DITAA Specification Document

Christopher Menart | Brad Schneider

(Specifications for ‘additional functionality’ are not included in this version of this document, pending determination of final ‘additional functionality’ requirements.)

Input Specification

The DITAA application accepts input files with a variety of ASCII constructs which are translated into diagram elements. This section specifies the various structures that may be provided to the DITAA application.

Overview

Allowable inputs can consist of free text and line groups. Line groups consist of any number of straight line components connected by corners. Any cycles in a line group which are not bisected by other lines from the same group form *closed shapes*, which are rendered with drop shadow and may be impacted by additional styling elements from both the input and command-line arguments.

Since the input language for DITAA is a two-dimensional language (a *picture language*), we express the grammar rules as either row vectors, column vectors, or two-dimensional matrices. The grammars represent most common structures for the elements of the input, but do not always express some of the special cases that may occur where elements overlap as this would lead to an unreasonably large number of element types that would likely be more difficult to understand.

Note that any text occurring in the document and not conforming to the following specification is treated as plain text and inserted without translation into the bitmap graphic output. The position of the plain text relative to the diagram elements is retained.

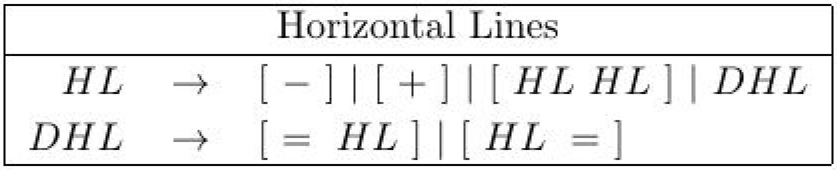
Lines and Closed Shapes

Horizontal Line Characters = {‘-’, ‘=’}
Vertical Line Characters = {‘|’, ‘:’}
Corner Characters = {‘+’, ‘/’, ‘\’}

Note that ‘*’ character may also count as belonging to ‘corner characters’ under certain circumstances! See section ‘Point Markers’.

A **horizontal line** consists of a sequence of one or more consecutive horizontal line and/or corner characters, containing at least one horizontal line character.

The following grammar rules describe the syntax of horizontal lines (*HL*) and dashed horizontal lines (*DHL*).



Horizontal Lines		
HL	$\rightarrow$	$[\,-\,] \mid [\,+\,] \mid [\,HL\ HL\,] \mid DHL$
DHL	$\rightarrow$	$[\,=\ HL\,] \mid [\,HL\ =\,]$

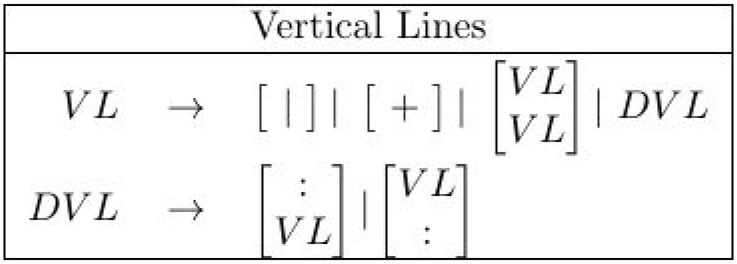
A **vertical line** is a collection *E* of one or more vertical line and/or corner characters, containing at least one vertical line character, such that

∃ natural numbers a, b, c s.t.

∀ i in [a, b], input[c,i] ∈ *E*

Each horizontal and vertical line character is considered to be part of the *largest* such line that can be defined. Lines are to be rendered as contiguous straight lines in the output.

The following grammar rules describe the syntax of vertical lines (*VL*) and dashed vertical lines (*DVL*).



Vertical Lines		
VL	$\rightarrow$	$[\,\vert\,] \mid [\,+\,] \mid \begin{bmatrix} VL \\ VL \end{bmatrix} \mid DVL$
DVL	$\rightarrow$	$\begin{bmatrix} : \\ VL \end{bmatrix} \mid \begin{bmatrix} VL \\ : \end{bmatrix}$

A **corner** is a corner character within a horizontal or vertical line. A corner can be a component of *both* a horizontal line and a vertical line.

While the corner character ‘+’ appears as a terminal in the grammar rules for both horizontal and vertical lines, semantic rules overlaid on the grammar dictate that the ‘+’ character is only expected where two lines meet (corners of shapes, and perpendicular intersections of lines).

Lines consisting of solely a corner character are somewhat ambiguously defined by the grammar above (a single ‘+’ character is both a horizontal and vertical line). However, this definition is necessary to account for the special case of a 2x2 matrix of ‘+’ characters, which does produce a closed shape.

A **line group** is a collection *L* of one or more horizontal, vertical lines, and corners such that

∀ corners *c0, c1* ∈ *L*, they can be arranged in a sequence *c0, c1, c2* s.t.
∀ c in [0, #L-1], *ci* and *c(i+1)* are joined by a line ∈ *L* or adjacent
AND all lines ∈ *L* are adjacent to one of the corners in *L*.

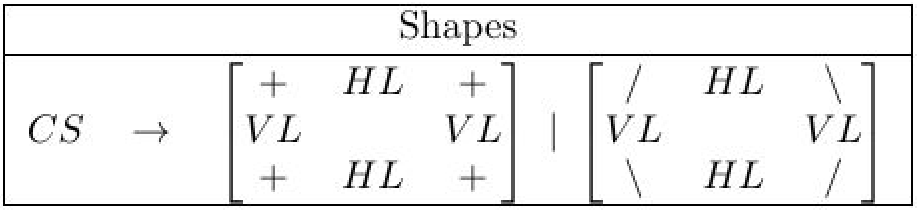
[old: ∀ lines *l0*, *ln* ∈ *L*, ∃ a sequence of lines l0, l1, l2 … *ln* ∈ *L* s.t.
∀ i in [0, n-1], *li* and *l(i+1)* share a corner]

If shared lines/adjacency can be modeled as edges which join two lines as nodes in a graph, a line group is a connected component.

Note that by the above definition, it is possible to have a line group with no lines, only corners.

A **closed shape** is a cycle of corners in a line group which does not contain any smaller cycles (i.e. it is a minimal cycle).

A closed shape will be rendered as such in the output diagram, with a drop shadow. The following grammar rules describe the format of a closed shape (*CS*) in the input. It is assumed in this case that the length of the horizontal lines are equal and the length of the vertical lines are equal. As described above, the lines meet at the corner characters, '+' (square corners) or '\' and '/' (round corners) to form a line group.



Shapes
$CS \quad \rightarrow \quad \begin{bmatrix} + & HL & + \\ VL & & VL \\ + & HL & + \end{bmatrix} \; \Big| \; \begin{bmatrix} / & HL & \backslash \\ VL & & VL \\ \backslash & HL & / \end{bmatrix}$

Dashed Lines

Dashed Line Characters = {':', '='}

Any line or line group containing one or more dashed line characters is rendered as with dashed instead of solid lines.

Styling Elements/Modifiers to Closed Shapes

Input text is **inside** a closed shape if

```
NOT ∃ a sequence of coordinates (I, J) s.t.
    I[0], J[0] is the coordinate of the input text AND
    ∀ i in [0, length(I)-1], |I[i] - I[i+1]| + |J[i] - J[i+1]| = 1 AND
    ∀ i in [0, length(I)-1] I[i], J[i] is NOT the coordinate of a line element from
        the shape AND
    (I[end] ∈ {0, max line length of input} OR J[end] ∈ {0, numlines in input})
```

In English, input text is **inside** a closed shape if there is no path of spaces from the input text to the outer edge of the input that does not pass the lines composing the shape--it is fully surrounded by those lines.

Note that for the purposes of two-dimensional grammar definition, the grammar rules do not indicate the size of the closed shape (that is, the size of the lines that create the enclosure). It is assumed that the shapes are large enough for all characters comprising the modifier to fit within the boundary of the shape.

Input text inside a close shape is a **styling element** if it matches one of the patterns defined as possible style elements that follow.

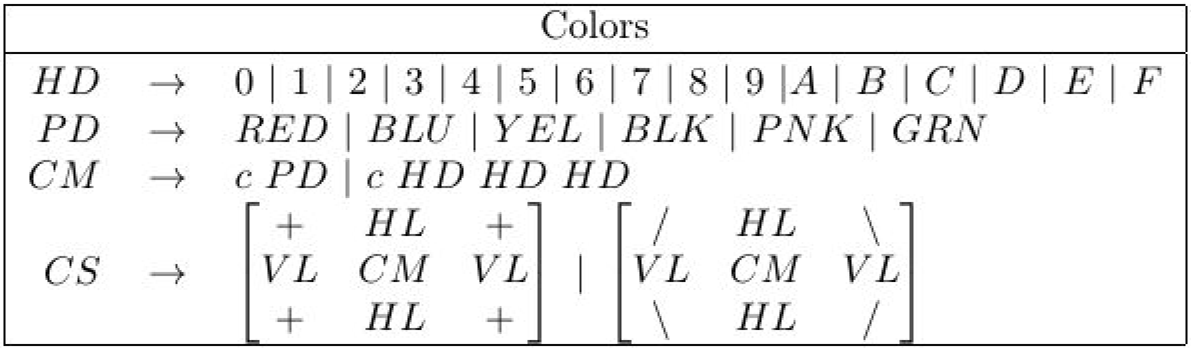
Colors

Hexadecimal Characters = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘0’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’]
Predefined Colors = [‘RED’, ‘BLU’, ‘YEL’, ‘BLK’, ‘PNK’, ‘GRN’]

A color styling element consists of four characters, where the first character is ‘c’, and the next three characters either match one of the strings in the Predefined Colors, or consist only of characters from the Hexadecimal Characters set.

A closed shape with a color styling element is rendered with the indicated color. [List the hex codes of the predefined colors or something?] If there are multiple color elements, only one is applied.

The following table illustrates grammar rules involved with modifying colors.



Colors

$$\begin{array}{lcl} HD & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid A \mid B \mid C \mid D \mid E \mid F \\ PD & \rightarrow & RED \mid BLU \mid YEL \mid BLK \mid PNK \mid GRN \\ CM & \rightarrow & c\,PD \mid c\,HD\,HD\,HD \\ CS & \rightarrow & \begin{bmatrix} + & HL & + \\ VL & CM & VL \\ + & HL & + \end{bmatrix} \mid \begin{bmatrix} / & HL & \backslash \\ VL & CM & VL \\ \backslash & HL & / \end{bmatrix} \end{array}$$

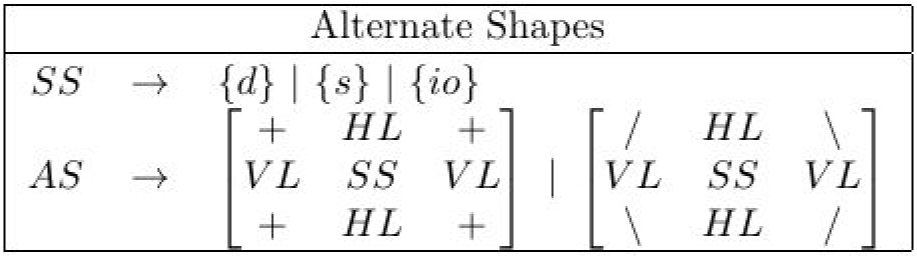
Alternate Shapes

An alternate shape styling element is one of three strings:

1. “{d}”: A closed shape with this tag is rendered with a wavy bottom edge.
2. “{s}”: A closed shape with this flag is rendered as a cylinder.
3. “{io}”: A closed shape with this flag is rendered as a skewed rhombus.

If there are multiple alternate shape styling elements in a single closed shape, only one is applied. They have no effect if the closed shape is not a rectangle (has 4 composing lines).

The following table illustrates a grammar relating to alternate shapes:



Alternate Shapes

$$\begin{array}{lcl} SS & \rightarrow & \{d\} \mid \{s\} \mid \{io\} \\ AS & \rightarrow & \begin{bmatrix} + & HL & + \\ VL & SS & VL \\ + & HL & + \end{bmatrix} \mid \begin{bmatrix} / & HL & \backslash \\ VL & SS & VL \\ \backslash & HL & / \end{bmatrix} \end{array}$$

Bullet Points

If the pattern “ o ” is present, and followed by one or more characters of free text (text not part of another special command as specified in this document) the ‘o’ is rendered as a round bullet point.

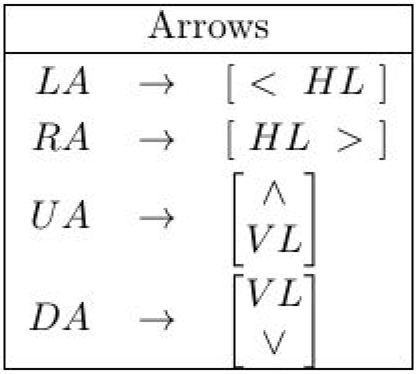
Arrowheads

A character *c* at position *x, y* is an **arrowhead** if any of the following are true:

c == ‘>’ && input[x-1, y] is in a line.

```
        If so, c is rendered as a right-facing arrow at the end of that line.
c == ‘<’ && input[x+1, y] is in a line.
        If so, c is rendered as a left-facing arrow at the end of that line.
C == ‘^’ && input[x, y-1] is in a line.
        If so, c is rendered as an up-facing arrow at the end of that line.
c ∈ {‘v’, ‘V’} && input[x, y+1] is in a line.
        If so, c is rendered as a down-facing arrow at the end of that line.
```

Note that *c* is not considered to be part of the line. If there are more line characters on other sides of *c*, *c* does not join them as a corner would. A set of grammar production rules describing arrows is located below.



Arrows		
LA	$\rightarrow$	$[\, < \;\; HL \,]$
RA	$\rightarrow$	$[\, HL \;\; > \,]$
UA	$\rightarrow$	$\begin{bmatrix} \wedge \\ VL \end{bmatrix}$
DA	$\rightarrow$	$\begin{bmatrix} VL \\ \vee \end{bmatrix}$

Point Markers

Under certain circumstances, the ‘*’ character counts as a corner.

It must appear directly adjacent to at least one horizontal or vertical line character to count, and a second ‘*’ character must not appear directly adjacent to it opposite from that line character.

There is one difference between ‘*’ corners and ‘+’ corners. Two ‘*’ corners that are directly adjacent, with no intervening line characters, are not adjacent for the purposes of forming line groups.

This functionality is still experimental in the pre-existing implementation.

Output Specification

DITAA can be run in two modes, standard or html. In the standard mode of operation, the software outputs a single PNG file capturing the graphical depiction of the input based on the specification rules described above. In the HTML mode of operation, DITAA outputs PNG files and also outputs modified copies of the input HTML file.

In HTML mode, the same input specification rules that are listed previously in this document apply. The input ASCII markup must be contained in a pre-formatted text HTML tag:

<pre class=”textdiagram”> (ASCII contents) </pre>

The tag may optionally contain an ‘id’ attribute, which determines the name of the output PNG file that was built from the ASCII contents. If no ‘id’ attribute is provided, then the name is generated as ‘ditaa_diagram_x.png” where ‘x’ is a number.

The output HTML is a modified version of the input that includes references to the images produced by DITAA via <img> tags in place of the original <pre> tags..

Specifying Non-Functional Requirements

Source Size Reduction

It is a requirement to keep the size of the source code at a maximum of 95% the size of the pre-existing implementation found at <http://ditaa.sourceforge.net/>.

The size metric used will be source-lines-of-code, which will be determined using the Statistic plugin to Intellij IDEA. This tool reports 9580 SLOC for the pre-existing implementation, implying a target of 9101 SLOC.

Bug Fixing

The program should exhibit correct behavior in at least one case which causes a bug in the pre-existing implementation. This bug should be documented with a test case that demonstrates the incorrect/correct behavior, using a file identified as ‘fixedbug.txt’ in the project git repository at <https://github.com/CJMenart/CS-7140-2020>.

Acceptance/Conformance Tests

The program must pass all unit tests provided in the pre-existing implementation by exhibiting the same behavior as the pre-existing implementation, except where it is explicitly noted that behavior should or may diverge. (In particular behavior related to point markers as corners appears to be partly incidental, and is to be considered flexible.)

The program must pass all unit tests written in the course of constructing these specifications, again by matching the behavior of the pre-existing implementation except where otherwise noted. These additional test cases are to be packaged with the final code.

Additional test cases must be constructed for any/all additional functionality beyond that in the pre-existing implementation, as determined by the final requirements. These test cases are also to be packaged with the final code.

While unit testing is used to prove the correctness of specific functions within the implementation, additional testing is required to confirm that the final deliverable meets the specifications in this document. Since there are few high-level functions in the DITAA software, acceptance and conformance testing are largely considered to be one event. Sample inputs of two types will be provided. The first will cover instances of unique element types per input, such as lines, arrows, and closed shapes of varying sizes. The second test input type will cover combinations of these elements used together, testing common structures such as intersecting lines and shapes, combinations of dashed and solid lines, rounded corners, shared edges, etc. Test inputs will be designed such that all possible options (shapes, colors, etc.) are tested in multiple inputs.

Team Member Journals

Christopher Menart

10-2-20:

- Discussing possibility of (context-free?) grammar for specifying inputs?
- Inspecting DITAA test cases for better view of intended program behavior
    - What forms one unbroken line/shape?

10-3-20:

- Running all pre-existing DITAA test cases to see program behavior
    - Best to specify in terms of connected components of lines/corners
- Creating additional test cases for behavior with lots of corner characters
    - Behavior with '*' corners not quite consistent with '+' corners.
- Initial draft of specs of core behavior
    - May change
    - Teammates suggest possible 2-dimensional grammar?

10-6-20:

- Re-writing core DITAA specs (line/line group specs) to better match behavior
    - Behavior with directly-adjacent corners encourages specifying connected groups of corners instead of connected groups of lines

10-7-20:

- Writing specifications for most additional features (styles, decorations, 'Other' features, etc.)

10-9-20:

- Writing specs for non-functional requirements
- Summarizing journal for technical report

Brad Schneider

2020-10-02

- Began looking at spec document requirements
- Researched ways to describe 2d grammars
    - Sent PDF link to teammates

2020-10-06

- More work on interpreting inputs of DITAA into grammars

- Still some internal debate over how specific to get or how far to take this. Developed grammars for lines, arrows, closed shapes. Began to explore grammars around more combinations, e.g. ‘connected arrows’ which begin with a ‘+’ character that intersects another element, but seems that this can go down a rabbit hole and produce a spec that is actually more confusing to read.

2020-10-07

- Decided to eliminate the more specific structures from the grammar.
- Deeper review of draft thus far from teammates

2020-10-09

- Transformed notes on the input language into grammar rules using LaTeX for formatting.

2020-10-11

- Updated spec draft with some narrative and screenshots of grammars for lines, arrows, shapes.
- Researched conformance and acceptance testing. Determined that these are focused on exercising the input grammar of the elements based on the single function of DITAA.