Context and Caterpillars and Structured Documents*

Anne Brüggemann-Klein[1], Stefan Hermann[1] and Derick Wood[2]

[1] Institut für Informatik, Technische Universität München, Arcisstr. 21, 80290 München, Germany.
E-mail: {brueggem, hermann}@informatik.tu-muenchen.de.
WWW: http://www11.informatik.tu-muenchen.de.

[2] Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong.
E-mail: dwood@cs.ust.hk.
WWW: http://www.cs.ust.hk/~dwood.

Abstract. We present a novel, yet simple, technique for the specification of context in structured documents that we call caterpillar expressions. Although we are applying this technique in the specification of context-dependent style sheets for HTML, XML, and SGML documents, it is clear that it can be used in other environments such as query specification for structured documents and for computer program transformations.
In addition, we present a number of theoretical results that allow us to compare the expressive power of caterpillar expressions to that of tree automata.

1 Introduction

Context-dependent processing and specification are not new topics; they surface in almost all computing activities. What is somewhat surprising is that the issue of context as a topic in its own right does not appear to have been studied. In the Designer project, that we have been working on over a number of years, we were faced with the problem of the specification of context-dependent style rules. At first, we expected to use the traditional approach from the compiler and programming-languages community; namely, attribution [2, 12, 21]. In addition, we also expected to be able to use and modify previous approaches to style specification such as suggested by the DSSSL document [3] and Lie's work [14, 15] for SGML [10] and XML [8] style specification. Alternatively, we considered Munson's approach in the Proteus system [19] and Murata's more general approach [20] which uses a tree-automata-based approach. Murata's approach is the closest technique to ours, although more traditional and quite

* This paper will appear in the Principles of Digital Document Processing (PODDP 98) to be published in the Springer-Verlag Lecture Notes in Computer Science series. The work of the three authors was supported partially by a joint DAAD-HK grant. In addition, the work of the third author was supported under a grant from the Research Grants Council of Hong Kong.

different. But, we were faced with an additional constraint that changed our thinking. We wanted to provide a system for graphics designers to use to specify style rules and style sheets [5]. The point here is that they would be primarily computer naive. They would almost certainly find the manipulation of attributes difficult. Therefore, we decided to separate, somewhat, the specification of context from the issue of style specification. One observation about this separation is in order: we need only provide a mechanism that tests whether a specific context of a given part of a document is present or not. Based on this observation, the style rules may now incorporate conditional statements or expressions to express context-dependent choices. Thus, we can isolate context determination from style rule syntax to a large extent. The preliminary work on DESIGNER did just that [4, 7]. We have also investigated style-sheet specification for tables [23] and investigated some basic decidability questions for style sheets [6].

The contextual technique we introduce is also applicable to the compilation of computer programs, but is of less interest there as compiler designers and writers do not allow users to modify a compiler according to new context dependencies. We might use these techniques when developing code optimizers or other program transformation tools since, with both examples, there may be a number of individuals collaborating on the development [18].

Once we have isolated the specification of contexts from the more general specification of style sheets, we are able to provide naive users with better support for this aspect of style specification. Indeed, it also frees us to consider different techniques (different from attribution, for example) for context specification. Since regular expressions are understood by many people who are not programmers *per se,* and they are the simplest specification technique, we decided to use them for context specification. Mendelzon's research [9, 16] has shown, with his graph-theoretic query language G++ for databases, that this approach is not only powerful but also it can be visualized well [9]. (In Mendelzon's project, users provide restrictions of a graph-theoretic view of a database by graph-theoretic means. This visual process provides, essentially, a subgraph as the query.) Similar approaches have been used by Lauer *et al.* [13] to specify paths and processes for concurrent systems.

We make the well-accepted assumption that a set of similar documents are modeled by syntax trees or abstract syntax trees of a given grammar (an SGML DTD, an XML DTD or HTML) that generates the set of all such documents.

2 Caterpillars and context

It is natural to write of a context c of a node u; for example, if u is a paragraph, we may wish to determine whether u is the first paragraph of a chapter or of a section. In this setting, the property "is the first paragraph of a chapter or of a section" is the context. We choose to invert this intuitive notion by identifying all first paragraphs in chapters and sections of a document as the context "first paragraph." Thus, for a specific abstract syntax tree, the set of first-paragraph-in-a-chapter-or-in-a-section nodes are the context "first paragraph." In other words, for a given tree T, each set S of nodes of T may be a context in the sense that the nodes in S are all the nodes in T that, intuitively, have a specific context.

Since a real caterpillar crawls around a tree, we define a **contextual caterpillar** as a sequence of atomic movements and atomic tests. A caterpillar can move from the current node to its parent, to its left sibling, to its right sibling, to its first child, or to its last child. To prevent a caterpillar from dropping off a tree it is allowed to test whether it is at a leaf (external node), at the root, at the first sibling, or at the last sibling. Note that these navigational and testing operations define an ADT for trees; however, caterpillars are more than an ADT as they capture a specific sequence of ADT operations.

For example, given the partial document tree in Fig. 1, a first-paragraph-in-chapter caterpillar is:

$$\mathtt{p}, \mathtt{isfirst}, \mathtt{up}, \mathtt{up}, \mathtt{up}, \mathtt{ch},$$

where a node label is an implicit test on the current node, and a first-paragraph-in-section caterpillar is:

$$\mathtt{p}, \mathtt{isfirst}, \mathtt{up}, \mathtt{up}, \mathtt{up}, \mathtt{up}, \mathtt{sect}.$$

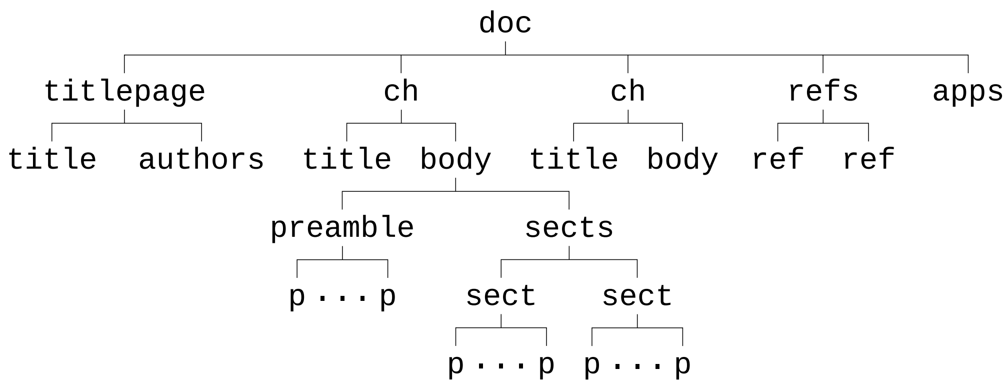

Fig. 1. An example document tree.

Similarly, a last-section caterpillar is:

$$\mathtt{sect}, \mathtt{islast}.$$

In each of these examples, if we place the caterpillar at any node in a given tree, it crawls and evaluates until it has executed the sequence successfully, it cannot carry out the next move, or it has obtained a false evaluation of a test. Clearly, this notion of a caterpillar's evaluation leads to a set of nodes in a tree that are the context of the caterpillar.

In general, we want to be able to specify contexts for all trees of a given grammar or SGML DTD and we may not be able to do so with a single caterpillar. For example, consider document trees in which emphasized text can be nested (as it can be in LaTeX); see Fig. 2. Typically, we emphasize text in a

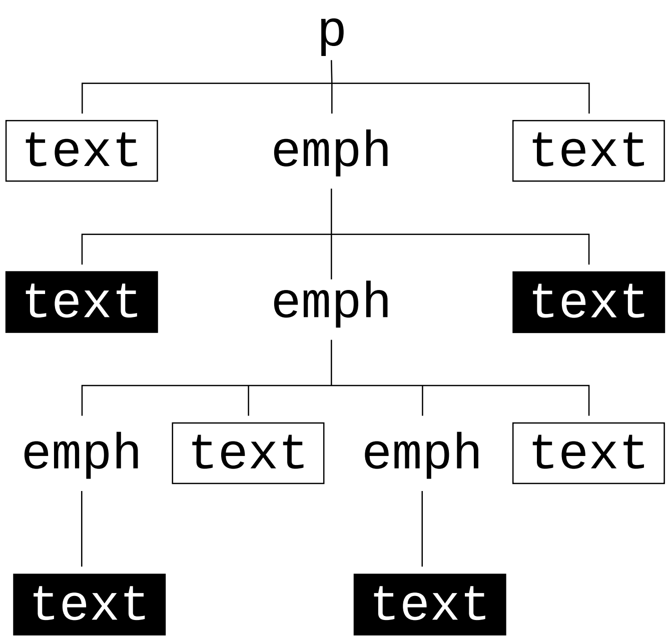

Fig. 2. A document tree with nested emphasized text.

roman environment by setting it in italic whereas we emphasize text in an italic environment by setting it in roman. Thus, we need to determine the parity of the nesting to be able to specify the appropriate typeface.

The caterpillar

$$\texttt{text}, \texttt{up}, \texttt{p}$$

and the caterpillar

$$\texttt{text}, \texttt{up}, \texttt{emph}, \texttt{up}, \texttt{emph}, \texttt{up}, \texttt{p}$$

specify contexts with even parity whereas the caterpillars:

$$\texttt{text}, \texttt{up}, \texttt{emph}, \texttt{up}, \texttt{p}$$

and

$$\texttt{text}, \texttt{up}, \texttt{emph}, \texttt{up}, \texttt{emph}, \texttt{up}, \texttt{emph}, \texttt{up}, \texttt{p}$$

specify odd-parity contexts in the tree of Fig. 2.

We need, however, to specify caterpillars of any length as the depth of emphasis nesting is not bounded, even though it is finite. The solution is simple, yet powerful.

Rather than allowing only a finite number of caterpillars for each context, we allow infinite sets of caterpillars. Since we can consider a caterpillar to be a string over the set of tests and nonterminals (or elements) of the grammars, a set of caterpillars is a language in the usual language-theoretic sense.

We use regular expressions, called **caterpillar expressions,** to specify such languages, called **caterpillar languages,** and we use finite-state machines to model their execution. For example, we specify all even-parity-emphasis contexts for trees of the form given in Fig. 2 by the following caterpillar expression:

$$\texttt{text} \cdot [\texttt{up} \cdot \texttt{emph} \cdot \texttt{up} \cdot \texttt{emph}]^* \cdot \texttt{up} \cdot \texttt{p}$$

and the odd-parity-emphasis contexts as follows:

$$\texttt{text} \cdot \texttt{up} \cdot \texttt{emph} \cdot [\texttt{up} \cdot \texttt{emph} \cdot \texttt{up} \cdot \texttt{emph}]^* \cdot \texttt{up} \cdot \texttt{p}.$$

Our method of specifying contexts singles out those nodes in the trees of a grammar for which the execution of one of the caterpillars in the given caterpillar language evaluates to true when started on these nodes. One immediate implication of this model is that we must separately specify even-parity contexts and even-parity contexts since we need to specify different actions in each case.

3 Caterpillar expressions and tree automata

We provide the formal definition of caterpillars and caterpillar languages, and demonstrate how we can model their execution with tree automata. Merk [17] has investigated some of the basic properties of caterpillar expressions.

Document trees (or abstract syntax trees) have node labels from an alphabet Σ. Since we view SGML (and grammatical) documents as trees, element names (or nonterminals) are the node labels. The content of a document is represented as an external node or leaf of such a tree whose label is also in Σ. Each SGML DTD (or document grammar) defines a set of trees $\mathcal{T}$. Not every set of trees we are discussing needs to be defined by an SGML DTD though. Given a set $\mathcal{T}$, a context $\mathcal{C}$ for $\mathcal{T}$ maps any tree T in $\mathcal{T}$ to a subset $\mathcal{C}(T)$ of $nodes(T)$. Hence, we say that a node v of T is in the context $\mathcal{C}$ or belongs to $\mathcal{C}$ if and only if $v \in \mathcal{C}(T)$. In other words, $\mathcal{C}(T)$ denotes the nodes of T that are in context $\mathcal{C}$.

Let Δ denote the alphabet of moves and tests; that is,

$$\Delta = \{\texttt{up}, \texttt{left}, \texttt{right}, \texttt{first}, \texttt{last}, \texttt{isfirst}, \texttt{islast}, \texttt{isleaf}, \texttt{isroot}\}.$$

A string x over $\Sigma \cup \Delta$ is a caterpillar and it denotes a context c_x for the set of Σ-labeled trees as follows. For any tree T and any node v of T:

- if $x = \lambda$, then v belongs to c_λ.
- if $x = aw$, where $a \in \Sigma \cup \Delta$ and $w \in (\Sigma \cup \Delta)^*$, v belongs to c_{aw} if and only if one of the following conditions holds:
  1. a is in Σ, v has label a and v belongs to c_w.
  2. $a = \texttt{up}$, v has a parent $u \in T$ and u belongs to c_w.
  3. $a = \texttt{left}$, v has a direct left sibling $u \in T$ and u belongs to c_w.
  4. $a = \texttt{right}$, v has a direct right sibling $u \in T$ and u belongs to c_w.
  5. $a = \texttt{first}$, v has children in T and v's leftmost child is u and u belongs to c_w.
  6. $a = \texttt{last}$, v has children in T and v's rightmost child is u and u belongs to c_w.
  7. $a = \texttt{isfirst}$, v is the leftmost node among its siblings in T and v belongs to c_w.
  8. $a = \texttt{islast}$, v is the rightmost node among its siblings in T and v belongs to c_w.
  9. $a = \texttt{isleaf}$, v is an external node in T and v belongs to c_w.
  10. $a = \texttt{isroot}$, v is the root node of T and v belongs to c_w.

Note that this formal definition of the meaning of a caterpillar corresponds to the informal notion we used in the previous section.

A language L over the alphabet $\Sigma \cup \Delta$ also denotes a context, namely

$$c_L = \bigcup_{x \in L} c_x.$$

Hence, a node v of a tree T is in the context c_L of language L if, starting from v, it is possible to perform at least one sequence $x \in L$ of moves and tests in T.

We call $\Sigma \cup \Delta$ the caterpillar alphabet of Σ. A language, regular expression or finite-state machine over a caterpillar alphabet is called a caterpillar language, a caterpillar expression, or a caterpillar machine, respectively. We now restrict our attention to **regular caterpillar languages** (languages defined by caterpillar expressions) and to **regular caterpillar contexts** (contexts defined by regular caterpillar languages). We can define how caterpillar machines operate on trees basing the definition on sequences of configurations in the standard manner. A configuration in this case consists of a node of a tree (the current node), a state of the machine (the current state), and a string x over the caterpillar alphabet (the remaining input string).

Theorem 1. *Given a tree T with m nodes and a caterpillar machine M with n transitions, we can compute the set of nodes in T that are in the context $c_{L(M)}$ in time $O(m \cdot n)$.*

We can define tree automata using the approach of Thatcher [22]. The key point about his approach is that the node labels are not ranked, any node label may label any node in a tree independently of the number of children that node has. We omit the formal definition of tree automata here since it is well documented in the literature and the definition is technically laborious even though the concept is simple.

We use a variant of tree automata called **endmarked tree automata** which are tree automata with two designated labels $\top$ and $\perp$ that are used as endmarkers for the trees it processes.

A tree T with endmarkers has labels from an alphabet Σ that contains the endmarkers $\top$ and $\perp$; furthermore, T must be labeled in such a way that:

1. The root of T is labeled $\top$.
2. Each external node is labeled $\perp$.
3. The root is has only one child.
4. External nodes have no siblings.
5. All other nodes in T are labeled with symbols from $\Sigma - \{\top, \perp\}$.

Tree automata come in two flavors: bottom-up tree automata and top-down tree automata. In addition, they can be deterministic or nondeterministic. The following results are well known.

Proposition 2. *For every tree language L, L is accepted by a nondeterministic bottom-up tree automaton if and only if it is accepted by a nondeterministic top-down tree automaton.*

Proposition 3. *For every tree language L, L is accepted by a nondeterministic bottom-up tree automaton if and only if it is accepted by a deterministic bottom-up tree automaton.*

This result does not hold for top-down tree automata.

To explore the relationship between regular caterpillar languages and tree automata we introduce the notion of **caterpillar tree-regular languages.** A language L of trees is a caterpillar tree-regular language if there is a regular caterpillar context c such that T is in L if and only if the root of T is in the context c. We state the following new results without proof:

Theorem 4. *For each extended context-free grammar, the set of its syntax trees is a caterpillar tree-regular language.*

Theorem 5. *Each finite tree language is caterpillar tree regular.*

Theorem 6. *There are (finite) caterpillar tree-regular languages that cannot be accepted by any deterministic top-down tree automaton.*

Theorem 7. *A finite union of caterpillar tree-regular languages is also a caterpillar tree-regular language.*

Given a string language L, it denotes a tree language $RF(L)$, the **root-to-frontier language of L,** as follows: A tree T is in $RF(L)$ if and only if the labels of each root-to-frontier path in T form a string in L.

Theorem 8. *If L is a regular string language, then its root-to-frontier tree language $RF(L)$ is a caterpillar tree-regular language.*

We now state, again without proofs, the results that demonstrate that caterpillar tree languages are deterministic bottom-up tree-automaton languages. We prove this result indirectly using two-way nondeterministic tree automata. In such automata, the computation starts at the frontier of a tree and terminates at the root as with bottom-up tree automata, but in addition it also allows the computation to locally move up or down the tree. The similarity to two-way finite-state machines should be clear.

Theorem 9. *Each caterpillar tree-regular language is accepted by a two-way nondeterministic automaton.*

Finally, we can emulate two-way tree automata with bottom-up tree automata to give the last result.

Theorem 10. *Each caterpillar tree-regular language is accepted by a deterministic bottom-up tree automaton.*

4 Closing remarks

We can reprove two of Kilpeläinen's results [11] about tree inclusion problems using the notion of caterpillar context. The proofs are simpler and more uniform than Kilpeläinen's original proofs.

Lastly, a number of readers of the first draft of this paper mentioned the similarity to the programming language developed in the Logo Group of the MIT Artificial Intelligence Laboratory [1]. It was designed to explore mathematics by students who ranged in age from preschool to postdoctoral. One of the cute aspects of the language is the use of turtle geometry to investigate geometrical notions. It was called turtle geometry because a user manipulated a "turtle" (a representation of the cursor's current position) with a simple command-based language. Children, and even adults, were able to draw complex figures very quickly using the turtle as metaphor and guide. We hope that our use of caterpillars will garner a similar response from graphics designers.

References

1. H. Abelson and A.A. diSessa. *Turtle Geometry: The Computer as Medium for Exploring Mathematics.* MIT Press, Cambridge, MA, 1980.
2. H. Alblas and B. Melichar. *Attribute Grammars, Applications and Systems.* Springer-Verlag, Heidelberg, 1991. LNCS 545.

3. J. Bosak. Web page on DSSSL online. http://sunsite.unc.edu/pub/sun-info/standards/dsssl/dssslo/do960816.htm, 1997.
4. A. Brüggemann-Klein. Formal models in document processing. Habilitationsschrift. Faculty of Mathematics at the University of Freiburg, 1993.
5. A. Brüggemann-Klein and S. Hermann. Design by Example: A user-centered approach to the specification of document layout. In F. Rowland and J. Meadows, editors, *Electronic Publishing '97: New Models and Opportunities. Proceedings of an ICCC/IFIP Conference held at the University of Kent at Canterbury, England, 14-16 April 1997*, pages 223–236. ICCC Press, 1997. URL: `http://www11.informatik.tu-muenchen.de/proj/designer/canterbury.ps`.
6. A. Brüggemann-Klein and T. Schroff. Grammar-compatible stylesheets. In C. Nicholas and D. Wood, editors, *Proceedings of the Third International Workshop on Principles of Document Processing (PODP 96)*, pages 51–58, Heidelberg, 1996. Springer-Verlag. Lecture Notes of Computer Science 1293.
7. A. Brüggemann-Klein and D. Wood. Electronic style sheets. Interner Bericht 45, Institut für Informatik, Universität Freiburg, January 1992.
8. D. Connolly. W3C web page on XML. http://www.w3.org/XML/, 1997.
9. M. P. Consens, F. C. Eigler, M. Z. Hasan, A. O. Mendelzon, E. G. Naik, A. G. Ryman, and D. Vista. Architecture and application of the Hy+ visualization system. *IBM Systems Journal*, 33(3):458–476, 1994.
10. ISO 8879: Information processing—Text and office systems—Standard Generalized Markup Language (SGML), October 1986. International Organization for Standardization.
11. P. Kilpeläinen and H. Mannila. Grammatical tree matching. Report C-1991-61, Department of Computer Science, University of Helsinki, Finland, 1991.
12. D. E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.
13. P. E. Lauer, P. R. Torrigiani, and M. W. Shields. COSY: A system specification language based on paths and processes. *Acta Informatica*, 12:109–158, 1979.
14. H. Lie. W3C web page on Cascading Style Sheets. http://www.w3.org/Style/css/, 1997.
15. H. Lie and B. Bos. *Cascading Style Sheets: Designing for the Web*. Addison-Wesley Publishing Company, Reading, MA, 1998.
16. A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal of Computing*, 24(6), December 1995.
17. M. Merk. Spezifikation von Mustern als Kontexte. Master's thesis, Institut für Informatik, Universität Freiburg, 1994.
18. John David Morgenthaler. *Static Analysis for a Software Transformation Tool*. PhD thesis, University of California, San Diego, Department of Computer Science and Engineering, 1997. Also available as Technical Report CS97-552 and from URL: `http://www-cse.ucsd.edu/users/jdm/Papers/Dissertation.html`.
19. E. V. Munson. *Proteus: An Adaptable Presentation System for a Software Development and Multimedia Document Environment*. PhD thesis, Computer Science Division, University of California, Berkeley, 1994.
20. M. Murata. Transformation of documents and schemas by patterns and contextual conditions. In C. Nicholas and D. Wood, editors, *Proceedings of the Third International Workshop on Principles of Document Processing (PODP 96)*, pages 153–169, Heidelberg, 1997. Springer-Verlag. Lecture Notes in Computer Science 1293.

21. W. Schreiber. *Generierung von Dokumentverarbeitungssystemen aus formalen Spezifikationen von Dokumentarchitekturen.* PhD thesis, Institut für Informatik, Technische Universität München, 1996.
22. J.W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1:317–322, 1967.
23. X. Wang and D. Wood. Xtable—A tabular editor and formatter. In A. Brown, A. Brüggemann-Klein, and A. Feng, editors, *EP96, Proceedings of the Sixth International Conference on Electronic Publishing, Document Manipulation and Typography*, pages 167–180, 1996. Special Issue, Electronic Publishing—Origination, Dissemination and Design 8(2 and 3).