Using Propositional Logic to Model and Prove Conditionals
COMP 280, Spring 2000

Consider the following two programs for determining whether to hire a college graduate. How could we determine whether these two programs always produce the same answers?

```
(define (hire1? a-grad)
  (cond [(and (>= (gpa a-grad) 3.0)
              (top-college? a-grad)) true]
        [(and (< (gpa a-grad) 3.0)
              (large-donation? a-grad)) true]
        [(and (< (gpa a-grad) 3.0)
              (knows-Scheme? a-grad)) true]
        [else false]))

(define (hire2? a-grad)
  (or (and (< (gpa a-grad) 3.0)
           (cond [(knows-Scheme? a-grad) true]
                 [else (and (large-donation? a-grad)
                            (or (top-college? a-grad)
                                (large-donation? a-grad)))]))
      (and (>= (gpa a-grad) 3.0)
           (top-college? a-grad))))
```

First, we notice that the actual definitions of the helper functions are irrelevant. Therefore, we can replace the calls to the helper functions with boolean variables. Let's use the following correspondence between variables and expressions:

$A \equiv$ (>= (*gpa a-grad*) 3.0)
$B \equiv$ (*top-college? a-grad*)
$C \equiv$ (*large-donation? a-grad*)
$D \equiv$ (*knows-Scheme? a-grad*)

Using these variables, the programs would appear as follows:

```
(define (hire1? a-grad)
  (cond [(and A B) true]
        [(and (not A) C) true]
        [(and (not A) D) true]
        [else false]))

(define (hire2? a-grad)
  (or (and (not A)
           (cond [D true]
                 [else (and C (or B C))]))
      (and A B)))
```

We want to turn these into expressions in propositional logic (aka the boolean model that we discussed in class):

$$hire1? \equiv (A \wedge B) \vee (\neg(A \wedge B) \wedge ((\neg A \wedge C) \vee (\neg(\neg A \wedge C) \wedge (\neg A \wedge D))))$$

$$hire2? \equiv (\neg A \wedge (D \vee (\neg D \wedge (C \wedge (B \vee C))))) \vee (A \wedge B)$$

Now we wish to show that $hire1?$ and $hire2?$ are equivalent; that is, they agree on their value regardless of the values of A, B, C, and D. There are two ways to go about this:

1. Use equational reasoning with axioms
2. Enumerate all possible values of A, B, C, and D and check that the two expressions return the same value in each case.

We'll look at each in turn.

First, the proof by equational reasoning. We start by transforming $hire1?$. The underlined part of each expression is reduced using the rule specified in the subsequent line. See the text for the rules associated with each name.

$$\begin{array}{lll}
 & (A \wedge B) \vee (\neg(A \wedge B) \wedge (\underline{(\neg A \wedge C)} \vee (\underline{\neg(\neg A \wedge C)} \wedge (\neg A \wedge D)))) & \\
\equiv & \underline{(A \wedge B)} \vee (\underline{\neg(A \wedge B)} \wedge ((\neg A \wedge C) \vee (\neg A \wedge D))) & \text{(absorption)} \\
\equiv & \underline{(A \wedge B)} \vee (\underline{\neg A \wedge C}) \vee (\neg A \wedge D) & \text{(absorption)}
\end{array}$$

Next, we'll tranform $hire2?$ into the same expression. This will show that the two expressions are equivalent.

$$\begin{array}{lll}
 & (\neg A \wedge (\underline{D} \vee (\underline{\neg D} \wedge (C \wedge (B \vee C))))) \vee (A \wedge B) & \\
\equiv & (\neg A \wedge (D \vee (\underline{C} \wedge (B \vee \underline{C})))) \vee (A \wedge B) & \text{(absorption)} \\
\equiv & (\neg A \wedge \underline{\wedge} (D \underline{\vee} C)) \vee (A \wedge B) & \text{(absorption)} \\
\equiv & (\neg A \wedge D) \vee (\neg A \wedge C) \vee (A \wedge B) & \text{(distribution)}
\end{array}$$

We set up the truth table proof as follows. To complete it, fill in the values of $hire1?$ and $hire2?$ in each column (see pages 2-4 in the text for details if you haven't used these before):

A	B	C	D	$hire1?$	$hire2?$
false	false	false	false		
false	false	false	true		
false	false	true	false		
false	false	true	true		
false	true	false	false		
false	true	false	true		
false	true	true	false		
false	true	true	true		
true	false	false	false		
true	false	false	true		
true	false	true	false		
true	false	true	true		
true	true	false	false		
true	true	false	true		
true	true	true	false		
true	true	true	true		

Truth tables are boring and tedious. Can we write a program to check whether two propositional logic formulas are equivalent? Yes. Here's the skeleton of the program we wrote in class:

A formula is one of
- 'true
- 'false
- a symbol
- (*make-and formula formula*)
- (*make-or formula formula*)
- (*make-not formula*)

```
;; check-equiv-vars : formula formula vars → boolean
;; returns true iff the two formulas agree on their values for every
;; combination of values on the variables in vars. We assume that
;; vars contains all variables appearing in either input formula.
(define (check-equiv-vars F1 F2 vars)
  (cond [(empty? vars) (equal? (eval F1) (eval F2))]
        [else (and (check-equiv-vars (subst 'true (first vars) F1)
                                     (subst 'true (first vars) F2)
                                     (rest vars))
                   (check-equiv-vars (subst 'false (first vars) F1)
                                     (subst 'false (first vars) F2)
                                     (rest vars)))]))

;; eval : formula → boolean
;; reduces a formula containing only true and false to a boolean value

;; subst : boolsym sym formula → formula
;; replaces every occurrence of sym in formula with boolsym
```