Introduction to the theory of computation
week 11 (Course Notes, chapter 6)

Logically equivalent formulas

Two formulas F_1 and F_2 are logically equivalent if and only if every interpretation that satisfies one satisfies the other. This is a pretty tall order: you can specify different domains, different meanings for the predicates, different values for the constants, and a different valuation, σ, for the variables, but still F_1 and F_2 must have the same truth value in every interpretation. We don't have any finite representation, such as truth tables, to establish the equivalence (since there are infinitely many interpretations). However, first-order formulas have some standard equivalences that allow us to transform them into equivalent formulas. Here are some:

Negate quantifiers: $\neg\forall x F$ LEQV $\neg\exists x \neg F$. Symmetrically, $\neg\exists x F$ LEQV $\forall x \neg F$. The equivalence of these follows from the definition of a formula being true/false in first-order logic.

Factor quantifiers over conjunctions and disjunctions: Suppose x is not a free variable of E, then

$$\begin{aligned} E \wedge \forall x F &\quad \text{LEQV} \quad \forall x (E \wedge F) \\ E \wedge \exists x F &\quad \text{LEQV} \quad \exists x (E \wedge F). \end{aligned}$$

However, be careful if x is a free variable of E:[1]

$$\begin{aligned} \forall x E(x) \wedge \forall x F(x) &\quad \text{LEQV} \quad \forall x (E(x) \wedge F(x)) \\ \exists x E(x) \wedge \exists x F(x) &\quad \text{NOT LEQV} \quad \exists x (E(x) \vee F(x)) \end{aligned}$$

Again, suppose x is not a free variable of E, then

$$\begin{aligned} E \vee \forall x F &\quad \text{LEQV} \quad \forall x (E \vee F) \\ E \vee \exists x F &\quad \text{LEQV} \quad \exists x (E \vee F) \end{aligned}$$

Again, be careful if x is a free variable of E:

$$\begin{aligned} \forall x E(x) \vee \forall x F(x) &\quad \text{NOT LEQV} \quad \forall x (E(x) \vee F(x)) \\ \exists x E(x) \vee \exists x F(x) &\quad \text{LEQV} \quad \exists x (E(x) \vee F(x)). \end{aligned}$$

Factor quantifiers over implications: There is a strange asymmetry, depending on whether you are factoring from the consequent or the antecedent. Assume that x is not a free variable of E.[2]

$$\begin{aligned} E \to \forall x F &\quad \text{LEQV} \quad \forall x (E \to F) \\ E \to \exists x F &\quad \text{LEQV} \quad \exists x (E \to F) \\ \forall x E \to F &\quad \text{LEQV} \quad \exists x (E \to F) \\ \exists x E \to F &\quad \text{LEQV} \quad \forall x (E \to F) \end{aligned}$$

RENAME QUANTIFIED VARIABLES: Suppose variable y does not occur in F, and let F_y^x denote the formula obtained by replacing every free occurrence of x by y in F. Then $\forall x F$ LEQV $\forall y F_y^x$, and $\exists x F$ LEQV $\exists y F_y^x$. You can do this systematically to make sure that the names of all bound variables are distinct from each other and all free variables (rectification).

USE EQUIVALENCES FROM PROPOSITIONAL LOGIC: De Morgan's Law, commutativity, associativity, distributivity, etcetera still apply.

SUBSTITUTE EQUIVALENT SUB-FORMULAS: Use any of the above rules to substitute an equivalent part of a formula.

PRENEX NORMAL FORM

By using the logical equivalences listed above, you can transform formula F into an equivalent formula F' where all the quantifiers precede a sub-formula that is quantifier-free. If Q_i represents either $\forall$ or $\exists$, it is possible to apply the logical equivalences until F' has the form:

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n E.$$

We say that F' is in PRENEX NORMAL FORM (PNF). Try this:

$$\begin{aligned}
&\forall x(\forall z((P(x,y,z) \vee \exists u S(z,u)) \to M(z)) && \text{LEQV} && \text{[renaming]}\\
&\forall x(\forall v(P(x,y,v) \vee \exists u S(v,u)) \to M(z)) && \text{LEQV} && \text{[factor quantifier]}\\
&\forall x \exists v((P(x,y,v) \vee \exists u S(v,u)) \to M(z)) && \text{LEQV} && \text{[factor quantifier]}\\
&\forall x \exists v(\exists u(P(x,y,v) \vee S(v,u)) \to M(z)) && \text{LEQV} && \text{[factor quantifier]}\\
&\forall x \exists v \forall u((P(x,y,v) \vee S(v,u)) \to M(z))
\end{aligned}$$

Notice how rectification made some of the factoring possible.

When a formula is in PNF, the order of quantifiers $\forall$ and $\exists$ becomes obvious. If $S(x,y,z)$ is interpreted as $x + y = z$, and our domain is the integers, what can you say about changing the order of the quantifiers:

$$\forall x \exists y S(x,y,0) \qquad \exists y \forall x S(x,y,0).$$

The meaning of PNF formulas is very sensitive to the order of quantifiers. Is either the formula below, or its converse, a valid formula (is the consequent satisfied by every interpretation that satisfies the antecedent)?

$$\forall x \exists y M(x,y) \to \exists y \forall x M(x,y)$$

Is the following a valid formula?[3]

$$\forall x \exists y(M(x) \wedge F(y)) \leftrightarrow \exists y \forall x(M(x) \wedge F(y))$$

MORE SCOPE

We've already seen that free variables are those that are not bound by (i.e. don't occur within the SCOPE of) any quantifier. Another subtlety is WHICH quantifier a given bound (dummy) variable is bound by. Consider the following formula, where $F(y)$ is interpreted as "y is female," $M(y)$ is interpreted as "y is male," and $S(x,y)$ is interpreted as "x and y are siblings."

$$\forall x \forall y(F(y) \wedge \forall y(S(x,y) \to M(y)) \to \neg S(y,x))$$

The intended meaning is that every female is not a sibling of anyone who has only male siblings. Which quantified y binds which instance of variable y? The y in $S(y,x)$ is bound to the "nearest" (working out) $\forall y$. Things become clearer if you rename that y to, say, u, or draw a tree representation.

Formal languages (course notes, chapter 7)

In the theory of formal languages, we consider languages as sets of strings over an alphabet (loosely analogous to the written form of some natural languages such as English or Aramaic). Here are some definitions and conventions:

An alphabet: Is a set (often denoted Σ) whose elements are atomic (not sub-dividable) symbols (characters).

A string: (over Σ) is a finite sequence of symbols from Σ. The empty sequence, denoted ε, is a string of length zero. The set of all possible strings over Σ is denoted Σ^*. For example, 0110 is a string over $\Sigma = \{0, 1\}$. We usually use variables from the beginning of the Latin alphabet $(a, b, c \ldots)$ to denote symbols, and variables from the end of the Latin alphabet (w, x, y, z) to denote strings.

A language: (over Σ) is a subset of Σ^*. In particular, Σ^* itself is a language. We often assume that $\Sigma = \{0, 1\}$ in this course.

String operations: If x and y are strings, then $|x|$ is the length of x (so $|\varepsilon| = 0$), and xy is the concatenation of x and y. $(x)^R$ is the reversal of string x (so $(string)^R = gnirts$). The kth power of a string is defined recursively:

$$x^k = \begin{cases} \varepsilon, & k = 0 \\ x^{k-1}x, & k > 0 \end{cases}$$

Strings x and y are equal if $|x| = |y|$ and $x[i] = y[i]$ for $0 \leq i < |x|$. Notions of (proper) substrings, prefixes, and suffixes are exactly what you would expect, provided you allow ε to be a prefix, substring, or suffix of any string.

Why do we care about formal languages

Manipulating formal languages is an import of computer science:

- Compilers need to recognize the language of identifiers, the language of arithmetic expressions, and translate source code into the language of the processor's instruction set.
- Part of bioinformatics involves distinguishing "interesting" strings over $\Sigma = \{A, C, T, G\}$ from other strings.
- We've already constructed and manipulated languages of first-order formulas, propositional formulas.
- Difficult problems in theory derived from deciding whether a given string s is in some language L.

Operations on languages

Operations on languages combine set operations with (possibly repeated) string concatenation. Suppose L_1 and L_2 are languages over Σ:

Complement: $\overline{L_1}$ (the complement of L_1) is $\Sigma^* - L_1$.

Union: $L_1 \cup L_2$ contains any string that is in either language (or both).

Intersection: $L_1 \cap L_2$ contains any string that is in both languages.

Concatenation: L_1L_2 contains all strings of the form xy, where $x \in L_1$ and $y \in L_2$. Notice that $\emptyset L_1 = \emptyset$ for any language L_1.

KLEENE STAR: $\varepsilon \in L_1^*$, and if $x \in L_1^*$ and $y \in L_1$, then $xy \in L_1^*$. Informally, all possible concatenations of 0 or more strings from L_1.

EXPONENTIATION: Defined recursively:

$$L_1^k = \begin{cases} \{\varepsilon\}, k = 0 \\ L_1^{k-1} L_1, \quad k > 0 \end{cases}$$

Informally, all concatenations of k strings from L_1.

REVERSAL: $\text{Rev}(L_1) = \{(x)^R : x \in L_1\}$, the set of all reversals of strings in L_1.

Regular expressions

A very terse description of a formal language is often possible, using a regular expression. The set of regular expressions (RE) over Σ (this set is itself a language), is defined recursively:

BASIS: $\emptyset$, ε, and a (for each $a \in \Sigma$) are in RE.

INDUCTION STEP: If R and S are in RE, then so are $(R + S)$, (RS) and R^*. Notice that since each regular expression has finite length, it uses only a finite subset of Σ.

EXAMPLES: Over alpha $\Sigma = \{0, 1\}$, you may have regular expressions 0, $(0 + 1)$, $((01) + (10))$.

The language denoted by a given regular expression, R, is called $L(R)$, and defined inductively (what else?).

BASIS: $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\epsilon\}$, $L(a) = \{a\}$ (for any $a \in \Sigma$).

INDUCTION STEP: Suppose $L(R)$ and $L(S)$ have been defined inductively. Then $L(R + S) = L(R) \cup L(S)$, $L(RS) = L(R)L(S)$, and $L(R^*) = (L(R))^*$.

Reduce parentheses by omitting the outer parentheses and parentheses that don't change associative operations like union and concatenation. Also note that concatenation has higher precedence than union, and Kleene star has higher precedence than either. Here are some examples of regular expressions and the languages they denote:

Regular expression	Language
$(0 + 1)^*$	All strings containing only 0s and 1s (Σ^*).
$((0 + 1)(0 + 1)^*)$	All non-empty strings in Σ^*
$((0 + 1)(0 + 1))^*$	All even-length strings
$\varepsilon + 0 + 0(0 + 1)^*0$	All strings that don't begin or end in 1.

The computer utility "grep" implements regular expressions with Σ^* as a text file, $(a + e)$ is denoted $[ae]$, and $(a + b + \cdots + z)$ is denoted .*. Depending on where the file "words" lives, you might try:

grep ĝ[ae]m.*t$ /usr/share/dict/words

The material below was NOT covered during the lecture, but you should be familiar with it.

In general there is more than one way to denote the same language. Suppose $L = \{x \in \Sigma^* : x \text{ contains at least one } 0\}$. Then $L = L(R_1) = L(R_2) = L(R_3)$, where $R_1 = (0 + 1)^*0(0 + 1)^*$, $R_2 = 1^*0(0 + 1)^*$, and $R_3 = (0 + 1)^*01^*$. To prove from scratch that two regular expressions are equivalent means that you prove that the languages (sets) denoted by them are equal, usually by mutual inclusion.

CLAIM: $L(1^*0(0 + 1)^*) \subseteq L$.

PROOF: Let x be a arbitrary string of $L(1^*0(0+1)^*)$. This means, by definition, that $x = u0t$, where $u \in L(1^*)$, and $t \in L((0+1)^*)$. By inspection $u0t$ contains at least one 0, so $x \in L$. Since x was chosen arbitrarily, $L(1^*0(0+1)^*) \subseteq L$. QED.

CLAIM: $L \subseteq L(1^*0(0+1)^*)$. Let $x \in L$ be an arbitrary string in L. Since x has at least one zero, it certainly has a first zero, so x can be parsed as $x = u0v$, where u is the (possibly empty) prefix of x before the first 0, and v is the (possibly empty) suffix of x after the first 0. Thus $u \in L(1^*)$ (since it contains no zeros), and $v \in L((0+1)^*)$ (since it is some string over $\{0,1\}$), thus $x = u0v \in L(1^*)L(0)L((0+1)^*)$. Since x was chosen arbitrarily, $L \subseteq L(1^*0(0+1)^*)$. QED.

Thus we have shown that $L \subseteq L(1^*0(0+1)^*)$ and $L(1^*0(0+1)^*) \subseteq L$, so the two languages are equivalent. A similar sort of proof is necessary to show that $L(1^*0(0+1)^*)$ is the same language as $L((0+1)^*01^*)$.

CLAIM: $L((0+1)^*01^* \subseteq L(1^*0(0+1)^*)$.

PROOF: Let $x = u0t$ be an arbitrary element of $L((0+1)^*01^*)$, where $u \in L((0+1)^*)$, $0 \in L(0)$, and $t \in L(1^*)$. If u contains no zeros, then $u \in L(1^*)$, and (in any case) $t \in L((0+1)^*)$, and $x = u0t \in L(1^*0(0+1)^*)$. Otherwise, u has a first 0, and can be re-written as $u = z0y$, where z is the (possibly empty) zero-free prefix of u and y is the suffix of u following its first 0. Thus $z \in L(1^*)$, and $y0t \in (0+1)^*$, so $x = z0y0t \in L(1^*)L(0)L((0+1)^*)$, that is $x \in L(1^*0(0+1)^*)$. Since x was chosen arbitrarily, $L((0+1)^*01^*) \subseteq L(1^*0(0+1)^*)$. QED.

CLAIM: $L(1^*0(0+1)^*) \subseteq L((0+1)^*01^*)$.

PROOF: Let $x = u0t$ be an arbitrary element of $L(1^*0(0+1)^*)$, where $u \in L(1^*)$, $0 \in$ Ł(0), and $t \in L((0+1)^*)$. Certainly $u \in L((0+1)^*)$, and if t contains no zeros then $5 \in L(1^*)$, and we're done since $x = u0t \in L((0+1)^*01^*)$. Otherwise, t contains a last zero, and can be re-written as $t = z0y$, where z is the (possibly empty) prefix before the last zero, and y is the (possibly empty) zero-free suffix. In this case, $u0z \in L((0+1)^*)$ and $y \in L(1^*)$, so $x = u0z0y \in L((0+1)^*01*)$. In either case, $x \in L((0+1)^*01^*)$, and since x was chosen arbitrarily, this shows that $L(1^*0(0+1)^*) \subseteq L((0+1)^*01^*)$. QED.

EQUIVALENCE: Since $L(1^*0(0+1)^*)$ and $L((0+1)^*01^*)$ contain each others, they are equal.

Notes

[1] Try $E(x)$ means "x is odd," and $F(x)$ means "x is even" in the second formula.

[2] If you use the $\rightarrow$ rule, these become clear.

[3] Try factoring the quantifiers.