CS1112
Logical Equivalence & Satisfiability

Lecturer:

Professor Barry O'Sullivan

Office: 2-65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

http://osullivan.ucc.ie/teaching/cs1112/

Logical Equivalence and Satisfiability

Logical Equivalences

De Morgan's Laws

Tautologies, Contradictions and Satisfiability

Understanding Specifications

Your software must ensure at least one of these conditions:
(i)whenever the user is logged out, their balance is hidden
(ii)the balance cannot be visible at the same time as a transaction is being processed

p: the user is logged out
q: the user's balance is hidden
s: a transaction is being processed

$$((p \rightarrow q) \vee (\neg(\neg q \wedge s)))$$

1. How do we work out whether or not some given software satisfies the specification?
2. Can we work out the truth value for all possible situations?
3. If we write it as
   $p \rightarrow q \vee \neg\neg q \wedge s$
   does it make a difference?
4. What about
   $\neg(p \wedge s) \vee q$?

Equivalent Logical Formulae

Two formulas in propositional logic are equivalent if and only if they have the same truth functions.

i.e. no matter what values we assign to the atomic propositions, the two formulas have the same truth value.

If *s* and *t* are equivalent, we write $s \equiv t$

Example: $p \rightarrow q \equiv \neg p \vee q$

p	q	p→q	¬p	q	¬p∨q
T	T	T	F	T	T
T	F	F	F	F	F
F	T	T	T	T	T
F	F	T	T	F	T

the two statements have the same truth value for each assignment to *p* and *q*

De Morgan's Laws (I)

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

p	q	p∨q	¬(p∨q)	¬p	¬q	¬p∧¬q
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

Note:

≡ is not a connective. It is not part of the language of propositional logic.

De Morgan's Laws (II)

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

p	q	p∧q	¬(p∧q)	¬p	¬q	¬p∨¬q
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Conditional vs Disjunction

$$p \rightarrow q \equiv \neg p \vee q$$

p	q	p→q	¬p	¬p∨q
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Distributive Law (I)

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

p	q	r	q∧r	p∨(q∧r)	(p∨q)	(p∨r)	(p∨q)∧(p∨r)
T	T	T	T	T	T	T	T
T	T	F	F	T	T	T	T
T	F	T	F	T	T	T	T
T	F	F	F	T	T	T	T
F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	F
F	F	T	F	F	F	T	F
F	F	F	F	F	F	F	F

Distributive Law (II)

$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

p	q	r	q∨r	p∧(q∨r)	(p∧q)	(p∧r)	(p∧q)∨(p∧r)
T	T	T	T	T	T	T	T
T	T	F	T	T	T	F	T
T	F	T	T	T	F	T	T
T	F	F	F	F	F	F	F
F	T	T	T	F	F	F	F
F	T	F	T	F	F	F	F
F	F	T	T	F	F	F	F
F	F	F	T	F	F	F	F

More Logical Equivalences

$p \wedge T \equiv p$
$p \vee F \equiv p$

$p \wedge F \equiv F$
$p \vee T \equiv T$

$p \vee p \equiv p$
$p \wedge p \equiv p$

$\neg(\neg p) \equiv p$

$p \vee \neg p \equiv T$
$p \wedge \neg p \equiv F$

$p \vee q \equiv q \vee p$
$p \wedge q \equiv q \wedge p$

$p \vee (q \vee r) \equiv (p \vee q) \vee r$
$p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$

$p \vee (p \wedge q) \equiv p$
$p \wedge (p \vee q) \equiv p$

Exercise: verify these by creating truth tables

Set Equivalences

$S \cap U = S$
$S \cup \{\} = S$

$S \cap \{\} = \{\}$
$S \cup U = U$

$S \cup S = S$
$S \cap S = S$

$(S')' = S$

$S \cup S' = U$
$S \cap S' = \{\}$

$S \cup R = R \cup S$
$S \cap R = R \cap S$

$S \cup (R \cup W) = (S \cup R) \cup W$
$S \cap (R \cap W) = (S \cap R) \cap W$

$S \cup (S \cap R) = S$
$S \cap (S \cup R) = S$

$(S \cup R)' = S' \cap R'$
$(S \cap R)' = S' \cup R'$

Tautologies and Contradictions

A propositional formula that is always true, no matter what truth values are given to the atomic propositions,
is a tautology.

A propositional formula that is always false is a contradiction.

p	$\neg p$	$p \vee \neg p$
T	F	T
F	T	T

p	$\neg p$	$p \wedge \neg p$
T	F	F
F	T	F

p	q	$p \wedge q$	$(p \wedge q) \rightarrow p$
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	T

tautology

contradiction

tautology

Example: $(p \wedge q) \rightarrow (p \vee q)$ is a tautology

p	q	$p \wedge q$	$(p \vee q)$	$(p \wedge q) \rightarrow (p \vee q)$
T	T	T	T	T
T	F	F	T	T
F	T	F	T	T
F	F	F	F	T

Example: $(p \wedge q) \rightarrow (p \vee q)$ is a tautology

We can show the same result without truth tables, by applying known equivalences

$(p \wedge q) \rightarrow (p \vee q)$	$\equiv$	$\neg(p \wedge q) \vee (p \vee q)$	*using* $X \rightarrow Y \equiv \neg X \vee Y$
	$\equiv$	$\neg p \vee \neg q \vee (p \vee q)$	*using de Morgan's law*
	$\equiv$	$\neg p \vee \neg q \vee p \vee q$	$\vee$ *is associative*
	$\equiv$	$\neg p \vee p \vee \neg q \vee q$	$\vee$ *is commutative*
	$\equiv$	$(\neg p \vee p) \vee (\neg q \vee q)$	$\vee$ *is associative*
	$\equiv$	$T \quad \vee \quad T$	*basic equivalence*
	$\equiv$	T	*basic equivalence*

Is $(\neg(p \wedge s) \vee q)$ equivalent to $((p \rightarrow q) \vee (\neg(\neg q \wedge s)))$?

Satisfiability

A propositional formula for which there is at least one assignment of truth values to the atomic propositions that results in a true formula is satisfiable.

Example

$(p \wedge (q \vee r)) \rightarrow \neg p$ is a satisfiable formula

p	q	r	$q \vee r$	$p \wedge (q \vee r)$	$\neg p$	$(p \wedge (q \vee r)) \rightarrow \neg p$
T	T	T	T	T	F	F
T	T	F	T	T	F	F
T	F	T	T	T	F	F
T	F	F	F	F	F	T
F	T	T	T	F	T	T
F	T	F	T	F	T	T
F	F	T	T	F	T	T
F	F	F	F	F	T	T

Notation: $\models$

Often, it will be important to know exactly which assignments of truth values make a complex formula true.

At other times, we will want to know what complex formulae are made true by a particular set of truth values.

We use some special notation to indicate this, using the symbol $\models$ to indicate that whenever the statements on the left are true, then the formula on the right is true.

E.g. $p, q \models p \wedge q$

$\neg p \models \neg p \vee q$

$\neg p, q, r \models (p \wedge (q \vee r)) \rightarrow \neg p$

p	q	r	$q \vee r$	$p \wedge (q \vee r)$	$\neg p$	$(p \wedge (q \vee r)) \rightarrow \neg p$
T	T	T	T	T	F	F
T	T	F	T	T	F	F
T	F	T	T	T	F	F
T	F	F	F	F	F	T
F	T	T	T	F	T	T
F	T	F	T	F	T	T
F	F	T	T	F	T	T
F	F	F	F	F	T	T

$$p, \neg q, \neg r \models (p \wedge (q \vee r)) \rightarrow \neg p$$

$$\neg p, q, r \models (p \wedge (q \vee r)) \rightarrow \neg p$$

Boolean satisfiability

- Electronic design automation, FPGA design, hardware verification, etc., are all expressed as *satisfiability* problems
- The design is formalised as a logical formula, built from many atomic propositions
- The task is to find an assignment of truth values to the atomic propositions such that the larger formula is true, or to identify that no assignment is possible
- Formulae are expressed in conjunctive normal form – the conjunction of a set of clauses, where each clause is a disjunction of positive or negative propositions:

- E.g. $(p \vee \neg q \vee r) \wedge (q \vee \neg m) \wedge (\neg p \vee k \vee \neg x) \wedge (m \vee r) \wedge \ldots$

- Current programs must solve problems with *millions* of clauses ...

Next lecture ...

How to construct arguments