CMPSCI 250: Introduction to Computation

Lecture #8: Proofs With Quantifiers
David Mix Barrington
8 February 2012

Proofs With Quantifiers

- The Meaning of Quantifier Proofs
- The Four Proof Rules
- Instantiation: Eliminating ∃
- Existence: Introducing ∃
- Specifation: Eliminating ∀
- Generalization: Introducing ∀
- The Dog Example

The Meaning of Quantifier Proofs

- A quantified statement talks about a particular situation -- a set of objects divided into data types, and some predicates with arguments from particular types. For every legal **atomic statement**, which is a predicate with arguments of the proper type filled in, we need to have the truth value defined.

- We may also have **constants** -- values from specific types that are given names.

- Our final example today will have a set of dogs D, three unary predicates on dogs W(x) “x likes walks”, R(x) “x is a Rottweiler”, and T(x) “x is a terrier”, and a binary predicate S(x, y) “dog x is smaller than dog y”. We will also have two constants of type D, Cardie (c) and Duncan (d). There are an infinity of possible models of these predicates -- we want to show that any model that satisfies our premises also satisfies our conclusion.

The Four Proof Rules

- In the Forward-Backward method, we have one statement that we want to go forward from, and another we want to go backward from. The structure of these statements lets us know what kind of quantifier proof rule we can use. In particular, the *outermost* quantifier controls what we can do.

- To go forward from a universal statement, we want to **eliminate** the universal quantifier, that is, go from $\forall x: P(x)$ to $P(a)$. To go forward from an existential statement, we want to eliminate the $\exists$, going from $\exists x: P(x)$ to $P(a)$. The rules of Specification and Instantiation will allow us to do this, but we must be careful of our context and the meaning of our variables.

- To go backward from a universal statement, we want to prove $\forall x: P(x)$, **introducing** a universal quantifier. The rule of Generalization lets us do this, and the rule of Existence lets us introduce an existential and prove $\exists x: P(x)$.

Instantiation: Eliminating ∃

- Given the premise “there exists a dog that is a terrier” (∃x: T(x)), the rule of **Instantiation** lets us derive a statement T(a), eliminating the quantifier.

- In English, we would say “let a be the dog that exists by the premise, so that we know T(a)”. Here “a” must be a new variable, referring to a new dog. We *don’t know* whether that new dog is equal to any old dogs, or whether any of the other predicates are true for it. We know only its type and the fact T(a) that we got from the statement we instantiated.

- In essence we are giving a name to one of the dogs, who may or may not be one of the dogs we already know something about. A common error is to say something like “a terrier exists, therefore that terrier is Duncan”, claiming a name or a property of the instantiated object with no justification.

Existence: Introducing ∃

- How do we prove a statement like ∃x: P(x), introducing an existential quantifier? The rule of **Existence** says that from any statement of the form P(a), we can derive ∃x: P(x). For example, given the premise "Duncan is a terrier (T(d)), we can derive "there exists a dog that is a terrier" (∃x: T(x)).

- We have to be careful to introduce the existential quantifier so that its scope covers the entire statement that we are using. If I have premises T(d) and R(c), for example, I could derive ∃x: T(x) and ∃x: R(x). But it would be a mistake to derive ∃x: (T(x) ∧ R(x)), "there is a dog that is both a terrier and a Rottweiler". To get that I would need a single statement T(a) ∧ R(a), with the same a in both places.

- If I have ∃x:(T(x) ∧ R(x)), and I want to derive ∃y: T(y), I should *first* use Instantiation to say T(a) ∧ R(a) for some a, then separation to get T(a), then Existence. There is no rule to go directly from this premise to this conclusion, and being sloppy with these rules can lead to errors.

Specification: Eliminating ∀

- If we have a universal statement of the form ∀x: P(x), then the rule of **Specification** allows us to derive P(a), where a is *any* constant or variable of the same type as x. That is, we can derive P(a) for an a *of our choice.*

- If we have the statement ∀x: W(x),“all dogs like walks”, we can derive W(d), W(c), or W(y) for a free variable y that already appears in other statements.

- The one caveat to remember is that in principle we remove one universal at a time, and when we remove it we must set all occurrences of the bound variable to the *same* value. Given, say, ∀x: W(x) ∧ S(x, d) ∧ (T(y) → S(y, x)), we could derive W(c) ∧ S(c, d) ∧ (T(y) → S(y, c) or W(y) ∧ S(y, d) ∧ (T(y) → S(y, y), but we couldn’t replace some x’s with c’s and others with y’s. The one thing we can’t do is set x to an existing *bound* variable -- we could not go from ∀x: ∀y: T(y) → S(x, y) to ∀y: T(y) → S(y, y). This is because the bound y is defined *after* we set the value of x, so we can’t force the two to be the same.

Generalization: Introducing ∀

- We've just seen that universal statements are very powerful, so it stands to reason that we should have to work harder to prove them. The rule of **Generalization** allows us to prove new universal statements.

- To prove a statement ∀x: P(x), we first say "let y be arbitrary", where y is a new variable of the type of x. We then have to prove that P(y) is true, without using any information about y other than its type. If we do this, we may then derive ∀x: P(x).

- We most often use this in the form ∀x: (P(x) → Q(x)), so that we let y be arbitrary and then have to prove P(y) → Q(y). To do this we can assume P(y) and use it to derive Q(y), which may be possible if P and Q are related. When we do mathematical induction, we will prove statements of the form ∀x: P(x), where x is a natural, in part by proving ∀x: (P(x) → P(x+1)).

The Dog Example

- We have a set of dogs D, and predicates R(x) “x is a Rottweiler”, T(x) “x is a terrier”, S(x, y) “x is smaller than y”, W(x) “x likes to go for walks”.
- Premises: (1) All dogs like to go for walks ($\forall x: W(x)$), (2) Duncan is a terrier ($T(d)$), (3) Cardie is smaller than some Rottweiler ($\exists x: R(x) \wedge S(c, x)$), (4) All terriers are smaller than Cardie ($\forall x: T(x) \rightarrow S(x, c)$) (5) S is transitive ($\forall x: \forall y: \forall z: (S(x, y) \wedge S(y, z)) \rightarrow S(x, z)$.
- Desired conclusion: There exists a Rottweiler that is larger than some terrier who likes walks ($\exists x: \exists y: R(x) \wedge S(y, x) \wedge T(y) \wedge W(y)$).
- Overall strategy: Figure out which dogs x and y ought to be -- maybe constants, maybe dogs forced to exist by the premises. In this case y should be Duncan, and x should be the Rottweiler provided by premise (3).

More of the Dog Example

- We use Instantiation on (3) to get a dog r such that $R(r) \wedge S(c, r)$.
- We need four facts about d and r: We have $R(r)$, and we need $W(d)$, $T(d)$, and $S(d, r)$.
- We have $T(d)$ by (2), and we get $W(d)$ by Specification on (1).
- To get $S(d, r)$, we use Specification on (4) to get $T(d) \rightarrow S(d, c)$, Modus Ponens to get $S(d, c)$ since we have $T(d)$, and finally Specification on (5) to get $(S(d, c) \wedge S(c, r)) \rightarrow S(d, r)$ and Conjunction and Modus Ponens to get $S(d, r)$.
- Once we have these four facts we use Existence twice to get our desired conclusion $\exists x: \exists y: R(x) \wedge S(y, x) \wedge T(y) \wedge W(y)$.