Lecture Notes
CS2LAP: Logic and Prolog 2000/2001

Logic: Proof Methods Summary
Ulle Endriss

Contents

Version: October 18, 2000

These notes provide an overview of the three proof methods for classical propositional logic introduced in CS2LAP, namely truth tables, natural deduction, and the goal-directed system. The last section also introduces goal-directed deduction for first-order predicate logic. The material presented here is covered in more depth as well as breadth in the book *Elementary Logics: A Procedural Perspective* by Dov Gabbay (Prentice Hall Europe, 1998).

1 Truth Tables

Basic truth tables. The truth tables for the basic connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), and $\Rightarrow$ (implication) are given as follows:

A	$\neg A$
$\top$	$\bot$
$\bot$	$\top$

A	B	$A \wedge B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$

A	B	$A \vee B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$

A	B	$A \Rightarrow B$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\top$

From the last of these tables, for example, you can see that the only way to make an implication false is to make its antecedent true and its consequent false.

Computing truth tables. The basic truth tables can be used to compute the truth tables of more complex formulas. To do so, first count the number of propositional variables occurring in your formula. If there are n variables the table will have 2^n rows. Every row corresponds to a permutation of the possible truth values $\top$ and $\bot$, as you can see in the following example for $n = 3$:

A	B	C	...
$\top$	$\top$	$\top$	...
$\top$	$\top$	$\bot$	
$\top$	$\bot$	$\top$	
$\top$	$\bot$	$\bot$	
$\bot$	$\top$	$\top$	
$\bot$	$\top$	$\bot$	
$\bot$	$\bot$	$\top$	
$\bot$	$\bot$	$\bot$	

For every subformula of the formula of which you want to compute the truth table you should introduce a column. Then compute the truth values step by step according to the basic tables. Consider for example the formula $\neg(A \vee B)$. Its only subformula (besides the formula itself) is $A \vee B$. Hence, the truth table should look like this:

A	B	$A \vee B$	$\neg(A \vee B)$
$\top$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\top$	$\bot$
$\bot$	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\top$

The column for $A \vee B$ is just a copy of the basic truth table for $\vee$. The values in the last column are computed by applying the basic truth table for $\neg$ to the second but last column. That simply means for every $\top$ we put a $\bot$ and vice versa.

Some definitions. Recall that a formula for whose truth table all rows evaluate to $\top$ is called a *tautology*. If all rows have a $\bot$ the corresponding formula is a *contradiction* (also called an *inconsistent* formula). If for at least one row we get a $\top$ the formula is said to be *consistent*.

2 Natural Deduction

This section summarises most of what you should know about natural deduction (the forward rules).

Rules. Table 1 shows the 'official' set of natural deduction rules for CS2LAP. All other rules that you might have seen during the course or in the textbook can be proved from this set of official rules. And that's exactly what you have to do, if you want to use additional rules in coursework or exam questions: prove them first! (That is unless otherwise instructed, of course.)

The intended meaning of such a rule is the following: *given the formula(s) above the horizontal line you can infer the formula below the line.* Take for example the modus ponens[1] rule:

$$\frac{A, A \Rightarrow B}{B}$$

This says, that if you have (proved or as one of your assumptions) A and $A \Rightarrow B$ already, you can derive B. In (almost) plain English: if you know A and you know that B follows from A then you also know B.

Of course this also holds for P and Q or $A_1 \vee A_2$ and $\neg D$ or any other propositional formulas. A, B, C are just schemes!

When looking for these schemes in your formulas you always have to go by the main connective. For example, you cannot apply ($\wedge$E) to

$$P \wedge Q \Rightarrow R,$$

because the main connective of that formula is $\Rightarrow$, not $\wedge$. This would be easier to see if we had used parentheses, but it suffices to recall the precedence hierarchy of the connectives, i.e. that $\wedge$ binds stronger than $\Rightarrow$. Therefore, the above formula is equivalent to

$$(P \wedge Q) \Rightarrow R.$$

[1] Quite obviously, instead of modus ponens we could also say $\Rightarrow$-elimination.

$\wedge$-introduction ($\wedge$I) $$\frac{A, B}{A \wedge B}$$	$\wedge$-elimination ($\wedge$E) $$\frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$
$\vee$-introduction ($\vee$I) $$\frac{A}{A \vee B} \qquad \frac{B}{A \vee B}$$	$\vee$-elimination ($\vee$E) $$\frac{A \vee B, A \Rightarrow C, B \Rightarrow C}{C}$$
$\Rightarrow$-introduction ($\Rightarrow$I) $$\text{if } \frac{C_1, \ldots, C_k, A}{B} \text{ then } \frac{C_1, \ldots, C_k}{A \Rightarrow B}$$	modus ponens ($\Rightarrow$E) $$\frac{A, A \Rightarrow B}{B}$$
$\neg$-introduction ($\neg$I) $$\frac{A \Rightarrow B, A \Rightarrow \neg B}{\neg A}$$	$\neg$-elimination ($\neg$E) $$\frac{\neg A \Rightarrow B, \neg A \Rightarrow \neg B}{A}$$

Table 1: The Official Natural Deduction Rules for CS2LAP

Precedence hierarchy reminder. $\neg$ binds stronger than $\wedge$ binds stronger than $\vee$ binds stronger than $\Rightarrow$ (which again binds stronger than $\Leftrightarrow$, but that doesn't matter in this context as we aren't using any forward rules for $\Leftrightarrow$).

Boxes. Besides understanding the rules you also need to know about the natural deduction *notation*. Natural deduction proofs are organised as series of boxes. To prove $\mathcal{P} \vdash A$, i.e. to show that formula A logically follows from the set of formulas $\mathcal{P}$ you start by writing those formulas (data) into the first box. If there is no data, that's also fine – it means you are supposed to prove a tautology. You should write the goal A in the upper right-hand corner of your first box to indicate what you intend to prove. Then you can start applying rules. If any of the formulas in previous lines match the schemes in a deduction rule, apply it and add the conclusion to your box. Implications are proved using subcomputations: you start a new box, assume the antecedent and show the consequent from it.

An example. In the textbook on page 80 an additional rule for introducing an implication is listed:

$$(\Rightarrow\text{I1}) \ \frac{\neg A}{A \Rightarrow B}$$

As mentioned before, this rule is superfluous; its validity can be shown using only the rules from Table 1. We have to prove $\neg A \vdash A \Rightarrow B$.

		$A \Rightarrow B$
(1)	$\neg A$	data
(2)	$A \Rightarrow B$	from subcomputation
		B
(2.1)	A	assumption
(2.2)	$\neg B \Rightarrow A$	from subcomputation
		A
(2.2.1)	$\neg B$	assumption
(2.2.2)	A	from (2.1)
(2.3)	$\neg B \Rightarrow \neg A$	from subcomputation
		$\neg A$
(2.3.1)	$\neg B$	assumption
(2.3.2)	$\neg A$	from (1)
(2.4)	B	($\neg$E) with (2.2) and (2.3)

So how did that work? The main box is used to show that $A \Rightarrow B$ follows from the data $\neg A$. $A \Rightarrow B$ is proved via subcomputation, in the second box. We assume A (2.1) and show B (2.4) by means of the following dirty little trick: apparently out of the blue, but actually using our logical intuition, we construct the two implications (2.2) and (2.3) with $\neg B$ in the antecedent. And the consequents of those implications are contradictory: A and $\neg A$, i.e. we can apply $\neg$-elimination and derive B. Note that (2.2.2) and (2.3.2) have not been derived applying a rule, they've just been rewritten from previous lines.

In this proof you get examples for all three categories of steps you can apply during a natural deduction:

- Subcomputations, like e.g. (2). To show $A \Rightarrow B$ we assume A and show B, possibly also using other formulas of the box. Subcomputations are actually applications of ($\Rightarrow$I). This is the only rule where we don't write down the name of the rule (because we write 'subcomputation' instead).[2]

- Applications of the other rules from Table 1, e.g. (2.4). Quoting one, two, or three of the previous line numbers and the name of the rule we write down the conclusion of that rule.

- Rewriting, e.g. (2.2.2). Any formula *within the scope of the current box* may be repeated.

The '*within the scope of the current box*' bit is important! It also applies to subcomputations and rule applications. It means, you can only use formulas from the box you're currently working in or from boxes *around* your current box. Assume, for example, you

[2]It is also possible (and done so in the textbook) to present ($\neg$I) and ($\neg$E) as subcomputations, but you can always achieve the same result by first introducing the necessary implications (using subcomputation) and then applying ($\neg$I) or ($\neg$E), just as it has been done in the example above.

were stuck in line (2.3.1). At this point you could for example use (1), but not (2.2.2) – the two small boxes are not in the scope of each other. Goes without saying you also cannot use formulas you haven't yet proved to hold: in, say, line (2.3) you cannot use (2), because you are still inside the subcomputation-box that will eventually prove (2).

Some more examples. The following is a natural deduction proof for the transitivity of $\Rightarrow$. We have to show $(A{\Rightarrow}B) \wedge (B{\Rightarrow}C) \vdash A{\Rightarrow}C$.

			$\underline{A{\Rightarrow}C}$
(1)	$(A{\Rightarrow}B) \wedge (B{\Rightarrow}C)$	data	
(2)	$A{\Rightarrow}B$	$(\wedge\mathrm{E})$ with (1)	
(3)	$B{\Rightarrow}C$	$(\wedge\mathrm{E})$ with (1)	
(4)	$A{\Rightarrow}C$	from subcomputation	
			$\underline{C}$
	(4.1)	A	assumption
	(4.2)	B	modus ponens with (4.1) and (2)
	(4.3)	C	modus ponens with (4.2) and (3)

The next one is a proof for the double-negation rule, which is part of many deduction systems. Quite obviously from $\neg\neg A$ we can infer A. But how to formally prove it in our official natural deduction framework? Again, like on page 6 we have to come up with a trick to get A with an application of $(\neg\mathrm{E})$.

			$\underline{A}$
(1)	$\neg\neg A$	data	
(2)	$\neg A{\Rightarrow}\neg A$	from subcomputation	
			$\underline{\neg A}$
	(2.1)	$\neg A$	assumption
	(2.2)	$\neg A$	from (2.1)
(3)	$\neg A{\Rightarrow}\neg\neg A$	from subcomputation	
			$\underline{\neg\neg A}$
	(3.1)	$\neg A$	assumption
	(3.2)	$\neg\neg A$	from (1)
(4)	A	$(\neg\mathrm{E})$ with (2) and (3)	

The formula $((A{\Rightarrow}B){\Rightarrow}A){\Rightarrow}A$ is known as Peirce's law. To show that it is a tautology, besides the official rules we also use $({\Rightarrow}\mathrm{I}1)$, which has been proved on page 6. Alternatively, we could repeat the box used to prove $\neg A \vdash A{\Rightarrow}B$ here instead of line (1.2.2).

		$\underline{((A\Rightarrow B)\Rightarrow A)\Rightarrow A}$
(1)	$((A\Rightarrow B)\Rightarrow A)\Rightarrow A$	from subcomputation
		$\underline{A}$
(1.1)	$(A\Rightarrow B)\Rightarrow A$	assumption
(1.2)	$\neg A\Rightarrow A$	from subcomputation
		$\underline{A}$
(1.2.1)	$\neg A$	assumption
(1.2.2)	$A\Rightarrow B$	($\Rightarrow$I1) with (1.2.1)
(1.2.3)	A	modus ponens with (1.2.2) and (1.1)
(1.3)	$\neg A\Rightarrow\neg A$	from subcomputation
		$\underline{\neg A}$
(1.3.1)	$\neg A$	assumption
(1.3.2)	$\neg A$	from (1.3.1)
(1.4)	A	($\neg$E) with (1.2) and (1.3)

3 Goal-Directed Deduction

The goal-directed (backward) rules provide an alternative method to test whether a given formula (goal, conclusion, query) A logically follows from a given set of premises (assumptions, data) $\mathcal{P}$, i.e. whether $\mathcal{P} \vdash A$ holds.

Ready for computation form. The goal-directed method can only deal with the connectives $\wedge$ and $\Rightarrow$, and $\perp$ for falsity. Formulas also need to have a special structure called ready for computation form. Formulas in ready for computation form can be defined recursively as follows:

(i) Atoms A as well as $\perp$ are in ready for computation form.

(ii) An implication $A\Rightarrow B$ is in ready for computation form, if B (called the head or the consequent) is an atom or $\perp$ and A (called the body or the antecedent) is a conjunction of formulas in ready for computation form.

Examples. Here are some examples for formulas in ready for computation form:

1. $(A\Rightarrow B)\Rightarrow\perp$
2. $(A \wedge (A\Rightarrow B))\Rightarrow B$
3. $(((A \wedge B \wedge C)\Rightarrow A) \wedge D)\Rightarrow B$

And here are some that aren't:

1. $\neg A$ (Negation is not allowed.)
2. $(A\Rightarrow B)\Rightarrow(A\Rightarrow\perp)$ (The head isn't atomic.)
3. $A \wedge B$ (This is a conjunction of two ready for computation formulas however.)

Translation. Before applying any goal-directed rules all formulas have to be translated into ready for computation form. Any propositional formula with the connectives $\neg$, $\wedge$, $\vee$, and $\Rightarrow$ can be translated into an equivalent formula that is a *conjunction of formulas in ready for computation form* by exploiting the following equivalences:

$$\begin{array}{lcl} A\Rightarrow(B \wedge C) & \equiv & (A\Rightarrow B) \wedge (A\Rightarrow C) \\ A\Rightarrow(B\Rightarrow C) & \equiv & (A \wedge B)\Rightarrow C \\ \neg A & \equiv & A\Rightarrow\bot \\ A \vee B & \equiv & (A\Rightarrow\bot)\Rightarrow B \end{array}$$

These equivalences have to be used (in whatever order) as long as there still are subformulas they can be applied to. (Note that this presentation, though being feasible for human beings, does not lead itself to a straightforward implementation. For automatisation you would rather specify some more rules, which then could be applied systematically form the outside to the inside.)

Example. Here's an example for a translation:

$$\begin{array}{ll} A \vee B\Rightarrow\neg C & \equiv \\ ((A\Rightarrow\bot)\Rightarrow B)\Rightarrow(C\Rightarrow\bot) & \equiv \\ ((A\Rightarrow\bot)\Rightarrow B) \wedge C\Rightarrow\bot & \end{array}$$

In the first translation step we have rewritten $A \vee B$ as $(A\Rightarrow\bot)\Rightarrow B$ and $\neg C$ as $C\Rightarrow\bot$. Then we have transformed the resulting implication with another implication in the consequent into an implication with a conjunction in the antecedent.

Goal-directed proofs. Once all formulas have been translated into ready for computation form we can commence with the proof. The data is now a set of conjunctions of ready for computation formulas. We further split every conjunction into the set of all its conjuncts, i.e. we transform the data into a set of ready for computation formulas. The goal, however, could still be a conjunction.

It is also possible to start a goal-directed proof even if not all formulas involved are in ready for computation form yet, and to interleave rewriting steps and applications of the goal-directed rules – *if you know what you are doing!*

Rules. Given a set of data formulas in ready for computation form and a goal formula (which has to be a conjunction of formulas in ready for computation form) we use the following goal-directed rules to show that the goal logically follows from the data:

1. If the current goal A is atomic and can be found in the data, **succeed**.

2. If $\bot$ can be found in the data, **succeed**.

3. **Rule for implication:** to show $\mathcal{P} \vdash A\Rightarrow B$ show $\mathcal{P}, A \vdash B$.

4. **Rule for conjunction:** to show $\mathcal{P} \vdash A \wedge B$ show $\mathcal{P} \vdash A$ and $\mathcal{P} \vdash B$.

5. **Rule for atoms:** to show $\mathcal{P}, A{\Rightarrow}B \vdash B$ show $\mathcal{P}, A{\Rightarrow}B \vdash A$.
6. **Rule for atoms (with $\bot$):** to show $\mathcal{P}, A{\Rightarrow}\bot \vdash B$ (B atomic) show $\mathcal{P}, A{\Rightarrow}\bot \vdash A$.
7. **Restart rule:** at any time, we can replace our current goal with a previous one.[3]

Note that when using the rule for implication in case the antecedent is a conjunction it has to be split up into its conjuncts before being moved to the data formulas.

In classical propositional logic, when using the rule for atoms we can delete the implication we have used from the data.

Examples. We give two examples here. The formula $A \vee \neg A$ is known as *tertium non datur*. If we rewrite it in ready for computation form we get $(A{\Rightarrow}\bot) \wedge A{\Rightarrow}\bot$. Hence, the proof on the left-hand side shows the argument $\vdash A \vee \neg A$.

Next consider the following argument: $A \vee B, \neg B \vdash A$. We first translate into ready for computation form and then give a proof (right-hand side).

$$\vdash \quad (A{\Rightarrow}\bot) \wedge A{\Rightarrow}\bot$$

rule for $\Rightarrow$

$$A{\Rightarrow}\bot, A \quad \vdash \quad \bot$$

rule for atoms

$$A{\Rightarrow}\bot, A \quad \vdash \quad A$$

success

$$(A{\Rightarrow}\bot){\Rightarrow}B, B{\Rightarrow}\bot \quad \vdash \quad A$$

rule for atoms with $B{\Rightarrow}\bot$

$$(A{\Rightarrow}\bot){\Rightarrow}B, B{\Rightarrow}\bot \quad \vdash \quad B$$

rule for atoms

$$(A{\Rightarrow}\bot){\Rightarrow}B, B{\Rightarrow}\bot \quad \vdash \quad A{\Rightarrow}\bot$$

rule for $\Rightarrow$

$$(A{\Rightarrow}\bot){\Rightarrow}B, B{\Rightarrow}\bot, A \quad \vdash \quad \bot$$

restart

$$(A{\Rightarrow}\bot){\Rightarrow}B, B{\Rightarrow}\bot, A \quad \vdash \quad A$$

success

[3] Most of the rules are quite straight-forward to understand once you are familiar with the meaning of the logical connectives. The restart rule, however, is a little more subtle. To appreciate that it really is a valid rule consider the following informal explanation. We know that for any A either A or $\neg A$ holds. Therefore, in order to show $\mathcal{P} \vdash A$ we could show $\mathcal{P}, A \vdash A$ and $\mathcal{P}, \neg A \vdash A$ instead, the former of which directly succeeds. This means that for any A that becomes the goal formula in the process of a proof we could add $\neg A$ to the data. Hence, from $\mathcal{P} \vdash A$ we could step to $\mathcal{P}, \neg A \vdash A$; and several steps later we might get stuck at $\mathcal{P}^*, \neg A \vdash B$. Now, if we restart A and then $\mathcal{P}^*, \neg A \vdash A$ succeeds this means that we can deduce both A and $\neg A$ from $\mathcal{P}^*, \neg A$. In other words, the data is inconsistent and therefore $\mathcal{P}^*, \neg A \vdash B$ holds.

4 Goal-directed Deduction for FOL

The goal-direction system for first-order predicate logic works similarly to the propositional case. In first-order proofs it is usually more convenient to interleave proof steps and rewriting steps, in particular as we shall also use some special rewriting rules for quantified formulas.

We start by giving two further equivalences for translating formulas into ready for computation form. Next we will discuss the concept of universal and existential variables and based on this discussion define rewriting of formulas with free variables and Skolemization. Furthermore, before giving some example proofs we will also introduce unification.

Translation into ready for computation form. On top of the four equivalences given in the previous section we may also use the following two in order to get rid of negation symbols in front of quantifiers:

$$\begin{array}{rcl} \neg\forall x.P(x) & \equiv & \exists x.\neg P(x) \\ \neg\exists x.P(x) & \equiv & \forall x.\neg P(x) \end{array}$$

Alternatively, we could also use $\neg A \equiv A \Rightarrow \perp$, but often it will be more convenient to translate the negation inside the quantification.

Universal and existential variables. The symbol $\forall$ is called the *universal quantifier*, the symbol $\exists$ is called the *existential quantifier*. If we are given a formula $\forall x.P(x)$ this means that for all possible values of the universal variable x the predicate $P(x)$ holds. Therefore, if we want to use the formula to show some other formula, we can replace it with $P(u)$, where u is a free variable and substitute that variable as we wish.

Analogously, $\exists x.P(x)$ means that there exists a value for x which makes $P(x)$ true. Hence, when we want to use $\exists x.P(x)$ in a proof we are not allowed to choose just any value; all we know is that there is at least one such value. If $\exists x.P(x)$ is a closed formula (that is, if it has no free variables) this can be taken account of by substituting the variable x with some *new* constant symbol. This process is called *Skolemization.* In the general case, however, $\exists x.P(x)$ might contain free variables. Then the value of x that makes $P(x)$ true could depend on the choices made for those free variables. Therefore, we cannot simply Skolemize with a new constant symbol. Instead we have to introduce a term which is guaranteed to be new and whose value depends on the free variables appearing in the formula. A term whose functor is a new function symbol and whose arguments are the free variables in question does satisfy these requirements. Note that this general formulation of Skolemization also applies to the case where there are no free variables: constants are in fact 0-ary functions.

For goal-directed proofs, what has just been said applies to data formulas, but not to the goal to be shown! Why is this so? For example, if we want to show $\forall x.P(x)$, we have to be able to show it for any given arbitrary value. This is exactly the same situation as with the Skolemization of existentially quantified variables in the data. If on the other hand we want to show $\exists x.P(x)$ we can pick the particular x for which we want to provide a proof by ourselves. This again corresponds to the free variables used when rewriting universally quantified data formulas. To summarise, universally and existentially quantified variables in the data and the goal are dual in the sense that whatever we do with a variable bound by $\forall$ ($\exists$) in the data we would also do if it was bound by $\exists$ ($\forall$) in the goal.

Rewriting with free variables. From this discussion we conclude that we need to define two rewriting rules for the introduction of free variables, one for data formulas and one for the goal formula.

1. If we encounter a formula of the form $\forall x.P(x)$ in the data we can add $P(x')$ to the data with a free variable x'. The variable name x' should be different from all other variable names used in the proof (although strictly speaking it doesn't have to). This rewriting step can be repeated arbitrarily often, each time with another free variable.[4]

2. If the goal formula has the form $\exists x.P(x)$ we can rewrite it as $P(x')$ with a free variable x'. Again, the variable name x' should not be in use already. Every time we restart $\exists x.P(x)$ we can choose a new free variable x' in the consecutive rewriting step.

Skolemization. The process of instantiating existential variables is known as Skolemization. Again, we have to distinguish two cases:

1. If a formula of the form $\exists x.P(x)$ with free variables $y_1, \ldots, y_n$ can be found in the data, we can Skolemize it by replacing the formula with $P(f(y_1, \ldots, y_n))$ for a new function symbol f (i.e. f is not allowed to appear anywhere in the proof so far). The old formula $\exists x.P(x)$ should be deleted from the data as it won't be useful anymore.

2. If the goal formula has the form $\forall x.P(x)$ (with free variables $y_1, \ldots, y_n$) we can replace it with $P(f(y_1, \ldots, y_n))$ for a new function symbol f. The old goal $\forall x.P(x)$ should not be restarted; it won't help for the proof to succeed.

As an example, suppose a logical argument with $\exists x.(A(x,y) \wedge B(k))$ in the data (where y is a free variable and k a constant symbol). Then we can replace that formula in the next proof step with $A(f(y), y) \wedge B(k)$, where f is the new Skolem function.

If the formula we wish to Skolemize does not contain any free variables (and this is indeed the case for many of the examples in the textbook) then we simply replace the existential variable with a new *constant* symbol.

[4] In example proofs we sometimes delete $\forall x.P(x)$ after rewriting, but we should only do this, if we do not plan to use it again with a new variable name later on in the proof.

Unification and variable substitution. The goal-directed rules for first-order logic are essentially the same as for propositional logic. Actually, if we only consider formulas that are not quantified and don't have any free variables, they are even exactly the same. We have seen already how to deal with quantifiers (namely by either rewriting formulas with free variables or by Skolemization), so we are left with the free variables.

Suppose your current goal was $A(k)$ and there was a formula $A(x)$ in the data (assume x is a free variable and k a constant symbol). As x is 'free' it can take any value, including k. So, if we were to *substitute* x with k, written as $\{x = k\}$, then we would have $A(k)$ also in the data and would succeed in the next proof step.

Hence, at any one time during a proof we can pick a free variable appearing somewhere in the data or the goal and substitute it with any constant symbol or term we like. (Now it should become clear why it is crucial to choose different names for free variables every time we choose a new one: otherwise the previously independent variables would suddenly be bound to have the same value, which in most cases is not what we want.)

If we have used the rule for conjunction before that and the free variable x we substitute appears in both sub-proofs, the variable instantiation has to be carried over to the other sub-proof.

Sometimes it is not even necessary to substitute a variable with a fixed value, for example, if $A(x)$ appeared in the data and $A(y)$ was the goal (for free variables x and y). In that case it is enough to *unify* the two variables, denoted $\{x = y\}$. After such a unification the two variables are treated as being the same, in particular any further unifications or substitutions made for x will also apply to y and vice versa. Again, if we are working in a sub-proof unification might also affect the sibling sub-proof.

Unification and variable substitution affect the rule for atoms and the success rule. If we wish to use one of those rules in the next proof step we first carry out the necessary unification/substitution as an extra step before applying the respective proof rule.

Examples. We exemplify what has been said by giving some sample proofs. We use $u, v, w, \ldots$ to denote free variables,[5] $k, l, \ldots$ for Skolem constants, and $f, g, \ldots$ for Skolem functions.

For the first example we shall attempt to prove $\forall x.\exists y.A(x, y) \vdash \exists y.\forall x.A(x, y)$. This is actually not a valid argument. One way of showing this would be to give a countermodel. If, for example, we interpret the predicate A as the relation $<$ over the domain of the real numbers, then the argument's premise $\forall x.\exists y.A(x, y)$ does hold in that model, because for every number x there exists a number y such that $x < y$. The conclusion $\exists y.\forall x.A(x, y)$, however, is not validated by that model: there is no number y such that all numbers x are smaller than y.

Another way of proving that the argument is not valid is to show that a goal-directed proof will not succeed.

[5] Do not mix up bound and free variables: if, for example, we have $\forall x.A(x)$ and $A(x)$ in the data, then in the first case x is a bound variable and has nothing to do with the free variable x appearing in the second formula $A(x)$. In the examples here we have avoided using, say, x for both bound and free variables, but for some of the examples in the textbook this is different.

$$\forall x.\exists y.A(x,y) \quad \vdash \quad \exists y.\forall x.A(x,y)$$

rewrite data with free variable

$$\exists y.A(u,y) \quad \vdash \quad \exists y.\forall x.A(x,y)$$

Skolemize data

$$A(u,f(u)) \quad \vdash \quad \exists y.\forall x.A(x,y)$$

rewrite goal with free variable

$$A(u,f(u)) \quad \vdash \quad \forall x.A(x,v)$$

Skolemize goal

$$A(u,f(u)) \quad \vdash \quad A(g(v),v)$$

The proof fails, because it is not possible to unify u with $g(v)$ and $f(u)$ with v at the same time. Note that in this example it would not have been enough just to use Skolem constants. Indeed, if we had Skolemized with constants k and l instead of $f(u)$ and $g(v)$ the unification would have been possible at the end, i.e. the proof would have succeeded – a wrong result!

The next proof is for the argument $\forall x.(A(x,x)\Rightarrow\bot) \vdash \forall x.\exists y.(A(x,y)\Rightarrow B(x))$.

$$\forall x.(A(x,x)\Rightarrow\bot) \quad \vdash \quad \forall x.\exists y.(A(x,y)\Rightarrow B(x))$$

rewrite data with free variable

$$A(u,u)\Rightarrow\bot \quad \vdash \quad \forall x.\exists y.(A(x,y)\Rightarrow B(x))$$

Skolemize goal

$$A(u,u)\Rightarrow\bot \quad \vdash \quad \exists y.(A(k,y)\Rightarrow B(k))$$

rewrite goal with free variable

$$A(u,u)\Rightarrow\bot \quad \vdash \quad A(k,v)\Rightarrow B(k)$$

rule for $\Rightarrow$

continues next column

$$\begin{array}{l} A(u,u)\Rightarrow\bot \\ A(k,v) \end{array} \quad \vdash \quad B(k)$$

rule for atoms

$$\begin{array}{l} A(u,u)\Rightarrow\bot \\ A(k,v) \end{array} \quad \vdash \quad A(u,u)$$

substitution $\{u = v = k\}$

$$\begin{array}{l} A(k,k)\Rightarrow\bot \\ A(k,k) \end{array} \quad \vdash \quad A(k,k)$$

success

The final example is a proof for the argument $\forall x.(A(x) \vee B(x)), \neg\exists x.A(x) \vdash \forall x.B(x)$. It sometimes helps to think about an argument first ("do you believe it?") before actually trying to prove it formally. This one clearly should succeed, because if we know that either $A(x)$ or $B(x)$ is true for all x and at the same time there is no x such that $A(x)$, then the first statement can only be true, if indeed $B(x)$ holds for all x.

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & \forall x.B(x) \\ \neg\exists x.A(x) & & \end{array}$$

Skolemize goal

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & B(k) \\ \neg\exists x.A(x) & & \end{array}$$

rewrite data with free variable

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & B(k) \\ A(u) \vee B(u) & & \\ \neg\exists x.A(x) & & \end{array}$$

rewrite data

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & B(k) \\ (A(u)\Rightarrow\bot)\Rightarrow B(u) & & \\ \forall x.\neg A(x) & & \end{array}$$

rewrite data with free variable

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & B(k) \\ (A(u)\Rightarrow\bot)\Rightarrow B(u) & & \\ \neg A(v) & & \end{array}$$

rewrite data

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & B(k) \\ (A(u)\Rightarrow\bot)\Rightarrow B(u) & & \\ A(v)\Rightarrow\bot & & \end{array}$$

substitution $\{u = k\}$

continues next column

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & B(k) \\ (A(k)\Rightarrow\bot)\Rightarrow B(k) & & \\ A(v)\Rightarrow\bot & & \end{array}$$

rule for atoms

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & A(k)\Rightarrow\bot \\ (A(k)\Rightarrow\bot)\Rightarrow B(k) & & \\ A(v)\Rightarrow\bot & & \end{array}$$

rule for $\Rightarrow$

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & \bot \\ (A(k)\Rightarrow\bot)\Rightarrow B(k) & & \\ A(v)\Rightarrow\bot & & \\ A(k) & & \end{array}$$

rule for atoms

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & A(v) \\ (A(k)\Rightarrow\bot)\Rightarrow B(k) & & \\ A(v)\Rightarrow\bot & & \\ A(k) & & \end{array}$$

substitution $\{v = k\}$

$$\begin{array}{lcl} \forall x.(A(x) \vee B(x)) & \vdash & A(k) \\ (A(k)\Rightarrow\bot)\Rightarrow B(k) & & \\ A(k)\Rightarrow\bot & & \\ A(k) & & \end{array}$$

success

Of course, we don't have to make every single rewriting an extra proof step, but may also do several of them within one step.