Computing Counterfactual Assumptions

William B. Starr

Dept. of Philosophy, Rutgers University
26 Nichol Ave.
New Brunswick, NJ 08904

Abstract

This paper discusses the implementation of Veltman (2005)'s update semantics for counterfactual conditionals in PLT Scheme.

1 Introduction

I will begin with one of the few interesting platitudes I've encountered in my life:

The Unification Claim More often than many realize, formal semanticists and computer scientists are doing the same thing.

To see why this is true, just consider what formal semanticists and computer scientists *do*. The formal semanticist engineers a representation language with a certain syntax. Then, using the recursive structure of that syntax, they define a recursive **interpretation function**, usually denoted by $[\![\cdot]\!]$, that assigns meanings to the formulae of their representation language, which — in the simplest case — are (Boolean) truth-values. Similarly, the computer scientist engineers a programming language with a certain syntax. Then, using the recursive structure of that syntax, they design a program, usually called an **interpreter**, that assigns meanings to programs written in their programming language, which — in the simplest cases — are (Boolean) truth-values.

The recognition of this common enterprise has led to exciting interdisciplinary exchanges and pushed the idea of **computational semantics** far beyond the

Email: wbstarr@rutgers.edu.
URL: http://eden.rutgers.edu/~wbstarr.

project of teaching computers to interpret natural language.[1] The possibility of this exchange generates a powerful reason for creating a computational model of a semantic theory: it represents the semantic theory in a format where connections to familiar concepts from computer science are salient, thus facilitating collaboration and exchange across the disciplines. But this is not the only reason for creating a computational model of a semantic theory. A computational model also provides a powerful tool for replicating a semanticist's results and testing their theory on new data. Furthermore, it takes the first step towards representing semantics in a way that could be linked to the other modules of language and communication, i.e. phonology, syntax and pragmatics, in a precise and powerful enough form that actual predictions about the multi-tiered process of communication could be formulated. With all of these reasons in mind, I conclude that creating a computational model of a semantic theory is an interesting and worthwhile project.

In this paper I will develop a computational model for Veltman (2005)'s **update semantics** for **counterfactual conditionals**.[2] [3] The implementation of this model will be carried out in **PLT Scheme**.[4] The following reasons suggest that Scheme is a good choice for this task:

Reasons to Use Scheme

- Its syntax is similar to that of the logics used for counterfactuals, making the relationship between the model and those logics more transparent.
- The mathematical concepts used in Veltman and others' semantics can be elegantly modeled using the Scheme's list data-structure and Scheme's ability to quantify and λ-abstract over lists.

In the process of doing so, I will explain Veltman's theory and replicate the results he reports (Veltman 2005: 169-170). Before I get started, I'll provide a brief roadmap, so the reader knows just what I am up to at each point of the process.

[1] See Barker (2001), Blackburn & Bos (2005), Shan (2005).

[2] Returning to the recent point of representing meaning in a way that facilitates incorporation into an integrated theory of communication, I am particularly interested in modeling Veltman's theory in a way that will facilitate the use of its output representations in models of probabilistic reasoning, causal inference and default reasoning.(Halpern 2003; Pearl 2000) This side of the project will have to be pursued and discussed at a later date.

[3] In what follows, I will assume familiarity with conditionals, counterfactuals and the general motivation for investigating them. For all of the necessary background on these topics see Starr (2007).

[4] For more on PLT Scheme see Felleisen *et al.* (2001) and the resources available at `http://www.plt-scheme.org/`. I will assume a basic familiarity with Scheme, along the lines of what one could acquire from taking CS503.

1.1 Outline

This paper approaches the task of implementing Veltman's theory incrementally. At each stage I will present a logical language and a Scheme program that models it. The first stage centers on (classical) **propositional logic**, whose semantics is likely to be familiar — does **truth-table** or **Boolean circuit** ring a bell? (§2). At the next stage, the goal will be to model **modal propositional logic**. (§3) This will amount to building an interpreter to do **possible worlds semantics**.[5] With this in hand, I will be able to model the even more fine-grained conception of meaning used in **update semantics**.(§4)[6] At the final stage, the tools developed in §3 and §4 are teamed up to build a model of the semantics for counterfactuals developed by Veltman.(§5) In closing, I will discuss what has been achieved and directions for future research.(§6)

2 Propositional Logic in Scheme

Classical Propositional Logic (=: CPL) is a tool that will be familiar to those that have either taken an elementary logic class, written a program or studied Boolean circuits. In this section, I will be building a Scheme program that models CPL . My explanation of this process will exploit the duality between formal semantics and computer science stated in my justification the Unification Claim. (§1) There were two parallels suggested there:

- A formal semanticist specifies the syntax of their representation language where a computer scientist specifies the syntax of their programming language.
- A formal semanticist defines a function $[\![\cdot]\!]$ from syntactically well-formed expressions of their language to a range of values where a computer scientist writes an interpreter that outputs a value given a syntactically well-formed expression of their programming language.

Accordingly, my discussion of the implementation of CPL in Scheme will begin by describing, in parallel, the syntax of CPL and the syntax of a programming language I will call **Classical Propositional Scheme** (=: CPL Scheme), which is just a particular subset of the regular Scheme expressions. (§2.1) Taking the next step, I will present the $[\![\cdot]\!]$ function alongside an interpreter for CPL Scheme. (§2.2) We will then see that our model's predictions neatly

[5] Reference

[6] See Veltman (1996) for an extensive introduction to the framework of update semantics.

fit those of CPL.

2.1 Syntax: CPL & CPL Scheme

Table 1: CPL Syntax

$\mathcal{E}x$	$\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots, \neg, \wedge, \vee, \rightarrow,), ($
$\mathcal{A}t$	$\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$
$\mathcal{W}ff$	$\mathsf{A} \in \mathcal{A}t \implies \mathsf{A} \in \mathcal{W}ff$
	$\mathsf{A} \in \mathcal{W}ff \implies (\neg\mathsf{A}) \in \mathcal{W}ff$
	$\mathsf{A}, \mathsf{B} \in \mathcal{W}ff \implies (\mathsf{A} \wedge \mathsf{B}) \in \mathcal{W}ff$
	$\mathsf{A}, \mathsf{B} \in \mathcal{W}ff \implies (\mathsf{A} \vee \mathsf{B}) \in \mathcal{W}ff$
	$\mathsf{A}, \mathsf{B} \in \mathcal{W}ff \implies (\mathsf{A} \rightarrow \mathsf{B}) \in \mathcal{W}ff$
	Nothing else is in $\mathcal{W}ff$

Table 2: Examples

$\mathcal{W}ff$	Not $\mathcal{W}ff$
p	(p)
$(\mathsf{p} \wedge \mathsf{q})$	$(\mathsf{q} \wedge \mathsf{p}$
$(\neg\mathsf{q})$	$\neg\mathsf{q}$
$((\mathsf{p} \wedge \mathsf{q}) \rightarrow (\mathsf{r} \vee \mathsf{q}))$	$\rightarrow \wedge\mathsf{p}($
$((\neg\mathsf{p}) \wedge (\mathsf{r} \rightarrow \mathsf{q}))$	$(\mathsf{q} \wedge \vee\mathsf{r})$
$((\mathsf{r} \wedge \mathsf{q}) \vee \mathsf{p})$	$\neg(\mathsf{r})$
$((\mathsf{r} \wedge \mathsf{r}) \rightarrow \mathsf{r})$	qq

Isn't there is a more succinct way of stating the syntax of CPL than Table 1? Indeed there is. It is known as **Backus Naur Form** (=: BNF). [7]

BNF for CPL

$$\mathcal{W}ff ::= \mathcal{A}t \mid (\neg\mathcal{W}ff) \mid (\mathcal{W}ff \wedge \mathcal{W}ff) \mid (\mathcal{W}ff \vee \mathcal{W}ff) \mid (\mathcal{W}ff \rightarrow \mathcal{W}ff)$$

The basic idea is to read '|' as 'or' and each term in-between as stating that combining any two $\mathcal{W}ff$ in the way pictured by the term is also a $\mathcal{W}ff$. In future sections we will rely on BNF notation, but for symmetry here we will display the parallel syntax of CPL Scheme in a tabular format:

Table 3: CPL Scheme Syntax

$\mathcal{E}x$	$0, 1, 2, \ldots, \texttt{not}, \texttt{and}, \texttt{or}, \texttt{then},), ($
$\mathcal{A}t$	$0, 1, 2, \ldots$
$\mathcal{W}ff$	$\texttt{A} \in \mathcal{A}t \implies \texttt{A} \in \mathcal{W}ff$
	$\texttt{A} \in \mathcal{W}ff \implies (\texttt{not A}) \in \mathcal{W}ff$
	$\texttt{A}, \texttt{B} \in \mathcal{W}ff \implies (\texttt{A and B}) \in \mathcal{W}ff$
	$\texttt{A}, \texttt{B} \in \mathcal{W}ff \implies (\texttt{A or B}) \in \mathcal{W}ff$
	$\texttt{A}, \texttt{B} \in \mathcal{W}ff \implies (\texttt{A then B}) \in \mathcal{W}ff$
	Nothing else is in $\mathcal{W}ff$

Table 4: Examples

$\mathcal{W}ff$	Not $\mathcal{W}ff$
`0`	`(0)`
`(0 and 1)`	`(1 and 0`
`(not 1)`	`(not 1`
`(0 and (1 or 2))`	`(0 and (1 or 2)`
`((not 0) then (2 or 1))`	`(0 and or)`
`((0 and 3) or (not 1))`	`not 0`
`(1 then (not 0))`	`11`

[7] For more on Backus Naur Form notation see http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html.

We can also succinctly state the syntax for CPL Scheme using BNF notation:

BNF for CPL Scheme
$\mathcal{W}ff ::= \mathcal{A}t \mid$ (`not` $\mathcal{W}ff$) | ($\mathcal{W}ff$ `and` $\mathcal{W}ff$) | ($\mathcal{W}ff$ `or` $\mathcal{W}ff$) | ($\mathcal{W}ff$ `then` $\mathcal{W}ff$)

The clear correspondence between CPL and CPL Scheme is illustrated by the following table:

Table 5: CPL - CPL Scheme Correspondence

CPL	CPL Scheme
p	`0`
$(p \wedge q)$	`(0 and 1)`
$((\neg p) \rightarrow (r \rightarrow p))$	`((not 0) then (2 then 1))`
$(r \wedge (r \rightarrow (p \wedge (r \vee (\neg q)))))$	`(2 and (2 then (0 and (2 or (not 1)))))`

2.2 Semantics: CPL & CPL Scheme

The semantics for CPL is built on the idea that the connectives $\neg, \wedge, \vee, \rightarrow$ are Boolean functions, i.e. functions from one or two binary values (**True** or **False**) to a single binary value (**True** or **False**). Below, the function each of these connectives is taken to express is summarized in the table for that connective.

Table 6: $f_{\wedge}(x, y)$

A	B	(A ∧ B)
T	T	T
T	F	F
F	T	F
F	F	F

Table 7: $f_{\vee}(x, y)$

A	B	(A ∨ B)
T	T	T
T	F	T
F	T	T
F	F	F

Table 8: $f_{\rightarrow}(x, y)$

A	B	(A → B)
T	T	T
T	F	F
F	T	T
F	F	T

Table 9: $f_{\neg}(x)$

A	(¬A)
T	F
F	T

Of course, when we talk about the *semantics of* CPL, we have in mind a function $[\![\cdot]\!]$ that maps the $\mathcal{W}ff$ of CPL to some set of values. So, how does settling on definitions of $f_{\wedge}(x, y)$, etc. further this project? It does so in two ways. First, it allows us to settle on which values $[\![\cdot]\!]$ will traffic in: the Booleans, (T, F). Second, those functions collectively show us how to reduce the question of which Boolean a complex formula denotes to the question of which Boolean(s) it's part(s) denote(s). For example, Table 6 shows us that $[\![(A \wedge B)]\!] = f_{\wedge}([\![A]\!], [\![B]\!])$. This is progress, but we aren't finished yet. This reduction trick is useless if we can just keep applying it. After all, we want $[\![\cdot]\!]$ to actually deliver a value at some point, not just amuse us endlessly with tricks.

So, when can't we apply the trick? Well, the trick works for conjunctions, disjunctions, negations and conditionals and, glancing back at the **syntax** of CPL (Table 1), there is only one kind of $\mathcal{W}ff$ left: atomic ones. This implies that we can only coherently ask after the value of $[\![\mathsf{A}]\!]$ once we have fixed the values of all the atomic formulae contained in A. This goal is achieved with **atomic valuations**.

Atomic Valuations An atomic valuation v is a (total) function from $\mathcal{A}t$ to $\{\mathrm{T}, \mathrm{F}\}$.

We shall henceforth represent the dependence of our interpretation function on a valuation by writing $[\![\cdot]\!]_v$. When we are finding the value of $[\![A]\!]_v$ v either tells us which row of the relevant table to look at — in the case that A is complex — or it tells us directly which Boolean value A should get.

We are now ready to define our interpretation function for CPL.

Definition 1 ($[\![\cdot]\!]_v$)

$$[\![\mathsf{A}]\!]_v = \begin{cases} v(\mathsf{A}) & \text{if } \mathsf{A} \in \mathcal{A}\mathsf{t} \quad (1) \\ f_\neg([\![\mathsf{B}]\!]_v) & \text{if } \mathsf{A} = (\neg\mathsf{B}) \quad (2) \\ f_\wedge([\![\mathsf{B}]\!]_v, [\![\mathsf{C}]\!]_v) & \text{if } \mathsf{A} = (\mathsf{B} \wedge \mathsf{C}) \quad (3) \\ f_\vee([\![\mathsf{B}]\!]_v, [\![\mathsf{C}]\!]_v) & \text{if } \mathsf{A} = (\mathsf{B} \vee \mathsf{C}) \quad (4) \\ f_\rightarrow([\![\mathsf{B}]\!]_v, [\![\mathsf{C}]\!]_v) & \text{if } \mathsf{A} = (\mathsf{B} \rightarrow \mathsf{C}) \quad (5) \end{cases}$$

(*Add some examples of how this definition works*)

Our next goal is to build a CPL Scheme interpreter. Conveniently, we can write this interpreter in Scheme itself.[8] As it turns out, the same basic strategy I used for writing Definition 1 can be used to write this interpreter. Recall that strategy:

- Use $f_\wedge(x, y)$-$f_\neg(x)$ to reduce the interpretation of complex formulae to the interpretation of their parts.
- Use v to interpret v when the first strategy reaches an atomic formula.

Stated as such, it almost sounds like a description of a process. Indeed, thinking of it in this way is helpful and motivates the following two questions:

- **Q1**: How should we represent $f_\wedge(x, y)$-$f_\neg(x)$ in Scheme in order to reduce

[8] I will return to this point and its implications.

the interpretation of a complex formula to the interpretation of its parts?
- **Q2**: How should we represent v in Scheme in order to interpret atomic $\mathcal{W}ff_{CPLS}$?

I will address **Q2** first.

2.2.1 Scheme Valuations

v is supposed to carry information about the truth-value of each atomic formula. Recall that the atomic formulae of CPL Scheme are the natural numerals `0,1,2,...`. One way of exploiting this choice is to represent v as a list of truth-values where the truth-value of an atomic formula n is stored at the nth position of v. For example, suppose our only atomic formulae are `0` and `1`, and `v` is the list `'(T F)`. The value at the 0th position of `v` is `T`, so `v` represents the valuation v where `0` is true. Given this innovation, we can define a Scheme function `(at-interp atom val)` which returns $\mathtt{V}_n$ when given `n` and `'(`$\mathtt{V}_0$`,...,` $\mathtt{V}_m$`)`, for $n \leq m$.

Listing 1: at-interp.scm

```
1 (define (at-interp atom val)
2   (list-ref val atom))
```

Table 10: at-interp.scm

Input	Output
`(at-interp 0 '(T F T))`	`T`
`(at-interp 1 '(T F T))`	`F`
`(at-interp 2 '(T F T))`	`T`

I now turn to answering **Q1**.

2.2.2 Boolean Functions in Scheme

It will be simple enough to model $f_{\neg}(x)$ in Scheme, so we will devote our attention to $f_{\wedge}(x, y) - f_{\rightarrow}(x, y)$ for remainder of this section. Let's take another look at Tables 6-9. Notice that the bolded column is unique for each connective, and think a little more about the information each row encodes and how it uniquely characterizes that connective.

A	B	(A ∧ B)
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

A	B	(A ∨ B)
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

A	B	(A → B)
T	T	**T**
T	F	**F**
F	T	**T**
F	F	**T**

The first row of the final column represents the value returned when both inputs are T; the second when the first is T and the second is F; the third when the first is F and the second T; the fourth when both inputs are F. We can think of this observation as a set of instructions telling us how to compute a Boolean function given this special column of values:

- If both inputs are T, look up the first value of the column and return it.
- If the first input is T and the second F, look up the second value of the column and return it.
- If the first input is F and the second T, look up the third value of the column and return it.
- If the first input is F and the second F, look up the fourth value of the column and return it.

This set of instructions can easily be transformed into a Scheme program. We can think of this special column as a list of values, so `and`'s list is `'(T F F F)`. We can then define a function `(abo op-list i1 i2)` that takes such a list, and two inputs, and returns a value in accordance with our instructions.

Listing 2: abo.scm

```
1 (define (abo op-list i1 i2)
2   (if (eq? i1 'T)
3       (if (eq? i2 'T)
4           (list-ref op-list 0)
5           (list-ref op-list 1))
6       (if (eq? i2 'T)
7           (list-ref op-list 2)
8           (list-ref op-list 3))))
```

Table 11: abo.scm

Input	Output
`(abo '(T F F F) T T)`	T
`(abo '(T F F F) F T)`	F
`(abo '(T F T T) T F)`	F
`(abo '(T T T F) F F)`	F
`(abo '(T F F F) T F)`	T

2.2.3 Putting it All Together: interp.scm

Now we have all the pieces, we just need to put them together. The goal is to write a function `(interp A v)` which takes a $\mathcal{W}ff_{\mathrm{CPLS}}$ and a Scheme valuation `v` and returns a value x such that $[\![\mathsf{A}]\!]_v = x$, where `A` is the CPL Scheme translation of A and `v` is the Scheme valuation corresponding to v. We will define `(interp A v)` piece-wise, depending on `A`'s form; just as we did for $[\![\cdot]\!]_v$ in Definition 1,

Listing 3: interp.scm

```
1  (define (interp A v)
2    (cond
3      [(list? A)
4       (cond
5         [(and (= (length A) 3)
6               (eq? (list-ref A 1) 'and))
7          (abo '(T F F F) (interp (car A) v) (interp (list-ref A 2) v))]
8         [(eq? (list-ref A 0) 'not)
9          (cond
10           [(eq? (interp (list-ref A 1) v) 'T) 'F]
11           [(eq? (interp (list-ref A 1) v) 'F) 'T])]
12        [(and (= (length A) 3) (eq? (list-ref A 1) 'or))
13         (abo '(T T T F) (interp (car A) v) (interp (list-ref A 2) v))]
14        [(and (= (length A) 3) (eq? (list-ref A 1) 'then))
15         (abo '(T F T T) (interp (car A) v) (interp (list-ref A 2) v))]
16        [else '(that's not a wff!)])]
17     [else
18      (list-ref v A)]))
```

Lines 5-7 mirror line (3) in Definition 1, and covers the interpretation of formulas like `(A and B)`. Lines 8-11 mirror line (2) of Definition 1, interpreting formulas like `(not A)`. Disjunction is handled by lines 12-14 and parallels line (4) of Definition 1. Similarly, lines 14-15 interpret formulas like `(A then B)` in the same manner as formulas like $(\mathsf{A} \rightarrow \mathsf{B})$ are treated in line (5) of Definition 1. Line 16 is there to cover the case where the program receives a 3-membered list as input but the input is not a formula. Finally, lines 17-18 tell us to process atomic formulae in the way outlined by at-interp.scm of §2.2.1 (see Listing 1), paralleling line (1) of Definition 1.

3 Possible Worlds Semantics in Scheme

It is time to admit something. Our detour through atomic valuations (§2.2) was neither optimal nor necessary.

Why not Optimal? Requiring a specific valuation v to compute $[\![\mathsf{A}]\!]_v$ and `(interp A v)` amounts to requiring an agent to have complete information about the atomic formulae in order to determine what they should think about the truth-value of any given formula. This requirement clashes with the fact that I can know that the faucet is not simultaneously running and not running without knowing whether or not the faucet is running. More specifically, I don't need to know whether or not `p` and `0` are true to know that $(\neg(\mathsf{p} \wedge (\neg\mathsf{p})))$ and `(not (0 and (not 0)))` are true. Furthermore, developing a semantics for counterfactuals in Boolean logic is hopeless. [9]

Why not Necessary? There is an alternative framework that can do everything the valuation-based strategy did while providing some key resources needed to overcome both problems that I just raised for the valuation-based approach. Why then the detour through the valuation-based approach? It is familiar and makes the transition to more sophisticated frameworks easier on our brains. We will now turn to our current poster-boy for semantic analysis: a specific breed of possible-worlds semantics called **Modal Propositional Logic** (=: MPL), and its Scheme cousin, **Modal Propositional Scheme** (=: MPL Scheme, MPLS).

3.1 Semantics: MPL & MPL Scheme

We have only very simple additions to the syntax, which does not merit an independent subsection:

BNF for MPL
$\mathcal{W}\mathit{ff}_{\mathsf{MPL}} ::= \mathcal{W}\mathit{ff}_{\mathsf{CPL}} \mid (\Box\mathcal{W}\mathit{ff}_{\mathsf{MPL}}) \mid (\Diamond\mathcal{W}\mathit{ff}_{\mathsf{MPL}})$

BNF for MPL Scheme
$\mathcal{W}\mathit{ff}_{\mathrm{MPLS}} ::= \mathcal{W}\mathit{ff}_{\mathrm{CPLS}} \mid (\texttt{nec}\ \ \mathcal{W}\mathit{ff}_{\mathrm{MPLS}}) \mid (\texttt{pos}\ \ \mathcal{W}\mathit{ff}_{\mathrm{MPLS}})$

Intuitively, $\Diamond\mathsf{A}$ means *it is logically possible that* A, while $\Box\mathsf{A}$ means *it is logically necessary that* A. In MPL formulae are taken to denote sets of **possible worlds**. While this sounds exotic, we've already encountered possible worlds, although we didn't use that name for them. Possible worlds are just the atomic valuations we saw back in §2.2, which are functions from $\mathcal{A}t$ to {T, F}. We

[9] See Starr (2007) for a detailed explanation of this latter point.

can now drop the v parameter from our interpretation function, making $[\![\cdot]\!]$ a function from $\mathcal{W}ff_{\mathsf{MPL}}$ into $\mathscr{P}(W)$. Of course, this means we are going to have to rethink $f_{\wedge(x,y)} - f_{\neg(x)}$. Fortunately, we can redefine these functions in terms of standard set-theoretic concepts. This is illustrated in the following definition

Definition 2 ($[\![\cdot]\!]$)

$$[\![\mathsf{A}]\!] = \begin{cases} \{w \mid w(\mathsf{A}) = \mathrm{T}\} & \text{if } \mathsf{A} \in \mathcal{A}\mathsf{t} \quad (1) \\ W - [\![\mathsf{B}]\!] & \text{if } \mathsf{A} = (\neg\mathsf{B}) \quad (2) \\ [\![\mathsf{B}]\!] \cap [\![\mathsf{C}]\!] & \text{if } \mathsf{A} = (\mathsf{B} \wedge \mathsf{C}) \quad (3) \\ [\![\mathsf{B}]\!] \cup [\![\mathsf{C}]\!] & \text{if } \mathsf{A} = (\mathsf{B} \vee \mathsf{C}) \quad (4) \\ (W - [\![\mathsf{B}]\!]) \cup [\![\mathsf{C}]\!] & \text{if } \mathsf{A} = (\mathsf{B} \rightarrow \mathsf{C}) \quad (5) \\ \{w \mid \exists w' \in [\![\mathsf{B}]\!]\} & \text{if } \mathsf{A} = (\Diamond\mathsf{B}) \quad (6) \\ \{w \mid [\![\mathsf{B}]\!] = W\} & \text{if } \mathsf{A} = (\Box\mathsf{B}) \quad (7) \end{cases}$$

Although MPL doesn't contain any new concepts, it does significantly enrich the expressive power of the language. Adding the $\Box$ and $\Diamond$ allows us to express logical truth and logical contingency *in the object language*, which is something that was not possible when our interpretation function mapped formulae and valuations to truth-values. Now that we have redefined meanings as sets of possible worlds, it is worth asking whether or not it still makes sense to talk about the truth of formulae. Indeed, there are two relevant concepts of truth that we can define within the possible worlds framework, namely truth-at-a-world and logical truth.

Definition 3 A is true at w, $w \models \mathsf{A} \iff w \in [\![\mathsf{A}]\!]$

Definition 4 A is logically true, $\models \mathsf{A} \iff \forall w \in W : w \in [\![\mathsf{A}]\!]$

Fact 1 $\exists w \in W : w \models (\Box\mathsf{A}) \iff \models \mathsf{A}$

Fact 1 follows directly from Definitions 3, 4 and 2.7. This framework provides us with a better model of how agents can use semantic reasoning to learn logical truths even if they don't know which valuation most accurately represents the actual world. They simply reason as follows. $[\![(\mathsf{A} \vee \neg\mathsf{A})]\!] = W$, so regardless of which world is the actual one, the actual world must be in $[\![(\mathsf{A} \vee \neg\mathsf{A})$, and by Definition 3 $(\mathsf{A} \vee \neg\mathsf{A})$ must be true at the actual world. However, this advance comes with a cost. It also follows that all logical truths express the same proposition, yet when I believe in the law of excluded middle I do not seem simultaneously believing in Church's Thesis. But, this is a matter to be explored elsewhere. We will now move to building an interpreter for MPL

Scheme.

Just as in §2.2.1, I are going to model valuations as lists of truth-values, only we are going to call them possible worlds now. To model sets of possible worlds, I will use lists of possible worlds. Although have more structure than simple sets, this choice will prove useful. What will our model of the set of all possible worlds for three atomic formulae look like?

$$W^3 = \texttt{'((T T T) (T T F) (T F T) (T F F) (F T T) (F F T) (F T F) (F F F))}$$

So, our new interpreter will need to deliver a list of lists of truth-values when given a formula. The trick is getting it to produce the *right* lists. To do this, we will also need to tell it what worlds are possible, i.e. W. We will also need to mimic the operations of set union, complementation and intersection with lists. Fortunately, someone has already defined these operations and they are included in SRFI 1, which comes standard in PLT Scheme. With these operations we have everything we need to write our interpreter, which is pictured in Listing 4 below.

Listing 4: w-interp.scm

```
(require (lib "1.ss" "srfi"))
(define (w-interp A W)
  (cond
    [(list? A)
     (cond
       [(and (= (length A) 2)
             (eq? (list-ref A 0) 'not))
        (lset-difference eq? W (w-interp (list-ref A 1) W))]
       [(and (= (length A) 3)
             (eq? (list-ref A 1) 'and))
        (lset-intersection eq?
                           (w-interp  (car A) W)
                           (w-interp (list-ref A 2) W))]
       [(and (= (length A) 3)
             (eq? (list-ref A 1) 'or))
        (lset-union eq?
                    (w-interp  (car A) W)
                    (w-interp (list-ref A 2) W))]
       [(and (= (length A) 3)
             (eq? (list-ref A 1) 'then))
        (lset-union eq?
                    (lset-difference eq? W
                                     (w-interp (list-ref A 0) W))
                    (w-interp (list-ref A 2) W))]
       [(and (= (length A) 2)
             (eq? (list-ref A 0) 'nec))
        (if (equal?
             (filter (lambda (x) (eq? (interp (list-ref A 1) x) 'T)) W)
             W)
            W
            '())]
       [(and (= (length A) 2)
             (eq? (list-ref A 0) 'pos))
        (if (empty?
             (filter (lambda (x) (eq? (interp (list-ref A 1) x) 'T)) W))
            '()
            W)]
       [else '(that was not a A!)])]
    [else (filter (lambda (x) (eq? (list-ref x A) 'T)) W)]))
```

The possible worlds framework presented here has been extended with the notion of relative similarity between worlds in order to provide semantic analyses for counterfactuals.[10] However, these approaches face serious difficulties that I will not have space to discuss here. Instead, I will explore an alternative approach developed in Veltman (2005) that has historical roots in Veltman (1976, 1996) and Kratzer (1981, 1989).

[10] Principally Stalnaker (1968) and Lewis (1973).

4 Update Semantics in Scheme

Veltman (1996, 2005)'s framework of update semantics makes use of semantic values like $[\![\mathsf{A}]\!]$ but understands the meaning of a formula A to be something more fine-grained, namely its **information update potential**. According to this view, knowing the meaning of A amounts to knowing the change A brings about in the cognitive state of anyone who wants to incorporate the information conveyed by it. Formally speaking, the meaning $[\mathsf{A}]$ of A is an operation on information states and $S[\mathsf{A}]$ denotes the result of applying $[\mathsf{A}]$ to the information state S, i.e. it is the result of updating S with A. In this section we will develop the formal details of this approach and implement it in Scheme. In 5 we will use these two tools to offer an analysis and implementation of counterfactuals along the lines pursued by Veltman.

4.1 Two Kinds of Update

Although its motivation will not be clear until we discuss counterfactuals, Veltman (2005: 165) models an information state S as a pair consisting of two sets of worlds U_S and F_S. $w \in U_S$ if each of the propositions that an agent in state S considers to be a **general laws** holds in w. $w \in F_S$ if, for all the agent in state S knows, w might be the actual world. This and a few other details are summarized in the following definition.

Definition 5

- (**Worlds**) $W^n = \{w \mid w : \mathcal{A}t \mapsto \{T, F\}\}$, where $\mid \mathcal{A}t \mid = n$ for finite n
- (**Situations**) $I^n = \{s \mid s \subseteq w \in W^n\}$, i.e. s is a partial function from $\mathcal{A}t$ to $\{T, F\}$
- (**States**) $S = \langle U_S, F_S \rangle$, where either (i) or (ii) holds:
  (i) $\varnothing \neq F_S \subseteq U_S \subseteq W^n$
  (ii) $F_S = U_S = \varnothing$

We can easily model each of these constructs in Scheme. We already have sets of worlds, but we don't have situations. Recall that a world is something like `'(T T F)`, assuming there are three atomic formulas. Situations are supposed to be partial versions of worlds, i.e. contain as much or less information about the atomic formulae. To do this I will model situations as follows:

Scheme Situations If $w =$ `'(`$\mathtt{v}_0$`,...,`$\mathtt{v}_n$`)` is a Scheme world, then the result of replacing i members of w with `*` is a **Scheme situation**, for $0 \leq i \leq n$.

I have not been able to write a Scheme program that generates all of the situations in any given world, but I have produced the following function that

produces all of the situations in a world of length 1, 2, or 3.

Listing 5: sit-in.scm

```
(define (sit-in w)
  (cond
    [(= (length w) 1)
     (list w)]
    [(= (length w) 2)
     (list (cons '* (list (list-ref w 1)))
           (cons (car w) '(*))
           w)]
    [(= (length w) 3)
     (list w
           (cons '* (cdr w))
           (cons '* (cons '* (cdr (cdr w))))
           (cons (car w) (cons '* (cdr (cdr w))))
           (cons (car w) (cons '* '(*)))
           (cons (car w) (cons (car (cdr w)) '(*)))
           (cons '* (cons (car (cdr w)) '(*))))]))
```

Although this is hardly a solution to the problem, it will suffice for practical purposes.

Implementing states in Scheme is a bit easier. We just take it to be a pair of lists of worlds. For example the state $\mathbf{1} = \langle W^2, W^2 \rangle$ is:

```
'(((T T) (T F) (F T) (F F)) ((T T) (T F) (F T) (F F)))
```

Now that we have precise picture of what information states are, we can be more precise about what it is to update an information state. We'll begin with our logical formalism first.

Definition 6 (Updates)

(1) a. $S[\mathsf{A}] = \langle U_S, F_S \cap [\![\mathsf{A}]\!] \rangle$, if $F_S \cap [\![\mathsf{A}]\!] \neq \varnothing$
b. $S[\mathsf{A}] = \langle \varnothing, \varnothing \rangle =: \mathbf{0}$, otherwise

(2) a. $S[\Box\mathsf{A}] = \langle U_S \cap [\![\mathsf{A}]\!], F_S \cap [\![\mathsf{A}]\!] \rangle$, if $F_S \cap [\![\mathsf{A}]\!] \neq \varnothing$
b. $S[\Box\mathsf{A}] = \mathbf{0}$, otherwise

As these conditions suggest, formulas with $\Box$ update what the agent believes about the general laws and by extension the actual world. In contrast, $\Box$-free formulae merely updates the worlds compatible with what the agent knows about the actual world. In general, these updates will communicate less, since they rule out possibilities in a more contingent way. Also note that $\Box$ does

not nest.

Modeling these conditions is Scheme turns out to be pretty easy. The Scheme program displayed in Listing 6 shows one way of doing it.

Listing 6: update.scm

```
(define (update S A W)
  (cond
    ; If non-law, update F only.
    [(or (not (list? A))
         (and (not (and (eq? (list-ref A 0) 'law)))
              (not (eq? (list-ref A 1) '>))))
     (if (not (equal? (lset-intersection equal?
                                         (car (cdr S))
                                         (w-interp A W)) ()))
         (list (car S) (lset-intersection equal?
                                          (car (cdr S))
                                          (w-interp A W)))
         (list () ()))]
    ; If law, update L & F.
    [(and (= (length A) 2)
          (eq? (list-ref A 0) 'law))
     (if (not (equal? (lset-intersection equal?
                                         (car (cdr S))
                                         (w-interp (car (cdr A)) W)) ()))
         (list (lset-intersection equal?
                                  (car S)
                                  (w-interp (car (cdr A)) W))
               (lset-intersection equal?
                                  (car (cdr S))
                                  (w-interp (car (cdr A)) W)))
         (list () ()))]))
```

5 Counterfactual Assumptions in Scheme

In this section, our goal will to be give a precise account of what state results from updating a state S with (if − hadA)

(*My current draft of this section is incredibly rough so I've chosen not to include it. The Scheme implementation is finished,* veltman.scm*, but connecting it to the formalism and motivating it piece-by-piece has proved to be a huge undertaking. I hope to send along a draft with a finished version of this section soon. But, I hope the finished program and this document are enough for the moment.*)

6 Conclusion

References

Barker, C. (2001). 'Introducing Continuation'. In R. Hastings, B. Jackson & Z. Zvolenszky (eds.) *Proceedings from Semantics and Linguistic Theory XI*, 20–35, Ithaca, NY: Cornell University.

Blackburn, P. & Bos, J. (2005). *Representation and Inference for Natural Language: A First Course in Computational Semantics*, vol. 1. Stanford, CA: CSLI Publications.
URL http://homepages.inf.ed.ac.uk/jbos/comsem/book1.html

Felleisen, M., Findler, R. B., Flatt, M. & Krishnamurthi, S. (2001). *How to Design Programs: An Introduction to Computing*. Cambridge, MA: MIT Press.
URL http://www.htdp.org/

Halpern, J. Y. (2003). *Reasoning about Uncertainty*. Cambridge, Massachusetts: The MIT Press, ISBN 0-262-08320-5.

Kratzer, A. (1981). 'Partition and Revision: The Semantics of Counterfactuals'. *Journal of Philosophical Logic*, **10**(2): 201–216.

Kratzer, A. (1989). 'An Investigation of the Lumps of Thought'. *Linguistics and Philosophy*, **12**(5): 607–653.

Lewis, D. K. (1973). *Counterfactuals*. Cambridge, Massachusetts: Harvard University Press.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge, England: Cambridge University Press, ISBN 0521773628 (hardback).

Shan, C. (2005). *Linguistic Side-Effects*. Ph.D. thesis, Harvard University, Department of Computer Science, Cambridge, MA.
URL http://www.cs.rutgers.edu/~ccshan/dissertation/book.pdf

Stalnaker, R. C. (1968). 'A Theory of Conditionals'. In N. Rescher (ed.) *Studies in Logical Theory*, 98–112, Oxford: Basil Blackwell Publishers.

Starr, W. B. (2007). 'Using Counterfactuals'. Slides for Presentation in Computer Science 504, Rutgers University Center for Cognitive Science.

Veltman, F. (1976). 'Prejudices, Presuppositions and the Theory of Counterfactuals'. In J. Groenendijck & M. Stokhof (eds.) *Amsterdam Papers in Formal Grammar*, Proceedings of the 1st Amsterdam Colloquium, 248–281, University of Amsterdam.

Veltman, F. (1996). 'Defaults in Update Semantics'. *Journal of Philosophical Logic*, **25**(3): 221–261.

Veltman, F. (2005). 'Making Counterfactual Assumptions'. *Journal of Semantics*, **22**: 159–180.
URL http://staff.science.uva.nl/~veltman/papers/FVeltman-mca.pdf