A Constraint-Based Specification for Box Layout in CSS2

Brian Michalowski

Technical Report UW-CSE-98-06-03
Department of Computer Science and Engineering
University of Washington
June 1998

Author's address:

Brian Michalowski
Dept. of Computer Science & Engineering
University of Washington
PO Box 352350
Seattle, Washington 98195 USA
bam@cs.washington.edu

Abstract

Cascading Style Sheets provide a flexible mechanism for governing the appearance of Web pages. Cascading Style Sheets Level 2 (CSS2) are an enhancement to the original CSS1 specification, giving Web page designers additional control over the appearance of Web pages. However, the CSS2 specification is written in English, leaving open the possibility of ambiguity or inconsistency. We present a formalization of a subset of the CSS2 specification using constraints hierarchies to help ensure that potential problems in the specification are caught and corrected. We also comment on the formalization process.

1 Introduction

1.1 Cascading Style Sheets

Cascading style sheets are a mechanism suggested by the World Wide Web Consortium (W3C) to fix a fundamental problem with older versions of HTML — their inability to separate content and appearance. By writing different style sheets, Web users can change the appearance of a document without ever having to edit the original document. The most recent version of cascading style sheets, Cascading Style Sheets Level 2 (CSS2), gives both Web page designers and Web page viewers more control over the font size, colors, and layout of pages than they had before.

The layout of pages in CSS2 is governed by a box model — hierarchical elements of a document are laid out in nested boxes which govern the elements' sizes and locations. This model is similar to the model used by TeX and many other layout packages. These boxes are subject to various constraints, some specified by the user and others specified by the default behavior of the style sheets.

The W3C specification for box layout in CSS2 describes the constraints in plain English, which is useful for communicating the information to humans but can easily be ambiguous or contain inconsistencies. Several members of W3C suggested that we look at the specifications for box layout in CSS2 and formalize them to help remove any ambiguities or inconsistencies in the specification.

1.2 Constraints

Constraints have long been used to describe properties of user interfaces that should be enforced, such as ensuring that the left side of a window should always be to the left of the right side. Since the wording of the CSS2 specification was so constraint-like, constraints seemed like a natural way of formalize the specification. In particular, a *constraint hierarchy* seemed appropriate. A constraint hierarchy is a set of constraints each of which has a strength associated with it indicating how important it is that that constraint be satisfied. Constraint strengths are modeled mathematically as integers, but for convenience are often given symbolic names. For example, in the case of floating boxes in CSS2, it is *required* that floating boxes do not appear before floating boxes that were declared earlier in the document, there is a *strong* preference for a floating box to be as close as possible vertically to where it was declared, and there is a *weak* preference for floating boxes to the be as far to the left (or right, as the case may be) as possible.

In general, whenever a variable in CSS2 is declared, such as the minimum width of a paragraph being 100 pixels or the width of a floating box being 200 pixels, we add a constraint indicating this to the set of constraints in the constraint solver. Every box has constraints associated with it, even ones that don't explicitly assign any values, since constraints also govern the behavior common to all boxes. The only declaration that does not generate a constraint is the setting of a field to *auto*. The CSS2 spec makes frequent use of a value called *auto* to represent the case when the rendering engine should compute the value based on other information. It became apparent that having a field set to *auto* did not involving adding any constraints, merely using the constraints already in the constraint engine to compute that value.

One issue that comes up in computing the best solution to a set of constraints is determining which error metric to use to decide between two potential solutions. Intuitively, a solution S is better than solution T if both solutions match for some number of levels k, but on level $k+1$ S has a smaller error than T. For example, if T and S have the same error for the *required* and *strong* constraints, but S has a smaller error for the *weak* constraints, then S is a better solution.

For the CSS2 constraints, we chose a *locally error better* metric, which means that for each constraint at level $k+1$ the error for S is less than or equal to the error for T, and for at least one of those constraints the error of S is strictly less than the error for T. A *locally predicate better* metric, which treats all constraints that are not satisfied as having the same error, seemed inappropriate for this domain, since it's better for a floating box to only be pushed down 5 lines instead of 50. It did not seem necessary to use a *global* metric, in which the total error at each level is considered instead the error of for each constraint, since in this domain each constraint should be satisfied as well as possible independently of the others, and since generally it is computationally more difficult to find global solutions to constraint hierarchies.

Refer to [1] for a more complete discussion of constraint hierarchies.

1.3 The Scope of This Document

This document attempts to formalize the part of the CSS2 spec concerned with box layout. In the most recent version of the specification, this information was contained in sections 8, 9, and 10. However, not all information in sections 8 through 10 are described here. This document does not describe border styles, background colors, or other attributes that do not affect the layout of boxes.

This document assumes that all information is locally available, and any conversion or inheritance of values has already taken place. For example, it assumes that all height and width percentages have been converted into pixel values. It also assumes that shorthands such as *margin* have been expanded into their separate *margin-top*, *margin-left*, *margin-right* and *margin-bottom* declarations. These sorts of conversions do not affect the final layout of the boxes; they are just shorthands to aid the style sheet creator, and as such are not described in this document. For similar reasons, this document assumes that 'compact' and 'run-in' boxes have already been resolved into 'block' and 'inline' boxes. In addition, left-to-right layout is assumed for the sake of simplicity.

The constraints presented in this document are based on the specification for Cascading Sytle Sheets, Level 2 as presented in http://www.w3.org/TR/REC-CSS2-19980512, last edited on May 12, 1998 [2]. While this is intended to be the final version of the specification, in case the specification has evolved since then, consult http://www.w3.org/TR for the most recent version of the "Cascading Style Sheets, level (2) Specification".

2 The Format of the Constraints

The constraints presented in this document contain both attributes, such as the `width` of a box, and values, such as *auto*. For the remainder of this document, attributes that appear in running text will be display in a `monospaced` font, while values and constraint strengths will be displayed in *italics*. To avoid visual clutter, these items will not be typeset different for constraints listed in tables.

The constraints in the following sections appear in the form:

8.1	req	ref.TC - padding-top?	= ref.TP

The first column indicates which section of the CSS specification contains the English analogue of the constraint, in this case, section 8.1. The second column indicates the strength of the constraint: *req* (required), *strong*, *medium*, *weak*, or *vweak* (very weak). (An additional pseudo-strength, *REMOVE*, indicates that a constraint should be removed from the engine instead of being added.)

The remainder of each line lists the actual relationship that should hold. A "?" after a variable indicates that it is read-only [1]; the value of that variable may not be changed while attempting to satisfy that constraint.

The constraints in this section make use of several terms to describe each node. `Ref` (short for "reference box") describes the box that contains the current node. `Actual` describes where the box is ultimately placed. This can be different from `ref` if the box is positioned relatively. `Previous` refers to the `ref` box of the most recent element in the document tree that is not absolutely positioned, fixed, or a float. `Enclosing` refers to the `actual` box that establishes the containing block for the current node. The exact specifications are enumerated in section 10.1 of the CSS2 specification, but roughly speaking, *enclosing* refers to the viewport or the printed page for fixed boxes and the current node's parent for other types of boxes.

In addition, the following edges are defined:

{T, L, R, B} M = {top, left, right, bottom} margin edge
{T, L, R, B} B = {top, left, right, bottom} border edge
{T, L, R, B} P = {top, left, right, bottom} padding edge
{T, L, R, B} C = {top, left, right, bottom} content edge

{T, L, R, B} O = {top, left, right, bottom} outer edge
{T, L, R, B} I = {top, left, right, bottom} inner edge

content edges and inner edges are synonymous (*C = *I)
border edges have no synonyms
padding edges and containing box edges are synonymous
margin edges and outer edges are synonymous (*M = *O)

3 The Constraints

3.1 Constraints for All Boxes

This section describes constraints that are applicable for all types of boxes.

A few terms used in the constraints of this section are defined here. *Medium-width* is the width in pixels of a border that is declared 'medium', a value that can vary from user agent to user agent. Since boxes attempt to be big enough to hold whatever their content is, *heightOfContent()* is a function that represents the height of whatever a box's contents are. In the case of inline boxes this can get very complicated, and since this activity is beyond the scope of this paper we simply represent it as an externally computed function. In essence, *heightOfContent()* is whatever the height would be if the height were declared to be *auto*. *WidthOfContent()* is analogous, although it usually only has a specific value if the element it describes is a replaced element.

The line 'actual.* = ref.*?' indicates a strong preference for each attribute of the `actual` box to be the same as the corresponding attribute of the `ref` box. Only for relatively and absolutely positioned boxes and fixed boxes are these values different.

8.1	req	ref.TC - padding-top?	= ref.TP
8.1	req	ref.LC - padding-left?	= ref.LP
8.1	req	ref.RC + padding-right?	= ref.RP

8.1	req	ref.BC + padding-bottom?	= ref.BP
8.1	req	ref.TP - border-top?	= ref.TB
8.1	req	ref.LP - border-left?	= ref.LB
8.1	req	ref.RP + border-right?	= ref.RB
8.1	req	ref.BP + border-bottom?	= ref.BB
8.1	req	ref.TB - margin-top?	= ref.TM
8.1	req	ref.LB - margin-left?	= ref.LM
8.1	req	ref.RB + margin-right?	= ref.RM
8.1	req	ref.BB + margin-bottom?	= ref.BM
8.1	req	LC + width	= RC
8.1	req	TC + height	= BC
8.1	medium	height	= heightOfContent()
8.1	medium	width	= widthOfContent()
8.3	vweak	margin-top	= 0
8.3	vweak	margin-right	= 0
8.3	vweak	margin-bottom	= 0
8.3	vweak	margin-left	= 0
8.4	vweak	padding-top	= 0
8.4	vweak	padding-right	= 0
8.4	vweak	padding-bottom	= 0
8.4	vweak	padding-left	= 0
8.5	vweak	border-top-width	= medium-width
8.5	vweak	border-right-width	= medium-width
8.5	vweak	border-bottom-width	= medium-width
8.5	vweak	border-left-width	= medium-width
9.3.1	strong	actual.*	= ref.*?
9.5		forall floating boxes fb to the left	
	req	ref.LO >= fb.RB?	
9.5		forall floating boxes fb to the right	
	req	ref.RO <= fb.LB?	
10.2	req	width	> 0
10.3	vweak	left	= 0
10.3	vweak	right	= 0
10.3	vweak	top	= 0
10.3	vweak	bottom	= 0
10.2	req	height	> 0

3.2 Line boxes

This section attempts to describe the behavior of boxes that are laid out horizontally from left-to-right if possible, and if there is no room on the current row are then laid out on the next row. It also models the fact that floats increase the margins of line boxes to ensure that the line boxes and floating boxes do not overlap. These constraints use the external functions *leftFloat()*, which returns the rightmost left-floating element on the current line, and *rightFloat()*, which returns the leftmost right-floating element on the current line.

```
9.4.2               if (previous.RM + width + margin-left + margin-right
                    + border-left + border-right + padding-left
                    + padding-right <= enclosing.RP)
        strong          ref.LM = previous.RM?
        strong          ref.TM = previous.TM?
                    else
        strong          ref.TM = previous.BM?
        strong          ref.LM = 0
                    endif

9.5     strong      ref.LM? + margin-left >= leftFloat().RM
9.5     strong      ref.RM? - margin-right <= rightFloat().LM
```

3.3 Normally positioned block boxes

Much of this section describes the behavior of margins of normally positioned block boxes when they collapse. The first set of constraints apply when only a set of top margins adjoin (i.e. at the top of a document,) in which case the outermost top margin is set to the appropriate value and the others are set to zero. The second set of constraints applies when both top and bottom margins or only bottom margins adjoin (i.e. at the middle or end of a document.) In this case the outermost bottom margin is set to the appropriate value, and the others are zeroed out.

In this section, {AM} ("adjoining margins") is the set of all adjoining margins at a location that do not involve floating or absolutely positioned boxes. The function *maxpos* returns the positive element of a set which has the greatest absolute value, and *maxneg* returns the negative element of a set with the greatest absolute value.

```
8.3.1               if (TM is part of adjoining margin && previous == null)
        req             ref.TM = maxpos(AM) + maxneg(AM)
                    else if (TM is part of adjoining margin)
        req             ref.TM = 0
                    endif

8.3.1               if (BM is part of adjoining margin && BM is outermost)
        req             ref.BM = maxpos(AM) + maxneg(AM)
                    else if (BM is part of adjoining margin)
        req             ref.BM = 0
                    endif

9.4.1   strong      ref.TM = previous.BM?
9.4.1   strong      ref.LM = enclosing.LC?
10.3.3  strong      margin-left + border-left-width + padding-left + width +
                    padding-right + border-right-width + margin-right = enclosing.width
10.3.3  weak        margin-left = margin-right
10.6.3  strong      margin-top + border-top-width + padding-top + height +
                    padding-bottom + border-bottom-width + margin-bottom = enclosing.height
10.6.3  weak        margin-top = margin-bottom
```

3.4 Position-based constraints

These constraints describe the various effects that positioning a box has on its layout.

position: relative

9.3.2	req	actual.TO = ref.TO? + top?
9.3.2	req	actual.LO = ref.LO? + left?
9.3.2	req	actual.RO = ref.RO? - right?
9.3.2	req	actual.BO = ref.BO? - bottom?

position: absolute || position:fixed

9.3.2	req	actual.TO = enclosing.TO? + top?
9.3.2	req	actual.LO = enclosing.LO? + left?
9.3.2	req	actual.RO = enclosing.RO? - right?
9.3.2	req	actual.BO = enclosing.BO? - bottom?
10.3.7	strong	left + margin-left + border-left-width + padding-left +
		width + padding-right + border-right-width + margin-right + right = enclosing.width?
10.3.7	weak	margin-left = margin-right
10.6.4	strong	top + margin-top + border-top-width + padding-top +
		height + padding-bottom + border-bottom-width + margin-bottom + bottom = enclosing.height?
10.6.4	weak	margin-top = margin-bottom

3.5 Floats

This section describes the constraints that govern the placement of floats. The constraints for both left- and right-floating boxes are enumerated for the sake of completeness, although the constraints for both are very similar. The nine constraints for each type of box follow exactly the nine rules listed in section 9.5.1 for floating box layout.

float:left

9.5.1	req	ref.LO >= enclosing.LP?
9.5.1	req	ref.LO >= lf.RO? or
		ref.TO >= lf.BO?
9.5.1	for all right-floating boxes rf to the right,	
	req	ref.RO <= rf.LO?
9.5.1	ref.TO >= enclosing.TI?	
9.5.1	for all previous block-level or floated box elements pb,	
	req	ref.TO >= pb.TO?
9.5.1	for all previous line box elements pl,	
	req	ref.TO >= pl.TO?
9.5.1	if (ref.LO > enclosing.LI)	
	req	ref.RO <= enclosing.RI?
9.5.1	medium	ref.TO = 0
9.5.1	weak	ref.LO = 0

float: right

9.5.1	req	ref.RO <= enclosing.RP?
9.5.1	req	ref.RO<=rf.LO or
		ref.TO >= lf.BO?
9.5.1	for all left-floating boxes lf to the left,	
	req	ref.LO >= lf.RO?
9.5.1	ref.TO >= enclosing.TI?	
9.5.1	for all previous block-level or floated box elements pb,	
	req	ref.TO >= pb.TO?
9.5.1	for all previous line box elements pl,	
	req	ref.TO >= pl.TO?
9.5.1	if (ref.RO < enclosing.RI)	
	req	ref.LO >= enclosing.LI?
9.5.1	medium	ref.TO = 0
9.5.1	weak	ref.LO = 0

3.6 Clear

Any box can be declared to *clear* left, right, or both, meaning that floating boxes cannot appear to the left, right, or either side of that box, respectively. This section lists the constraints enforcing this property.

clear: left || clear: both

9.5.2	forall previous left-floating boxes lb	
	req	ref.TO >= lb.BO?

clear: right || clear: both

9.5.2	forall previous right-floating boxes rb	
	req	ref.TO >= rb.BO?

3.7 Other constraints

This section describes miscellaneous, usually simple-to-describe constraints not covered elsewhere. In this section, *value* refers to the numeric value that was used in the attribute declaration. For example, if the line "margin-left = 10" were declared in a document, *value* would have the value 10. Since *value* is a constant and not an actual variable, it is not explicitly listed as read-only, even its value would never changes.

margin-{top,bottom}: *value*

8.3	strong	margin-{top,bottom} = *value*
10.6.4	REMOVE	margin-top = margin-bottom

margin-{left,right}: *value*

8.3	strong	margin-{left} = *value* **or**
	medium	margin-right = *value*
10.3.3	REMOVE	margin-left = margin-right
	padding-{top,right,bottom,left}: *value*	
8.4	strong	padding-{top,right,bottom,left} = *value*
	border-{top,right,bottom,left}-width: *value*	
8.5	strong	border-{top,right,bottom,left}-width = *value*
	{top, left, right, bottom}: right	
9.3.2	strong	{top, left, right, bottom} = *value*
	z-index: *value*	
9.9	strong	z-index = *value*
	width: *value*	
10.4	strong	width = *value*
	max-width: *value*	
10.4	req	width <= *value*
	min-width: *value*	
10.4	req	width >= *value*
	height: *value*	
10.5	strong	height = *value*
	max-height: *value*	
10.5	req	height <= *value*
	min-height: *value*	
10.5	req	width >= *value*

4 Conclusions and Future Work

The process of translating the CSS2 specification into constraints was very helpful, and revealed several ambiguities in the specification and sections where the behavior was underspecified, such as the behavior when `left` was defined to be *auto* but `width` and `right` were not.

These constraints could be made even more useful by by adding them to a constraint solver. This would create a simulation of the CSS2 specification instead of merely a formalization, and would be an easy way of checking for errors within the constraints listed in this document and for discrepancies in the original specification. This would be especially useful for future versions of Cascading Style Sheets, enabling the designers of future versions of the specification to easily test proposed modifications.

Some work needs to be done before this can happen. The constraints in this document currently rely on several external functions whose behaviors need to be specified before the constraints could actually be solved. In particular, this document does not address the packing of inline elements into line boxes, an issue that needs to be addressed in greater detail.

The two short-term additions to this document that would be the most helpful would be completing the constraints listed herein and adding constraints to describe tables. The CSS2 specification changed while this document was being created, and some sections, such as 10.3 and 10.6, were radically changed. As a result, the constraints describing these sections are incomplete. Section 17 of the CSS2 specification, the section describing tables, was also evolving rapidly, and as a result we did not even attempt to formalize the behavior of tables. This would be a worthwhile project, since the behavior of tables is very complicated and somewhat confusing.

The treatment of floats in this document is somewhat cumbersome, and there may be more efficient ways of structuring the constraints for floating boxes. This touches on the general problems of determining when constraints are the best formalism and when other formalisms should be used, and determining how to combine the different formalisms in simulation. While constraints have been quite helpful in describing the CSS2 specification in many sections, there are other sections where constraints do not seem appropriate.

Despite the inadequacies of this document, we believe that the constraints listed herein are a useful first step in modeling the behavior of Cascading Style Sheets Level 2, and that a small amount of additional work could provide those interested in Cascading Style Sheets with an easy mechanism for simulating changes to the specification. We hope that these constraints will prove useful to designers of future versions of Cascading Style Sheets, and to anyone who is interested in the issues inherent in Web page layout.

5 Acknowledgments

The author would like to thank Alan Borning at the University of Washington for providing information about constraint hierarchies and for helping to edit this paper. He would also like to thank Håkon Lie and Ian Jacobs at the World Wide Web Consortium for answering the author's questions about the Cascading Style Sheet Level 2 specification. This project has been funded in part by the National Science Foundation under Grant IRI-9302249.

References

[1] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, September 1992.

[2] Bert Bos, Håkon Lie, Chis Lilley, and Ian Jacobs. Cascading style sheets, level 2 specification. http://www.w3.org/TR/REC-CSS2/.