DRACULAE Version 0.3 User's Manual

Diagram Recognition Application for Computer Understanding of Large Alegbraic Expressions

Richard Zanibbi
Diagram Recognition Laboratory
Queen's University
Kingston, Ontario, CANADA

April 30, 2004

1 Introduction

DRACULAE implements a tree-transformation based approach to recognizing the syntax and semantics of mathematical expressions [3, 5, 4], implemented in the TXL tree transformation language [1, 2]. DRACULAE was first implemented in Java in 1999. Using TXL, which has a convenient pattern-replacement syntax for tree transformations, DRACULAE was completely re-implemented in May to July 2000. Since that time DRACULAE has been through a number of clean-ups and alterations.

DRACULAE produces TEX and operator tree output from a list of symbols with bounding boxes. An operator tree represents the semantics of a mathematical expression, an ordered application of operators to operands. DRACULAE also produces intermediate data in the form of Baseline Structure Trees [4], which describe the location of symbols and/or tokens on baselines in an expression.

There are many dialects of mathematical notation, for instance created when a person defines new operations, which may change the syntax and/or semantics of symbols in an expression. This distribution of DRACULAE defines only a single dialect, but in Section 4 we describe starting points for extending DRACULAE to cope with new dialects, by taking advantage of DRACULAE's compiler-based architecture.

DRACULAE currently recognizes individual expressions (e.g. not spread across lines) which do not contain tabular structures such as matrices and multiple indices on summations. The current symbol layout analysis is simple, and is sometimes fragile as a result (you may notice that superscript/subscript detection may be unreliable on certain inputs, for instance). Also, the expression grammar currently used to produce operator trees is quite limited (though it may be easily extended: see Section 4).

In the remainder of this section we list contributors to DRACULAE, provide notes on copying and distributing DRACULAE, and summarize the subdirectories of DRACULAE. In subsequent sections we describe how to use DRACULAE, give an overview of how DRACULAE processes input, provide notes on altering and extending DRACULAE, and finally give contact information for bugs, comments and suggestions.

1.1 Contributors

DRACULAE's currently has only one author, Richard Zanibbi. Version 0.1 was implemented in Java in 1999, and this was reworked into the TXL-based Version 0.2 between May 2000 and March 2002. Version 0.3 makes use of the now freely available TXL compiler.

1.2 Copying and Distribution

1.3 Subdirectories

Here is a summary of the DRACULAE subdirectories.

bin/: contains DRACULAE executable programs, and a test script for TXL.

> **old/:** unused Bourne Again Shell (bash) scripts for running DRACULAE in Version 0.2; kept here for reference.

doc/: contains the DRACULAE documentation you are now reading.

examples/: contains some example files to try running the different DRACULAE scripts on, and to provide examples of legal input files.

src/: contains the DRACULAE source code. In the src/ directory are the main programs (.Txl files) corresponding to the DRACULAE application programs. The subdirectories of src/ contain supporting grammars and rules for these programs:

> **Grammars/:** context-free grammars (.Grammar files).
>
> **Rules/:** rewrite functions and rules (.Rules files).

test/: simple test program and file used to test the TXL installation.

1.4 Installation

See the README. After installing TXL on your system, issue "make install." This will test your TXL installation and then build DRACULAE. TXL is available from http://www.txl.ca.

2 Using DRACULAE

There are three programs provided in DRACULAE. TXL source files have the extension .Txl, while executables have the extension .x. When successfully compiled using the provided makefile, a .Txl program will be used to generate an executable placed in the bin/ directory.

TXL programs can be run with the txl interpreter rather than compiled if desired. See the TXL documentation for details.

GetTeX.Txl: takes a .dat file describing expressions as lists of symbols with bounding boxes, returning a TEX string corresponding to DRACULAE's interpretation of the input.

GetSemantics.Txl: translates a Lexed Baseline Structure Tree (see Section 3) to an operator tree.

AlignSymbols.Txl: takes a single Baseline Structure Tree (BST) in a file as input and aligns and resizes symbol bounding boxes in the BST; this is used as a reformatting operation in the Freehand Formula Entry System (FFES) [6].

Each application may be run one of two ways. The first way is by using the TXL interpreter directly, e.g.:

txl examples/eg_symbols.dat src/GetTeX.Txl − <Command_Line_Arguments>

This runs GetTeX.Txl on the eg_symbols.dat file in the examples/ directory.

Alternatively, the faster compiled version of the program can be used. To run the compiled version of GetTeX.Txl (GetTeX.x) on the previous example, issue:

bin/GetTeX.x examples/eg_symbols.dat − <Command_Line_Arguments>

We we can obtain the expression's operator tree using the compiled TXL program bin/GetSemantics.x, like so:

bin/GetSemantics.x examples/DRACULAE.bst

We recommend looking at the FFES documentation and source code if you are interested in using AlignSymbols.x/.Txl (it makes more sense if you can see the result; this is the 'Align' operation of FFES). You can try running an example by issuing:

bin/AlignSymbols.x examples/DRACULAE.bst − <Command_Line_Arguments>

Currently AlignSymbols.Txl aligns only an *initial* BST; this means for instance that the symbols of 'cos x' would all be evenly spaced as 'c o s x' on a baseline, even though the TEX output of GetTeX.Txl would group 'cos' together as a single token.

Note that for GetSemantics.Txl, TXL will return a parse error if there are unrecognized structures in the input. This is not a bug, but rather a restriction of the simple expression grammar distributed with DRACULAE (see Section 4). Also, if you try and use an input file with an improper format with any of the programs, TXL will report a parse error.This is because all TXL programs begin by parsing their input using a context-free grammar. The grammar files defining input formats are described in Section 4.

2.1 Command Line Options

GetSemantics.Txl takes no command line options. However, **GetTeX.Txl** and **AlignSymbols.Txl** take the following arguments:

-help shows command line options (note: passes input directly to output)

-bleft indicate that the origin of the input if bottom-left (default is top-left)

-thresholdRatio <0.0-0.5> set ratio used to define symbol region thresholds (default: 1/6 (0.166667))

-centroidRatio <0.0 - 0.5> set ratio used to define symbol centroid Y-coordinates (default: 1/4 (0.25))

GetTeX.Txl/.x also takes the following two options, which are used to output the intermediate Baseline Structure Trees (BSTs) produced by GetTeX.Txl.

-intDir <dir> set directory to write intermediate output (.bst file) (produces file DRACULAE.bst unless filename given using the -intFile flag)

-intFile <file name> set filename of intermediate output (.bst file) (places file in the current directory unless intermediate output directory is given by -intDir flag)

3 A Quick Overview of Processing in DRACULAE

DRACULAE is structured similar to a compiler. Figure 1 shows the passes used in DRACULAE, which we will describe briefly here. For a more complete description of DRACULAE, please consult our research papers about DRACULAE [5, 4]. In the DRACULAE source, the Layout, Lexical and LATEX Generation passes correspond to GetTeX.Txl, and the Expression Analysis Pass corresponds to GetSemantics.Txl.

1. *Layout Pass:* a list of symbols and bounding boxes given as input are converted into a Baseline Structure Tree (BST), describing the layout of symbols in an expression. A baseline in a mathematical expression is a left-to-right ordered list of symbols intended to be perceived as adjacent. The *dominant* baseline of an expression is the baseline containing the symbols and operators that dominate each horizontally adjacent subexpression and the operators connecting them. In a baseline structure tree, symbols and regions (e.g. EXPRESSION,ABOVE,BELOW) are represented by individual tree nodes. Baseline symbols appear as left-to-right ordered child nodes of region nodes. The dominant baseline of each region is represented as child symbol nodes of a region node.

   In Figure 1b the Baseline Structure Tree for Figure 1a is given. It contains four baselines in four regions; the region containing the whole expression (EXPRESSION), the superscript region relative to the A (SUPER), and the regions above and below the fraction line (ABOVE,BELOW).

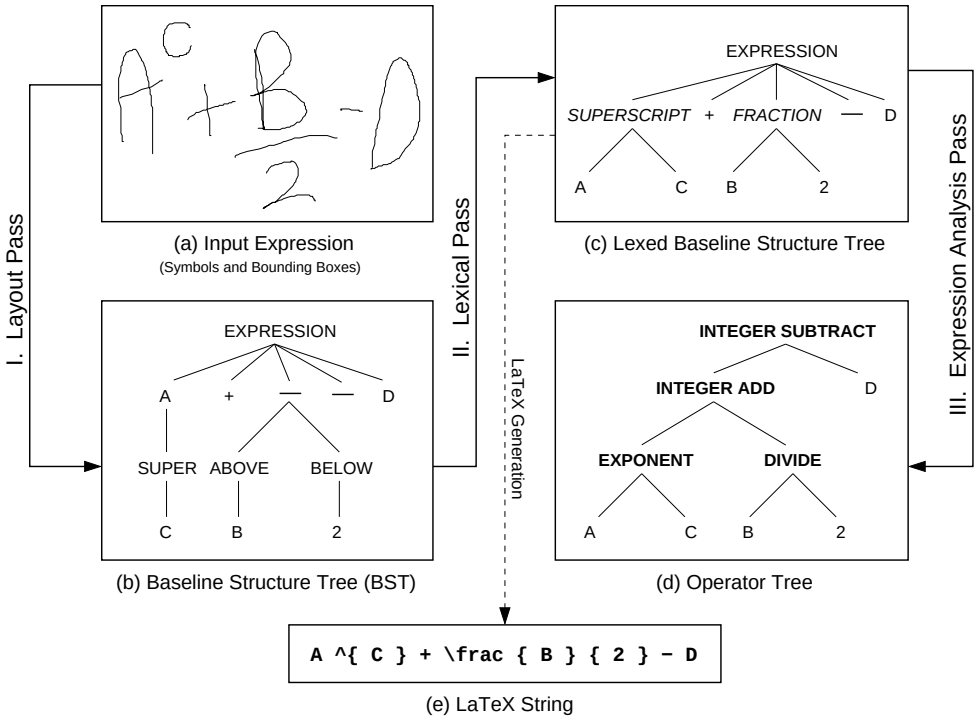

Figure 1: Overview of Processing in DRACULAE

The dominant baseline of the expression appears as the child nodes of the root of the tree (EXPRESSION).

In DRACULAE's Layout Pass, symbol positions are represented using a single point, called a *centroid*. The nested regions around symbols are determined using thresholds. The location of these thresholds and the centroid y-position can be controlled using the '-centroidRatio' and '-thresholdRatio' command-line arguments with GetTeX.Txl. Both centroids and regions are assigned based on the *Symbol Class* that an input symbol belongs to (see Section 4).

The Layout Pass builds a Baseline Structure Tree by recursively:

(a) Locating the leftmost symbol of the dominant baseline in a region (originally the entire expression). For example, in Figure 1a, the dominant baseline of the entire expression begins with 'A', as seen in Figure 1b.

(b) Locating all remaining symbols on the dominant baseline left-to-right. In Figure 1a, the remaining dominant baseline symbols are '+,-,-,D', as shown in Figure 1b.

(c) Partitioning all symbols not on the dominant baseline into nested regions relative to the baseline symbols. Figure 1b shows the 'C', 'B' and '2' partitioned into nested regions of the dominant baseline symbols.

(d) Recursively applying 1-3 in each nested region relative to the dominant baseline containing more than one symbol. In Figure 1b the nested regions of the dominant baseline contain only single symbols, so no further processing is needed.

2. *Lexical Pass:* lexical analysis of the tokens (e.g. symbols, function names, operators) and relations (ABOVE/BELOW/etc.) in a Baseline Structure Tree is performed. Function names, decimal numbers and compound symbols (e.g. $\geq, \leq, =$) are grouped into single units, and structures comprised of multiple baselines are explicitly labelled (e.g. the fraction in Figure 1). This lexical step simplifies the parsing required in the next pass, by explicitly labelling tokens and vertical structures.

3. *Expression Analysis Pass:* the Lexed BST is parsed using an expression grammar, and then transformed to produce an operator tree. The parse constructs a conventional mathematical

expression parse tree, in which operators and operands are located according to their precedence and associativity. Tree rewrites are then used to pre-order the operators in the tree (e.g. '2 + 2' becomes '+ 2 2'), and to make implicit operators explicit (e.g. 'ab' becomes 'MULTIPLY a b').

4 Altering and Extending DRACULAE

This section contains some quick pointers for people who wish to alter and/or extend DRACULAE. Before looking at the source code, we strongly recommend you read the TXL documentation available from http://www.txl.ca.

BST Grammar: the grammar defining legal Baseline Structure Trees and input file formats for the Layout, Lexical and LaTeX Generation passes is located in Grammars/BST.Grammar.

Layout Pass: here are some of the ways you can alter the Layout Pass (Rules/Layout_Analysis.Rules):

Thresholds: the thresholds defining regions around symbols in the different Symbol Classes are in Rules/Thresholds.Rules.

Centroid Positions: the functions that assign symbol centroid positions are located in Rules/SymbolFunctions.Rules.

Symbol Classes: the Symbol Classes specifying how centroids and thresholds are assigned to symbols are defined in Grammars/Symbol_Classes.Grammar. You may add to, or provide alternatives to this file to add new symbols to their appropriate symbol class, or to change the Symbol Class of a symbol.

Layout Search Functions: if you wish to examine or alter the search functions START and HOR used to locate baselines, start by looking at Rules/Regular_Hor.Rules, NoSuper-Sub_Hor.Rules and Start.Rules, used by GetTeX.Txl.

Lexical Pass: lexing of tokens occurs before lexing of relations (regions) in DRACULAE. Token lexing is in Rules/Token_Lexing.Rules, and relation lexing is in Relation_Lexing.Rules (these are both used by GetTeX.Txl. Both are sets of fairly simple tree rewrites, and you can alter or replace these to better recognize structures, or identify new structures.

Expression Analysis Pass: Grammars/Expression.Grammar is used by GetSemantics.Txl to parse a Lexed BST before producing an operator tree. You may change the grammar, or create alternative grammars, to cope with new math dialects containing structures not defined in the current grammar. You may also need to change the rewrites in Rules/Semantics.Rules to properly handle new dialects.

TeX Mappings: the routines producing TeX output from a Lexed BST are in TeX_Output.Rules (main function) and TeX_Symbol_List.Rules (defines symbol and label mappings).

Origin Mapping: Rules/MoveOrigin.Rules defines functions used to move the origin for input symbols from top-left to bottom-left (standard Cartesian) and back. This has been buggy, and may be a good place to look if you obtain really strange behaviour from DRACULAE.

4.1 Handling Dialects

In an (ideal) version of DRACULAE designed to handle multiple dialects, DRACULAE would allow the set of rewrites and grammars to be used in each pass to be given as arguments. For example, the user might be able to give the name of files containing the Lexical pass rewrites and expression grammar for the Expression Analysis Pass that match the dialect of a given input, rather than being restricted to a single language definition, which is currently the case.

5 Bugs, Comments and Suggestions

If you locate a bug while using DRACULAE, please send the problem input file with a message describing the bug to zanibbi@cs.queensu.ca. If this is not possible or appropriate, simply send a message describing the bug. **Please note** that DRACULAE is not designed to handle tabular structures such as matrices and multiple indices on summations etc. Also, we are already aware that the threshold and centroid layout model used in DRACULAE is fragile (e.g. superscript/subscript detection can be unreliable, for instance).

If you have any comments or suggestions about DRACULAE, we're interested in hearing them. Please send your comments and suggestions to zanibbi@cs.queensu.ca.

Acknowledgements

My supervisors Dorothea Blostein and James R. Cordy have contributed a great deal to the design of DRACULAE, along with Steve Smithies, Kevin Novins (University of Auckland, New Zealand) and James Arvo (California Institute of Technology). This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] J.R. Cordy, I. Charmichael, and R. Halliday. *The TXL Programming Language - Version 10*. TXL Software Research Inc., Kingston, Ontario, Canada, Jan. 2000.

[2] J.R. Cordy, C.D. Halpern, and E. Promislow. TXL: A rapid prototyping system for programming language dialects. *Computer Languages*, 16(1):97–107, Jan 1991.

[3] R. Zanibbi. Recognition of mathematics notation via computer using baseline structure. Technical Report ISBN-0836-0227-2000-439, Dept. Computer Science, Queen's University, Kingston, Ontario, Canada, August 2000.

[4] R. Zanibbi, D. Blostein, and J.R. Cordy. Baseline structure analysis of handwritten mathematics notation. In *Proc. Sixth Int'l Conf. Document Analysis and Recognition*, pages 768–773, 2001.

[5] R. Zanibbi, D. Blostein, and J.R. Cordy. Recognizing mathematics notation using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1455–1467, 2002.

[6] R. Zanibbi, K. Novins, J. Arvo, and K. Zanibbi. Aiding manipulation of handwritten mathematical expressions through style-preserving morphs. In *Proc. Graphics Interface*, pages 127–134, 2001.