Student Scribe for ECS 20, Lecture 3 – October 2, 2008 – Alex Chan-Kai

Today: o Sentential Logic (continued)

Review:

- Definition of a well formed formula (for sentential logic, over same set of proposition symbols ***P***) is:

1. **0** and **1** are WFF (formal symbols, not truth values)
2. every *P* in ***P*** is a WFF
3. If α and β are WFF, then so are (α ^ β), (α v β), (¬ α), (α → β), and (α <-> β) .

Let ***L*** be the language of WFFs (over some understood set of proposition symbols ***P***)

- "Order of Precedence"

¬ binds most tightly
^
v
->
<-> binds least tightly
Parenthesis can change the default order. Within a given precedence, the usual convention is that things group right-to-left.

We can associate any WFF to a **tree** where the leaves are the proposition symbols and constants 0 and 1 and the **internal node**s are marked with **logical operators**. Given a **truth assignment**, which was described last time, you can propagate up truth value to every node. We showed how to do this an example. Now we'd like to get more formal with what is going on when we do this. We could get more formal using the *language* of trees, but it is simpler to do so **recursively**, which is what we turn towards now.

Giving semantics to WFFs:

- Recall that a **truth assignment** is a function *t*: ***P*** → {0,1} (from the predicate symbols to binary truth values).
- **Extending a truth assignment**: define from *t* a function *t**: *L* → {0,1} defined by: (Prof. Rogaway used a *t* with a bar over it, but I'll just use a *t** because it's easier to type).

1. *t**(**0**) = 0 *t**(1) = 1
2. *t**(*P*) = *t* (*P*) for all P in ***P***.
3. *t**((α ^ β)) = 1 if *t**(α)= *t** (β)=1, and 0 otherwise

$t^*((\alpha \text{ v } \beta)) = 1$ if $t^*(\alpha) = 1$ or $t^*(\beta)=1$, and 0 otherwise
$t^*((\neg\, \alpha)) = 1$ if $t^*(\alpha) = 0$, and 0 otherwise
$t^*((\alpha \rightarrow \beta)) = 0$ if $t^*(\alpha) = 1$ and $t^*(\beta)=0$, and 1 otherwise
$t^*((\alpha <\!\!-\!\!> \beta)) = 1$ if $t^*(\alpha) = \beta$, and 0 otherwise

Some practice designing formulas …. and *circuits*, too

Exercise 1 Who won the fight?

Two fighters, A and B. Three judges, each votes "0" if he thinks A won and "1" if he things B won. We want to create a Boolean formula that computes who won, according to *majority vote*.

Majority(P,Q, R) = 1 iff at least two of P,Q,R are 1, and 0 otherwise.

First write out a truth table for what you want:

P	Q	R	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Disjunctive normal form (DNF) – The formula is the OR of terms, and each term is the AND of variables or their complements. We can take ANY truth table and "read it out" as DNF. Above, we get

$$(\neg P)QR \text{ v } P(\neg Q)R \text{ v } PQ(\neg R) \text{ v } (PQR)$$

We can simplify this a bit by "factoring out" QR from the first and forth disjunct giving a term $(\neg P \text{ v } P)QR = 1QR=QR$:

$$QR \text{ v } P(\neg Q)R \text{ v } PQ(\neg R)$$

In class Prof. Rogaway then drew it as a circuit, using AND, OR, and NOT games, which I show in the notes below.

Exercise 2: Find WFF that is 1 if *exactly one* of A, B, C, and D, are true.

For exactly one of these variables to be true, we need that *at least one* of the variables is true and *at most one* of the variables is true. The first is easy to translate into sentential logic:

$$A \vee B \vee C \vee D$$

The second is a little trickier: we want to say if A is true, for example than B must be false, that is, $A \rightarrow \neg B$; and $A \rightarrow \neg C$, and so forth:

$(A \rightarrow \neg B)$ $(A \rightarrow \neg C)$ $(A \rightarrow \neg D)$ $(B \rightarrow \neg A)$ $(B \rightarrow \neg C)$ $(B \rightarrow \neg D)$ $(C \rightarrow \neg A)$ $(C \rightarrow \neg B)$ $(C \rightarrow \neg D)$ $(D \rightarrow \neg A)$ $(D \rightarrow \neg B)$ $(D \rightarrow \neg C)$.

So and this formula and the prior one and you are done. More generally, to say that exactly one of $X_1, \ldots, X_n$ are true, we could use a formula of the form

$$\left(\bigvee_i X_i \right) \wedge \bigwedge_{i \neq j} (X_i \rightarrow \neg X_j)$$

Logical completeness. By virtue of the "DNF algorithm" that turns any **truth table** to a **WFF** that corresponds to the functionality, we know that any Boolean formula—or any truth table—can be represented using only the logical connectives $\{\wedge, \vee, \neg\}$. We say that this set of connectives are **logically complete**.

In fact, we can use a smaller set of connectives, eliminating **or** by using the identity:

$$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

which is known as DeMorgan's law. We could, alternatively, eliminate **and** by using the DeMorgan law:

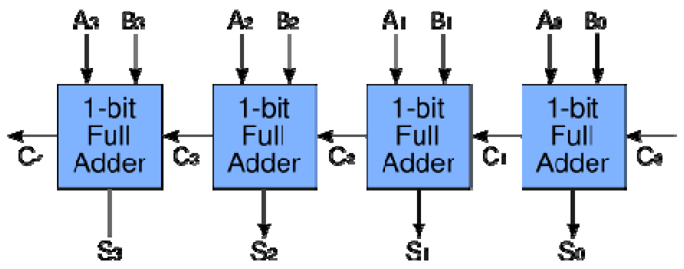
$$P \wedge Q \equiv \neg(\neg P \vee \neg Q).$$

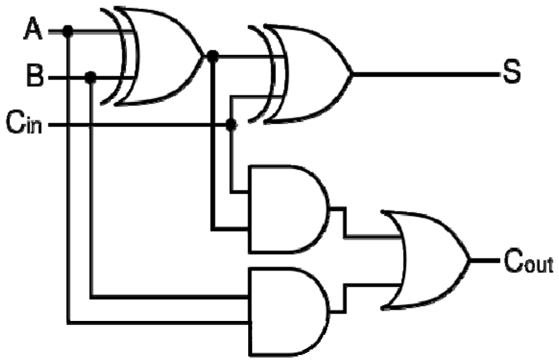
So we have show that

- $\wedge$ & $\neg$ are logically complete,
- v & $\neg$ are logically complete. In addition,
- NAND, all by itself, is logically complete, because we can rewrite $\wedge$ and $\neg$ using NAND (tie the inputs of the NAND together to make an inverter) and
- NOR, all by itself, is logically complete, because we can rewrite v and $\neg$ using NOR (tie the inputs of NOR together to make an inverter).

Exercise 3: Find a circuit to add up two 8-bit binary numbers.

Here is a diagram for how to use a **full adder**, and then a circuit for the adder. (In class we first wrote out the truth table for the full adder.)





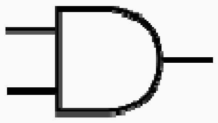
Tautologies, satisfiability, and logical equivalence

Definitions:

Given WFF's α and β,

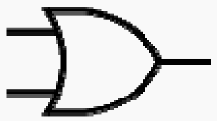
- α is a **tautology** if *t*(α) = 1 for all truth assignments *t*
- α is **satisfiable** if there exists a truth assignment *t* s.t. *t*(α)=1
- α is a **contradiction** if *t*(α) = 0 for all truth assignments *t*.
- α and β are (logically) **equivalent** if *t*(α)=*t*(α) for all truth assignments *t*.

Gates Diagrams:

Type	**Distinctive shape**	**Truth table**
AND		

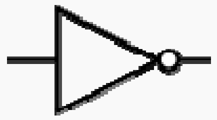
INPUT		**OUTPUT**
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR



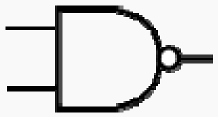
INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOT



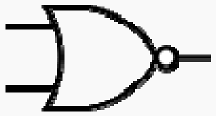
INPUT	OUTPUT
A	NOT A
0	1
1	0

NAND



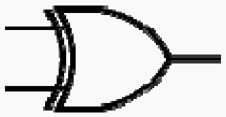
INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR



INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

XOR



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0