

FIRST-ORDER LOGIC: BOUND VARIABLES, FREE VARIABLES*

Ian Barland
John Greiner
Phokion Kolaitis
Matthias Felleisen
Moshe Vardi

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

In the previous examples we often re-used variable names, even within the same formula. This shouldn't be surprising or confusing, since we do the same thing in programs (another formal language). In fact, the same notions of **bound** and **free** variables occur in both situations. An occurrence of variable x is bound if it is in the body of a quantifier $\forall x \dots$ or $\exists x \dots$. Otherwise, the occurrence is free.

For example, in $\forall x : (\text{likes}(x, y))$, the variable y is free but x is not. So this is a statement about y ; we can't evaluate this to true/false until we get some context for y . It's useful as a subpart for some bigger formula.

NOTE: The concept "x free in ϕ " does **not** talk about the context of ϕ . So don't confuse it with "well, over on this part of the page, ϕ happens to occur as the sub-part of another formula containing $\forall x : (\dots)$, so x really is bound." (Just as 7 is prime, even though people sometimes use 7 in the context of 7+1.) Whether x is free in a ϕ can be determined by a function `isFree(x, ϕ)`, needing no other information to produce an answer.

Looking back at our previous examples, we can see that many of the formulas we made had no free variables — all variables were bound by some quantifier in the formula. The truth of such formulas depends only on the interpretation and not on any additional knowledge about what any free variables refer to. Thus, these formulas are common and important enough that we give them a special name, **sentences**.

A given variable name can actually have **both** bound and free occurrences within the same formula, as in $R(x) \wedge \exists x : (\neg(R(x)))$. (This formula about x is satisfiable: it says that R is true about x , but isn't true about everything.) In essence, there are two different underlying variables going on, but they each happen to have the same name; from scope it can be decided which one each occurrence refers to. In programming language terms, we'd say that the inner x (the local variable) **shadows** the outer x (the enclosing variable). In these terms, free variables in logic correspond to global variables in programs.

Clearly $\forall x : (R(x))$ is always equivalent to $\forall y : (R(y))$; variable names are entirely arbitrary (except maybe for their mnemonic value). So the previous formula might be more clearly re-written as $R(x) \wedge \exists y : (\neg(R(y)))$. (This careful re-writing while respecting a variable's scope is called **α -renaming**.) Even if

*Version 1.7: Jan 9, 2009 10:42 am US/Central

†<http://creativecommons.org/licenses/by/1.0>

17 quantifiers each used the same variable (name) x , we could carefully α -renaming 17 times, and end up with an equivalent formula where all quantifiers use distinct variables. This will be useful to avoid potential confusion, especially in the upcoming inference rules¹, where we'll be introducing and eliminating quantifiers.

Example 1

The formula $\forall x : (A(x)) \wedge \exists x : (B(x)) \wedge \forall x : (C(x))$ is equivalent to the more readable $\forall x : (A(x)) \wedge \exists y : (B(y)) \wedge \forall z : (C(z))$.

¹"First-Order Logic: inference rules": Note <<http://cnx.org/content/m10774/latest/#renaming-for-subst>>