CS 512 Formal Methods, Spring 2017 — Instructor: Assaf Kfoury

Quantified Boolean formulas

February 23, 2017 — Scribe: Benjamin Lawson

(These lecture notes are **not** proofread and proof-checked by the instructor.)

1 Administrative notes

1. Assignment #5 is posted
2. Mid-term will be posted on Thursday, March 2nd around 5pm and is due on Friday March 3rd by 11:59pm
3. Progress reports on term projects is delayed until the first week after spring break.

2 Proof rules for quantification

2.1 See slides 29-33 from Handout 13

for more details

2.2 Example

$$\forall x(\exists y(\neg x \lor y \lor \neg v) \to \exists z\big((x \to z) \lor \neg v\big), \forall x(\exists y(\neg x \lor y \lor \neg v) \vdash \forall z \exists y\big((x \to z) \lor \neg v\big)$$

Premises:

$$\forall x(\exists y(\neg x \lor y \lor \neg v) \to \exists z\big((x \to z) \lor \neg v\big) \text{ and}$$

$$\forall x(\exists y(\neg x \lor y \lor \neg v)$$

Consequence:

$$\forall z \exists y\big((x \to z) \lor \neg v\big)$$

	1	$\forall x(\exists y(\neg x \lor y \lor \neg v) \to \exists z\big((x \to z) \lor \neg v\big)$	premise
	2	$\forall x(\exists y(\neg x \lor y \lor \neg v)$	premise
*	3	$x_0 \exists y(\neg x_0 \lor y \lor \neg v)$	$\forall x$e 1
	4	$\exists y(\neg x_0 \lor y \lor \neg v)$	$\forall x$e 2
**	5	$\exists z\big((x_0 \to z) \lor \neg v\big)$	$\to$e $3, 4$
	6	$\forall x \exists z\big((x \to z) \lor \neg v\big)$	

* note: x_0 is a "fresh" new variable and can be named anything (as long as there isn't a another variable with that name.
** this is a substitution like $\varphi[x_0/x]$ (notation like the book)

2.3 Notation

Note that we can represent $\exists y(\neg x \vee y \neg v)$ simply as $\varphi(x, v)$ because y is bound. That is we can use *meta variables* to stand in for a WFF. To do this, you just write the free variable and you don't write down the bound variables. Common meta variables are φ and ψ This is because sometimes the details insides of the WFF don't really matter. In the example above, you may notice that $\neg v$ is never really considered for anything. It remains the same throughout the example. In fact, we can write $\varphi(x)$ for $\exists y(\neg x \vee y \neg v)$ since we don't really care about v. This shorthand help make automated tools run faster.

2.3.1 example

$$\forall x(\exists y(\neg x \vee y \vee \neg v) \to \exists z\big((x \to z) \vee \neg v\big), \forall x(\exists y(\neg x \vee y \vee \neg v) \vdash \forall z \exists y\big((x \to z) \vee \neg v\big)$$

$$\equiv$$

$$\forall x(\varphi(x, v)) \to \psi(x, v)), \forall x \varphi(x, v) \vdash \forall z \psi(x, v)$$

2.4

Keep in mind that $\forall, \exists$ are only bound to **T** and **F** in propositional logic. They are finite. We will need to be careful when we go to first-order and they become infinite.

2.5

Sentence: a complete statement. In PL it translates to "closed WFF with no free variables". We didn't see this term before because we always had free variables.

3 Prenex form

3.1 See slide 36 from Handout 13

for more details

3.2 definition

Prenex form: all the quantifiers are pushed to the from of the WFF.

3.2.1 In prenex form

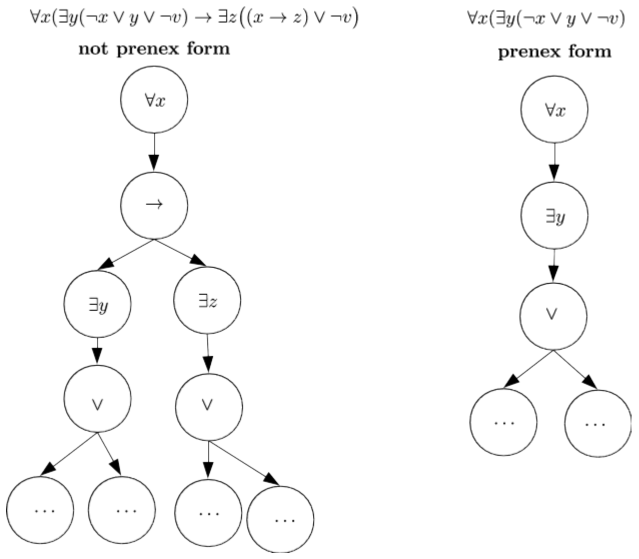
$$\forall x(\exists y(\neg x \vee y \vee \neg v)$$

3.2.2 Not in prenex form

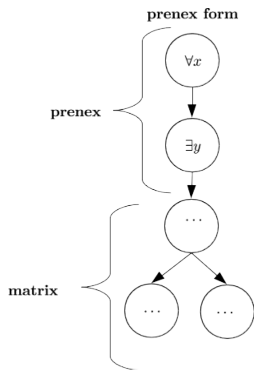
$$\forall x(\exists y(\neg x \vee y \vee \neg v) \to \exists z\big((x \to z) \vee \neg v\big)$$

3.2.3 How can you tell if a WFF is in prenex form?

You can check the parse tree of a WFF to see if it is in prenex form.


$\forall x(\exists y(\neg x \vee y \vee \neg v) \to \exists z((x \to z) \vee \neg v)$
not prenex form
$\forall x$
$\to$
$\exists y$
$\exists z$
$\vee$
$\vee$
$\forall x(\exists y(\neg x \vee y \vee \neg v)$
prenex form
$\forall x$
$\exists y$
$\vee$

The WFF on the right is not in prenex form because a connective is in between the quantifiers in the parse tree (i.e. $\to$ is child of $\forall$ and a parent of $\exists$). The WFF on the left is in prenex form because there are no quantifiers after the first connective $\vee$.
In general, the string of quantifiers is called the *prenex* and the rest of the WFF is called the *matrix*. There can't be any quantifiers in the matrix, if this is the case then the WFF isn't in prenex form.



You can push the quantifier up the parse tree, but you need to make sure you avoid *name capture*. This happens when you have a bound variable (i.e. x that is bound to $\forall x$) that has the same name as a free variable (i.e. somewhere in the WFF outside the scope of the $\forall x$ there exists another x). You can always just rename bound variables to something else to prevent name capture (so in our example, just give the x in the $\forall x$ a different name, like x_0).

If this doesn't make sense, or you don't believe it, review the formal semantics on slide 39 in handout 13 to test the soundness of this. You can also use *quantifier elimination* which is outlined in the next subsection.

Of note, pushing up quantifiers is a non-deterministic process. In other words, there are many ways to push up the quantifiers and they may end up in different orders. This is important because the number of alterations in a parse tree can affect the running time of QBF solvers.

3.3 Quantifier Elimination

$$\forall x \varphi(x) :\Leftrightarrow \varphi(x := T) \wedge \varphi(x := F)$$
$$\exists x \varphi(x) :\Leftrightarrow \varphi(x := T) \vee \varphi(x := F)$$

where $:\Leftrightarrow$ stands for "logically equivalent"
Another way to write this would be:

$$\forall x \varphi(x) :\Leftrightarrow \bigwedge_{i \in \forall x} \varphi(x := i)$$

$$\exists x \varphi(x) :\Leftrightarrow \bigvee_{i \in \forall x} \varphi(x := i)$$

But since we are only dealing with **T** and **F** we can actually write out all the possible values for x and expand the larger connectives.
$\forall x \varphi(x)$ means that every possible value of $\varphi(x)$ must be true (i.e. trying all values for x) and $\exists x \varphi(x)$ means that at least one possible values of $\varphi(x)$ must be true (again trying every value of x).

4 A trick to make automated tools faster

4.1 Why are some tools faster than others?

Recall from Assignment 4, that the tableaux method "prefers" DNF and that the resolution method "prefers" CNF. These tools need to consider every connective before coming to a conclusion.

4.2 the trick

Imagine you have the following WFF in CNF:

$$(A \vee \neg B \vee C \vee D) \wedge (A \vee \neg B \vee C \vee \neg E) \wedge (A \vee \neg B \vee C \vee \neg F)$$

Notice how each clause shares the first terms. Automated tools will need to check every connective, even though they will already know what $A \vee \neg B \vee C$ will evaluate to after the first time! We can use quantifiers to substitute in order to reduce the number of connectives.

$$(A \vee \neg B \vee C \vee D) \wedge (A \vee \neg B \vee C \vee \neg E) \wedge (A \vee \neg B \vee C \vee \neg F)$$

$$:\Leftrightarrow$$

$$\exists y \big(y \leftrightarrow (A \vee \neg B \vee C)\big) \wedge (y \vee D) \wedge (y \vee \neg E) \wedge (y \vee F)$$

Now we've seriously cut down on the number of connectives.