion of mathematical equations is problematic because LaTeX uses presentation tags. FrameMaker requires precise specification of an equation's structure. Its internal format for an equation is a tree, where a tree node represents an operator and its children represent the operands. This tree form is equivalent to use of content markup tags.

Similarly, the equation editor provided with the ubiquitous Microsoft Word™ word processing system requires full structural knowledge of an equation. Conversion from LaTeX or from an equation tagged with MathML presentation tags to MS Word would therefore require that the equation be analyzed. Conversion from MathML presentation tags to MathML content tags would also be useful.

Analysis of an equation from an arbitrary document defined with presentation tags is difficult. A difficulty that may be encountered is that the equation mechanism has been used to describe a construct which is not an equation at all (such as the ^3He example). Ignoring that issue, we have the problem that different disciplines use different sets of symbols with different associated meanings. Elaborating on that problem, we present the following examples.

- The circle symbol `\circ` might be used in LaTeX as

  `\[ A^\circ \]` or `\[ f \circ g \]`

  to yield the results A° and $f \circ g$. In the former case, it appears as an operand; in the latter as an operator.

- We cannot *a priori* know which symbols are used as parentheses and which as operators. Consider the different uses of the '<' symbol in the constructs $<\theta|$ and $x<0$. (The first example is from quantum physics.)

- In some notational systems, the parentheses do not necessarily balance.

- Even if we could recognize some symbols as operators, we may have to infer their precedence and associativity from the way the equation is spaced. For example, if we have an expression which is formatted as

  $$a - b \quad \times \quad c + d$$

  when printed, we would probably want to associate the operators with operands in the same manner as

  $$(a\text{-}b) \times (c{+}d)$$

```
<!-- Content Tagging -->
<reln>
    <eq/>
    <ci>r</ci>
    <apply>
        <over/>
        <apply>
            <fn occurrence="infix"><mo>&PlusMinus;</mo></fn>
            <apply>
                <minus/> <ci>b</ci>
            </apply>
            <apply>
                <root/>
                <apply>
                    <minus/>
                    <apply>
                        <power/> <ci>b</ci> <cn>2</cn>
                    </apply>
                    <apply>
                        <times/> <cn>4</cn> <ci>a</ci> <ci>c</ci>
                    </apply>
                </apply>
                <cn>2</cn>
            </apply>
        </apply>
        <apply> <times/> <cn>2</cn> <ci>a</ci> </apply>
    </apply>
</reln>
```

Figure 2: Example of Content Tags

2 A Grammar for LaTeX Equations

Following on from the issues raised in the previous section, we show how they can be solved in a grammar for LaTeX equations. Since the full grammar would be very large, we illustrate only some of the more interesting features, and we use them to motivate our form of cost expressions. We give two grammars, G1 and G2. The first one is to introduce the idea, while the second one is a full demonstration.

2.1 Handling Unbalanced Parentheses

Grammar G1 pairs up parentheses, using the first rule, when possible. (This is not our preferred solution, which is incorporated into Grammar G2.)

```
E      →     ( E )           %cost{0}
E      →     ( E             %cost{10}          Grammar G1
E      →     E )             %cost{10}
E      →     symbol          %cost{0}
```

The cost formulae used with these rules are expressions which happen to be constant. Three parses of the sentence “`((a)`” would proceed as follows. (Two additional parses which are similar to #3, but that reduce parentheses in other orders, are omitted.) The handle for each reduction is shown in bold, costs are shown underneath nonterminals.

$$\textit{\#1.}\quad (\;(\;\mathbf{a}\;) \;\Rightarrow\; (\;\mathbf{(}\;\underset{0}{\mathbf{E}}\;) \;\Rightarrow\; \mathbf{(}\;\underset{10}{\mathbf{E}}\;\mathbf{)} \;\Rightarrow\; \underset{0}{E}$$

$$\textit{\#2.}\quad (\;(\;\mathbf{a}\;) \;\Rightarrow\; (\;\mathbf{(}\;\underset{0}{\mathbf{E}}\;\mathbf{)} \;\Rightarrow\; \mathbf{(}\;\underset{0}{\mathbf{E}} \;\Rightarrow\; \underset{10}{E}$$

$$\textit{\#3.}\quad (\;(\;\mathbf{a}\;) \;\Rightarrow\; (\;(\;\underset{0}{\mathbf{E}}\;\mathbf{)} \;\Rightarrow\; (\;\mathbf{(}\;\underset{10}{\mathbf{E}} \;\Rightarrow\; \mathbf{(}\;\underset{10}{\mathbf{E}} \;\Rightarrow\; \underset{10}{E}$$

The least cost parse is the one labelled #1. It has a final cost of zero, because that is the value of the cost expression which generates the grammar’s goal symbol in the final reduction step.

2.2 Using White Space to Determine Operator Precedence

There are several notations in LaTeX equations for adding differing amounts of white space between symbols. For simplicity, we use only the symbol ‘~’ here; each occurrence forces a space to occur in the equation.

A desirable property would be for standard operator symbols (such as +, -, ×, /) to have their normal precedences when used without added spacing, but to allow inserted spaces to modify their precedences. It would be possible to devise a grammar which enumerates the operators with varying numbers of space symbols on each side, and associates different precedences with these operator/space combinations. However, the grammar would rapidly become unwieldy, especially considering that LaTeX provides six different spacing symbols.

Our least-cost grammar approach provides a simple solution. Grammar G2 is an example for expressions that may contain addition and multiplication operators, as well as (possibly unbalanced) parentheses:

```
S   →  E                      %cost{$1}

E   →  E OP E                 %cost{$2*($1 + 1.5*$3)}
E   →  ( E )                  %cost{$2}
E   →  ( E                    %cost{$2*2.0}                Grammar G2
E   →  E )                    %cost{$1*2.0}
E   →  symbol                 %cost{1.0}

OP  →  ~ OP                   %cost{0.7*$2}
OP  →  OP ~                   %cost{0.7*$1}
OP  →  +                      %cost{80.0}
OP  →  *                      %cost{100.0}
```

The construct *$n* represents the evaluated cost of the *n*-th element on the right-hand side of a grammar rule. In our implementation, the construct is meaningful only if this *n*-th symbol is a non-terminal. However, an enhanced implementation where the lexical analyzer supplies a cost for each terminal symbol is easy to imagine.

Using Grammar G2, the sentence `a+b*c` has two different parses with two different cost calculations. One parse (where '+' has a higher precedence than '*') is as follows:

```
a + b * c  ⇒  E + b * c  ⇒  E OP a * c  ⇒  E OP E * c  ⇒
              1             1  80           1  80  1

      E * c  ⇒  E    OP   c  ⇒  E    OP   E  ⇒  E
      200       200  100        200  100  1     20150
```

The other parse which corresponds to '+' having a lower precedence than '*' achieves a final cost of 20080, and would therefore be the parse selected by our method.

On the other hand, if the sentence to be parsed is `a+b~*~c` then the two distinct parses yield costs of 9873.5 and

14780. The parse which gives '`+`' a higher precedence than '`~*~`' is the one with the lower cost, and is therefore selected.

The forms of the cost expressions used in the example may appear to be mysterious. They are, however, easy to explain. First, the rules `OP` $\rightarrow$ `+` and `OP` $\rightarrow$ `*` have costs which correspond to relative precedence values. The rules which introduce the spacing symbol '`~`' cause a lower cost or lower precedence value to be computed, as intended. Now consider the rule which introduces an infix operator, namely `E` $\rightarrow$ `E OP E`. A cost expression of the form \$2×(\$1+\$3) encourages the lowest precedence operator to be applied last. We actually use a variation on that expression, namely \$2×(\$1 + 1.5*\$3). The 1.5 coefficient biases the parse towards left associativity of the operators, as is conventional for most operators in mathematics, by making it more expensive to build up composite right operands.

2.3 Uncertain Operators and Parentheses

When we are unsure as to which symbols are operators or parentheses, we can provide multiple uses for the symbols in the grammar. If, in conventional mathematics, a particular symbol such as '`+`' is used as an operator, we can supply a low cost grammar rule which represents that use. A high cost grammar rule which allows the symbol to be used as an operand is also supplied. For example, adding the rule

```
E  →  +      %cost{10000}
```

to Grammar G2 would permit sentences such as `+*+` to have a parse, albeit one with a very high cost.

Similar principles apply if we want to encourage a symbol such as '`<`' to be an operator, yet permit it to be used with '`>`', as a bracketing symbol.

3 Implementation of a Least Cost Parser

Our implementation of least cost parsing was inspired by Stolcke's probabilistic parsing algorithm [7], in particular the variant that discovers the Viterbi (or most probable) parse. However, our implementation is intrinsically simpler. Stolcke shows how Earley's general parsing method [2] can be extended to deal with probabilistic grammars, as do we.

3.1 Extending Earley's Method

Earley's parsing algorithm constructs a sequence of states, one state for each terminal symbol processed. Each state is implemented as a set of items, where an item represents a grammar rule that has been partially matched. The item also records how much of the rule's right-hand side has been matched, and it contains a reference back to the state where matching this rule commenced. Thus an Earley parser (without using lookahead) uses items that are conventionally depicted as thus:

$$[\ A \rightarrow \alpha \bullet \beta \,;\ @n\ ]$$

where $A \rightarrow \alpha\beta$ is a grammar rule being matched, the position of the dot indicates that α has been matched (and β is unmatched), and *n* is the number of the state where the parser started matching the rule $A \rightarrow \alpha\beta$.

We augment items in the Earley parser with two extra components, as follows:

$$[\ A \rightarrow \alpha \bullet \beta \,;\ @n \,;\ \overline{\mathbf{C}} \,;\ @r\ ]$$

The $\overline{\mathbf{C}}$ component is a vector of costs, one per symbol in α. The *r* component is called the *least cost predecessor.* It records the identity of a predecessor item *in the same state* which generated this particular item. The modified Earley parsing algorithm operates as follows.

Scanner Step. If the input symbol is a then for each item of the form

$$[\ A \rightarrow \alpha \bullet a\ \beta \,;\quad @n \,;\quad \overline{\mathbf{C}} \,;\quad @r\ ]$$

in the previous state, a new item

$$[\ A \rightarrow \alpha\ a \bullet \beta \,;\quad @n \,;\quad \overline{\mathbf{C}} \parallel 0 \,;\quad -\ ]$$

is added to the current state. The notation || represents concatenation, i.e. $\overline{\mathbf{C}} \parallel 0$ is a new vector with the same elements as $\overline{\mathbf{C}}$ but with 0 appended as an extra element at the end. The 'least cost predecessor' component is not used.

Completer-Predictor Step. The following item *i* in the current state

$$[\ A \rightarrow \alpha \bullet \,;\quad @n \,;\quad \overline{\mathbf{D}} \,;\quad @r\ ]$$

causes the cost calculation to be performed for the rule $A \rightarrow \alpha$ using input costs taken from $\overline{\mathbf{D}}$ (where *$k* represents the *k*-th element of $\overline{\mathbf{D}}$) and yielding a cost c_A. Then, for each item in state *n* with the form

$$[\ B \rightarrow \gamma \bullet A\ \delta \,;\quad @m \,;\quad \overline{\mathbf{C}} \,;\quad -\ ]$$

we consider adding the following new item to the current state.

$$[\ B \rightarrow \gamma\ A \bullet \delta;\quad @m\ ;\quad \overline{\mathbf{C}} \parallel c_A\ ;\quad @i\ ]$$

However, such an addition depends on whether a similar item already exists in the current state. The possibilities are handled as follows. If an item

$$[\ B \rightarrow \gamma\ A \bullet \delta;\quad @m\ ;\quad \overline{\mathbf{K}}\ ;\quad @j\ ]$$

does not exist in the current state, then the new item is added. If it does exist, then a comparison is made between the two cost vectors. There are three possible outcomes from an element-wise vector comparison:

1. $\overline{\mathbf{C}} \parallel c_A$ is less than $\overline{\mathbf{K}}$, then the new item replaces the existing item in the current state.
2. $\overline{\mathbf{C}} \parallel c_A$ is greater than or equal to $\overline{\mathbf{K}}$, then the new item is not added.
3. $\overline{\mathbf{C}} \parallel c_A$ is incomparable to $\overline{\mathbf{K}}$ (some elements are smaller, some are larger) then the new item is added to the current state, so that both items are present afterwards.

Our scheme implements a pruning strategy that tries to prevent the number of items in a state from exploding exponentially. It requires that every cost expression must be non-negative and a non-decreasing function of its arguments. More formally, if *f* is a cost expression with *k* arguments x_1, x_2, ... x_k, then we require that

$$\frac{\partial f}{\partial x_i} \geq 0 \qquad \text{for } 1 \leq i \leq k$$

For example, the pruning operation for outcome 1 above (deleting the existing item in the current state) assumes that no matter what values are later appended to the $\overline{\mathbf{C}} \parallel c_A$ cost vector, the cost evaluation must yield a smaller result than if the cost vector were $\overline{\mathbf{K}}$ with the same values appended. Our implementation supports cost expressions which use only addition and multiplication operators. Simple expressions that contain only addition and multiplication operations can be automatically tested to verify that they are monotonically non-decreasing with respect to each argument.

Because of outcome 3, above, it is possible that a state can contain two items which are identical except for their cost vectors (and least cost predecessor references). This means that we need to revise our statement of the first part of the completer-predictor step to prune more items from the current state. The modification is that if two or more items in the current state with the form

$$[\ A \rightarrow \alpha \bullet\ ;\quad @n\ ;\quad ??\ ;\quad ??\ ]$$

have the same production rule $A \to \alpha$ and the same parent state reference @n (and where ?? indicates a *don't care* value), then we evaluate the cost for all such items and use only the one with the least cost to generate new items in the current state.

4 Discussion and Conclusions

4.1 Parsing Efficency

The least cost parsing method described in this paper is not as efficient as Earley's method. When, in the modified completer-predictor step, an element-wise comparison of cost vectors is performed, it is possible to obtain the result 'incomparable'. This causes multiple occurrences of an item to appear in a state, whereas Earley's method would have had only one occurrence. If each such multiple occurrence can give rise to multiple occurrences of successor items in this or another state, an exponential explosion can occur. Thus the worst-case time complexity and the worst-case space complexity of the parsing method are both potentially exponential.

However, rewriting the grammar so that each rule contains at most two non-terminal symbols would eliminate the possibility of vector comparisons yielding the result 'incomparable'. The drawback is that the cost formulae need to be transformed too. A formula of the form $f(a,b,c)$ would, for example, have to be transformable into $f_1(f_2(a,b),c)$, and this imposes a strong restriction on the general form of the cost expressions. Thus, we can claim the same time and space bounds as for Earley's algorithm if preprocessing the grammar is permitted and if the cost expressions are sufficiently simple.

4.2 Comparison with Other Work

Least cost parsing with cost functions is *not* equivalent to selecting the most probable parse from a probabilistic parser. For example, there is no assignment of probabilities to grammar rules which would produce results equivalent to Grammar G1. The cost expression for the first rule in that grammar has the effect of forgetting how expensive (or how improbable) the parse of the nested expression is. This is, in fact, a desirable possibility for some constructions when analyzing LaTeX equations. The equation notation permits grouping of symbols with brace characters. The braces are not printed, and they must match in pairs. The symbols within braces would, almost always, have to be matched as a subexpression.

For example, here is a LaTeX equation

```
    \[ \frac{a+b}{a^2 + b^2} \]
```

which should produce output similar to the following.

$$\frac{a+b}{a^2+b^2}$$

In this situation, we must accept some parse of the subexpressions enclosed by the braces and not let the costs of those parses influence the result at a higher level.

Our approach can be seen as a special case of GLR parsing with scoring [8]. This approach allows a general scoring function to be associated with the parse tree, and the tree with the best score will be the one that is selected. The scoring approach is too general for the problem being posed. An efficient implementation with pruning of the search space is feasible only if the scoring function is heavily restricted and if the input sentences are not large. Our least cost parsing method automatically has an implementation which is both space and time efficient.

4.3 Application to Error-Correcting Parsers

In [1], a technique was proposed for error correcting parsers which involved augmenting the grammar with error productions. The proposed implementation was a variant of an Earley parser where items are augmented with counts of how often error productions must be used to match the input. Items with low counts supercede items with larger counts. It would be an easy matter to use cost expressions with our parsing method to achieve the same effect.

4.4 Conclusions

Least-cost parsing with cost expressions is a parsing approach that has an efficient implementation. It is a method that is ideally suited to analyzing presentation markup of mathematical equations, and may have other applications too. The parsing approach has been implemented and it works.

Acknowledgements

Financial support from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

References

[1] A. V. Aho and T. G. Peterson. A Minimum-Distance Error-Correcting Parser for Context-Free Languages, SIAM Journal on Computing 1, 4 (1972), pp. 305-312.

[2] J. Earley. An Efficient Context-Free Parsing Algorithm. Communications of ACM 6, 8 (1970), pp. 451-455.

[3] Extensible Markup Language (XML) 1.0, 10 February 1998. URL: `http://www.w3.org/TR/REC-xml`.

[4] S. O'Keefe. *Framemaker 5.5.6 for Dummies*. IDG Books, 1999.

[5] L. Lamport. *LATEX: A Document Preparation System*. 2nd Edition. Addison-Wesley, 1994.

[6] Mathematical Markup Language (MathML™) 1.01 Specification, 7 July 1999. URL: `http://www.w3.org/TR/REC-MathML`.

[7] A. Stolcke. An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities. Computational Linguistics 21, 2 (1995), pp. 165-201.

[8] K.-Y. Su, J.-N. Wang, M.-H. Su and J.-S. Chang. GLR Parsing with Scoring **in** *Generalized LR Parsing*. M. Tomita, editor. Kluwer Academic, 1991, pp. 93-112.