7 Propositional Calculus

$\boxed{7.1}$ Boolean Algebra

Values: **t**, **f** denote "true", "false"

Operations: $\wedge$, $\vee$, $\neg$ denote "AND", "OR", "NOT"

Truth Tables:

p	q	$p \vee q$	$p \wedge q$	$\neg p$
f	**f**	**f**	**f**	**t**
f	**t**	**t**	**f**	
t	**f**	**t**	**f**	**f**
t	**t**	**t**	**t**	

E.g.: $(\mathbf{t} \wedge \mathbf{f}) \Rightarrow \mathbf{f}$ $\quad (\neg \mathbf{t}) \Rightarrow \mathbf{f}$

$(\mathbf{t} \vee (\mathbf{t} \wedge (\neg \mathbf{t}))) \Rightarrow (\mathbf{t} \vee (\mathbf{t} \wedge \mathbf{f})) \Rightarrow (\mathbf{t} \vee \mathbf{f}) \Rightarrow \mathbf{t}$

$\boxed{7.2}$ Propositional Calculus

variable ::= $x \mid y \mid z \mid x_1 \mid y_1 \mid z_1 \mid \ldots$
constant ::= **t** | **f**
well-formed-formula ::= [wff]
constant | variable
| (well-formed-formula $\vee$ well-formed-formula)
| (well-formed-formula $\wedge$ well-formed-formula)
| ($\neg$ well-formed-formula)

E.g.: $(x\wedge y)$ $(x\wedge(y\wedge z))$ $((x\wedge y)\wedge z)$
$(x\wedge(y\vee(\neg(x_1\vee x_3))))$ $((\mathbf{t}\vee(x\wedge\mathbf{f}))\vee(y\wedge(\neg x)))$

Variables denote “propositions” like “Bob ran Sparky this morning” that take the values **t** and **f**.

$\boxed{7.3}$ Equivalence of Well Formed Formulae

Let w_1, $w_2 \in$ well-formed-formula:

Definition: An "assignment" is a mapping from variables to constants: e.g. $x_1 \mapsto \mathbf{t}$, $x_2 \mapsto \mathbf{f}$, $y \mapsto \mathbf{f}$, . . .

Definition: $w_1 \equiv w_2$ means that for any assignment w_1 and w_2 have the same value as boolean algebra expressions after replacing each variable by its assigned constant value.

Examples: $(x \vee y) \equiv (y \vee x)$ $\qquad (\neg(\neg x_1)) \equiv x_1$

I.e. $w_1 \equiv w_2$ means w_1 and w_2 have the same truth table.

$\boxed{7.4}$ Example of Equivalence

$(x \vee (y \wedge z)) \;\equiv\; ((x \vee y) \wedge (x \vee z))$ as we see from:

x	y	z	$(y \wedge z)$	$(x \vee (y \wedge z))$	$(x \vee y)$	$(x \vee z)$	$((x \vee y) \wedge (x \vee z))$
f	**f**	**f**	**f**	**f**	**f**	**f**	**f**
f	**f**	**t**	**f**	**f**	**f**	**t**	**f**
f	**t**	**f**	**f**	**f**	**t**	**f**	**f**
f	**t**	**t**	**t**	**t**	**t**	**t**	**t**
t	**f**	**f**	**f**	**t**	**t**	**t**	**t**
t	**f**	**t**	**f**	**t**	**t**	**t**	**t**
t	**t**	**f**	**f**	**t**	**t**	**t**	**t**
t	**t**	**t**	**t**	**t**	**t**	**t**	**t**

$\boxed{7.5}$ Elementary Equivalences

$(x \wedge y) \equiv (y \wedge x)$		[Commutativity]
$(x \wedge (y \wedge z)) \equiv ((x \wedge y) \wedge z)$		[Associativity]
$(x \vee y) \equiv (y \vee x)$		[Commutativity]
$(x \vee (y \vee z)) \equiv ((x \vee y) \vee z)$		[Associativity]
$(x \wedge x) \equiv x$	$(x \vee x) \equiv x$	[Idempotency]
$(\neg(\neg x)) \equiv x$		[Double Negation]
$(\mathbf{t} \vee x) \equiv \mathbf{t}$	$(\mathbf{f} \wedge x) \equiv \mathbf{f}$	[Nullity]
$(\mathbf{f} \vee x) \equiv x$	$(\mathbf{t} \wedge x) \equiv x$	[Units]
$(x \vee (\neg x)) \equiv \mathbf{t}$	$(x \wedge (\neg x)) \equiv \mathbf{f}$	[Excluded Middle]

$\boxed{7.6}$ Advanced Equivalences and Formal Algebra

$(x \wedge (y \vee z)) \;\equiv\; ((x \wedge y) \vee (x \wedge z))$	[Distributive Law]
$(x \vee (y \wedge z)) \;\equiv\; ((x \vee y) \wedge (x \vee z))$	[Distributive Law]
$(\neg (x \wedge y)) \;\equiv\; ((\neg x) \vee (\neg y))$	[DeMorgan's Law]
$(\neg (x \vee y)) \;\equiv\; ((\neg x) \wedge (\neg y))$	[DeMorgan's Law]

Formal Algebra of Propositional Calculus with Constants:

Grammar is just the grammar of wff with constants;

Axioms are just the above equivalences with

$\equiv$ replace by $=$

7.7 Constant Eliminating Rewrite Rules

$(\mathbf{t} \vee x) \implies \mathbf{t}$	$(\mathbf{f} \wedge x) \implies \mathbf{f}$	[Zeros]
$(\mathbf{f} \vee x) \implies x$	$(\mathbf{t} \wedge x) \implies x$	[Units]
$(\neg \mathbf{f}) \implies \mathbf{t}$	$(\neg \mathbf{t}) \implies \mathbf{f}$	[Not]

Example:

$$(\mathbf{t} \wedge (x \vee (\neg \mathbf{t}))) \implies (x \vee (\neg \mathbf{t})) \implies (x \vee \mathbf{f}) \implies x$$

All wff with constants can be rewritten to an equivalent wff that either has no constants or is one of the two trivial wff containing nothing but a constant (**f** or **t**)

$\boxed{7.8}$ Theorems and Satisfiability

Let $w \in$ wff:

Definition: w is a theorem if and only if $w \equiv \mathbf{t}$.

Definition: w is unsatisfiable if and only if $w \equiv \mathbf{f}$.

Lemma: w is a theorem if and only if $(\neg w)$ is unsatisfiable.

Automatic Theorem Proving, Method 1:

Make a truth table for w.

Drawback: If there are 100 variables there are $2^{100} \approx 10^{30}$ rows in the truth table.

$\boxed{7.9}$ Disjunctive Normal Form

Disjunctive Normal Form (DNF):

All $\vee$'s are outside all $\wedge$'s and $\neg$'s.

All $\neg$'s are inside all $\wedge$'s and $\vee$'s and are not doubled.

literal ::= variable | ($\neg$ variable)

dnf-clause ::= literal | (dnf-clause $\wedge$ dnf-clause)

dnf ::= dnf-clause | (dnf $\vee$ dnf)

E.g. $(\neg(p\vee(q\wedge(\neg p)))) \Rightarrow ((\neg p)\wedge(\neg(q\wedge(\neg p))))$

$\Rightarrow ((\neg p)\wedge((\neg q)\vee(\neg(\neg p)))) \Rightarrow ((\neg p)\wedge((\neg q)\vee p))$

$\Rightarrow (((\neg p)\wedge(\neg q))\vee((\neg p)\wedge p))$

$\boxed{7.10}$ DNF Automatic Theorem Proving

Automatic Theorem Proving Method 2 (DNFs):

(1) Rewrite $(\neg w)$ to DNF using DeMorgan's Laws, Double Negation, and Distributive Law.

(2) Check that <u>all</u> the dnf-clauses are unsatisfiable.
If yes, $(\neg w)$ is unsatisfiable and w is a theorem.
If no, w is not a theorem.

  (2a) A dnf-clause if unsatisfiable if and only if it contains both v and $(\neg v)$ for some variable v.

E.g. In $(((\neg p)\wedge p)\vee((\neg q)\wedge p))$ the first clause is unsatisfiable but the second clause can be satisfied by the assignment $q \mapsto \mathbf{f}$, $p \mapsto \mathbf{t}$.

$\boxed{7.11}$ DNF and CNF Rewrite Rules

DNF Rewrite Rules:

$(x \wedge (y \vee z)) \Longrightarrow ((x \wedge y) \vee (x \wedge z))$	[Distributive Law]
$(\neg(x \wedge y)) \Longrightarrow ((\neg x) \vee (\neg y))$	[DeMorgan's Law]
$(\neg(x \vee y)) \Longrightarrow ((\neg x) \wedge (\neg y))$	[DeMorgan's Law]
$(\neg(\neg x)) \Longrightarrow x$	[Double Negation]

CNF is the same as DNF with $\vee$ and $\wedge$ interchanged

CNF Rewrite Rules:

$(x \vee (y \wedge z)) \Longrightarrow ((x \vee y) \wedge (x \vee z))$	[Distributive Law]
$(\neg(x \wedge y)) \Longrightarrow ((\neg x) \vee (\neg y))$	[DeMorgan's Law]
$(\neg(x \vee y)) \Longrightarrow ((\neg x) \wedge (\neg y))$	[DeMorgan's Law]
$(\neg(\neg x)) \Longrightarrow x$	[Double Negation]

7.12 Propositional Calculus in LISP

constant ::= `T` | `F`

variable ::= symbol except constant or `NIL`

wff ::= constant | variable

  | (`NOT` wff) | (`AND` wff wff) | (`OR` wff wff)

Definition: `(IMPLIES P Q)` $\equiv$ `(OR (NOT P) Q)`

assignment ::= () | ((variable wff) . assignment)

Example Functions:

```
(ASSIGN '((X T) (Y (OR P Q))) '(AND X (NOT Y)))
    ==> '(AND T (NOT (OR P Q)))
(EVALUATE '(AND T (NOT F))) ==> 'T
```

7.13 Propositional Calculus LISP Functions

`(EVALUATE 'e)`	Evaluate `'e` as per Boolean Algebra.
`(WFF? 'e)`	Return true iff `'e` is a wff.
`(ASSIGN 'a 'e)`	Substitute using assignment `'a` into wff `'e`.
`(DNF 'e)`	Return DNF of `'e`.
`(LITERALS 'e)`	Return list of literals in `'e`.
`(VARIABLES 'e)`	Return list of variables in `'e`.
`(THEOREM? 'e)`	Return true if `'e` is a theorem. (But what method should be use?)

Comments:

1. Assignments have the form of a list of sublists with the first element of each sublist being a key which is to be looked up. Such a data structure is common in LISP, and is called an association list. There is a LISP function, `ASSOC`, designed to look up a sublist beginning with an atom, and return that sublist, or return `NIL` if there is no such sublist. Formally:

   ```
   (ASSOC 'x '()) ⟹ NIL
   (ASSOC 'x '((y . z) . w)) ⟹ '(y . z)        if (EQL 'x 'y)
                             ⟹ (ASSOC 'x 'w)   otherwise
   ```

   where `x`, `y`, `z`, `w` $\in$ s-expression

2. Note that we would prefer the name `SUBSTITUTE` for the `ASSIGN` function, but the there is already a function named `SUBSTITUTE` in COMMONLISP.