

Graph Rewriting for Natural Deduction
and the Proper Treatment of Variables

Willem Heijltjes

Master's thesis

Thesis supervisors:

Prof. Dr. Albert Visser
Dr. Vincent van Oostrom

Department of Philosophy, Utrecht University

2007

Preface

This thesis has been over one year in the making. Well over four times the allotted time, its size has grown to match, reaching four times the minimum size requirement. This to the horror of my supervisors, Albert Visser and Vincent van Oostrom, who had to read countless intermediate versions of the document, of which sometimes not a single letter made it to the final version.

Meetings with either of my supervisors usually followed the same outline:

—Willem: Good afternoon! Did you have a chance to read what I sent you last night/week/month?

—Albert/Vincent: I'm afraid not, but I was just about to. Maybe you can explain things as we read along?

—Willem: Well, in short, I've come across such-and-such a problem, and I found solutions A and B. Solution B is a little more elegant, but also requires an overhaul of most of what I've written so far.

—Albert/Vincent: Yes, I prefer solution B as well. Until next time, then!

Despite their simple format meetings could last several hours, during which we discussed many problems, solutions and philosophical implications, mostly to do with retrieving the email in which I sent the latest documents.

If I was well-prepared and supervision meetings were light-hearted, that was mainly because my preceding education by both Albert and Vincent has been downright excellent. During the last few years I took several of their courses, and I dare say that all of the skills I learned there were used in the making of this thesis. Wherever I go next, if it's not the Philosophy Department in Utrecht, I will surely miss Vincent's witty jokes, which I practically always fell for, and Albert's heart-warming optimism, which generally far exceeded my own where my thesis was concerned.

During the writing process I received support from countless other people, mostly in the form of a comment on the aesthetics of my pictures, and there are two people in particular whom I would like to thank. The first is my brother Bart, who corrected the grammar, spelling and style of the introduction and the first chapter; we guessed that anyone to get past that point is mainly interested in formulas anyway. The other is my girlfriend Saskia, who, besides everlasting love, provided me with the one thing that made this thesis possible: deadlines.

The public version of my thesis, the one before you, has as additions over the submitted version this preface, and the illustrated cover pages. Also, some minor errors in the appendices have been fixed.

Contents

Introduction	6
1 Natural Deduction	9
1.1 Building proofs	9
1.2 Proof rewriting	12
1.3 Free and bound variables	13
1.4 Proper variables	16
1.5 Substitution rules	18
2 Proof Graphs	19
2.1 The propositional fragment	19
2.2 Constructing and interpreting graphs	22
2.3 Bisimulation	24
2.4 Backpointers	25
2.5 The closing of assumptions	27
2.6 The first-order fragment	29
2.7 Substitution	31
2.8 Binding of proper variables	33
2.9 Dead terms	36
2.10 Rewriting	40
2.11 Translating proofs to graphs	45
3 Formal Definitions	51
3.1 Constraint definition	51
3.2 Pre-proof graphs	57
3.3 Binding in pre-proof graphs	60

4	Transformations	64
4.1	The <i>trans</i> operation	65
4.2	The <i>merge</i> operations	71
4.3	Adding and removing backpointers	76
4.4	The <i>read</i> operation	80
5	Graph Rewriting	84
5.1	Rewrite steps	84
5.2	Confluence	87
6	Correctness and Completeness	90
6.1	Building a proof graph	91
6.2	Reversibility of the translation	93
6.3	Reading back a proof graph	96
7	Conclusions	99
7.1	Further investigations	100
	Bibliography	101
	Appendices	102
	Appendix A: Natural deduction schemes	102
	Appendix B: Inference schemes for proof graphs	105
	Appendix C: Rewrite schemes for proof graphs	111
	Appendix D: Proof-to-graph translation schemes	116

Introduction

The name ‘natural deduction’ has, over the years, probably left many a student wondering what on earth is so natural about these deductions. Usually the first proof system taught, the student has to learn about open assumptions, closed assumptions, and variables that may or may not occur in specific locations, not to mention the peculiarities of some of the elimination rules, while still struggling with the concept of creating formal proofs.

Any confusion about the name will abruptly have ended the moment the student was confronted with Hilbert-style or combinatorial proofs. Indeed, the development of natural deduction in the 1930s was a response to the cumbersome nature of the contemporary proof systems developed by Frege, Russell, Hilbert and Heyting. The problem with those proof systems was that logicians and mathematicians used a very different style of reasoning to obtain informal proofs of their statements. Because of the differences between informal reasoning and the proof systems of the time, if one was to formally prove a statement, one essentially had to start from scratch.

Determined to amend this, Gerhard Gentzen presented natural deduction in his 1935 paper¹ in the form of the calculi NJ and NK, for intuitionistic and classical logic respectively. Characteristic of the informal mathematical reasoning on which it is based is the use of assumptions. This is illustrated by the example Gentzen gives, a proof of distribution of disjunction over conjunction:

“Suppose that either X or $Y \& Z$ holds. We distinguish the two cases: 1. X holds, 2. $Y \& Z$ holds. In the first case it follows that $X \vee Y$ holds, and also $X \vee Z$; hence $(X \vee Y) \& (X \vee Z)$ also holds. In the second case $Y \& Z$ holds, which means that both Y and Z hold. From Y follows $X \vee Y$; from Z follows $X \vee Z$. Thus $(X \vee Y) \& (X \vee Z)$ again holds. The latter formula has thus been derived, generally, from $X \vee (Y \& Z)$, i.e., $(X \vee (Y \& Z)) \supset ((X \vee Y) \& (X \vee Z))$ holds.”²

Despite its relative user-friendliness, natural deduction initially hardly caught on. In the same paper, Gentzen had presented another proof system, whose variations are today known as ‘sequent calculi’ or ‘Gentzen systems’. These systems outperformed natural deduction where classical logic was concerned and garnered all the attention, while natural deduction threatened to fade into obscurity. Remaining underground during the thirties, its gradual emergence thereafter was mainly due to its suitability as an introduction to formal proof systems.

The rise of computers created an interest in the properties of proof systems

¹Gentzen [1935]

²from: Gentzen [1969]

themselves. The correspondence between proofs and computer programs, expressed in the Curry-Howard isomorphism, turned out to be particularly precise for intuitionistic logic, more so than for classical logic, sparking a renewed interest in the former. In 1965 Dag Prawitz developed proof normalisation for natural deduction³. The more or less similar concept for sequent calculi called ‘cut elimination’ had already been present in the original paper by Gentzen, but until then there had been no equivalent for natural deduction. This eliminated the last fundamental advantage of sequent calculi over natural deduction, and although sequent calculi are still considered the better choice for dealing with classical logic, today both are widely studied and taught.

Another field that brought renewed attention to formal proof systems in general is that of computational (or formal) linguistics. The main aim being to obtain a formal representation of natural language, natural deduction has featured in many attempts at this.

Somewhat more recent works approach natural deduction from a graph perspective⁴, mostly to address the complexity of the proof normalisation process. In general those graph representations have ignored the formulas in a proof. This thesis will present a version of natural deduction in which both a proof and the formulas within it are represented in a single graph.

The main motive to include formulas in this approach was to get a better view on the behaviour of variables inside a proof. In most versions of natural deduction, half of the rules that deal with variables carry explicit restrictions with them on where those variables may or may not occur. Although from a technical perspective this is at most inconvenient, the question is whether there are more fundamental reasons behind the need for these restrictions.

In the hope of being able to present a more elegant proof system, in this thesis a graph system for natural deduction is constructed, with the correct treatment of variables as the main premise. Whether in the end that will be considered a success may turn out to be a matter of taste, especially since what the ‘correct’ treatment of variables exactly entails is itself very debatable.

The design of the graph system brings a lot of choices with it. The arguments to make those choices will be technical, but sometimes a little philosophical as well. This is not at all strange to works on logic: often, there will be talk of structures being ‘essentially the same’, appealing to some unspecified intuition on identity of formal constructs. This thesis will be no stranger to that phrase, but it will attempt to provide grounds for that intuition whenever possible.

On the other side, the technical specifications of graphs can be quite elaborate, and are often not that interesting. The informal description and the formal treatment of the graphs have therefore been separated as much as possible. The

³see: Prawitz [1965]

⁴see e.g. Statman [1974]

first chapter will introduce the reader to a standard case of natural deduction for intuitionistic logic, with the indication of possible areas of improvement, while the second chapter will elaborately describe the design of the proof graph system. Both these chapters refrain from giving formal specifications, so as not to hinder the casual reader.

The remaining chapters will give the formal details. In chapter three, proof graphs are formally described as a type of graph meeting certain specifications. Chapter four focuses on how to turn a natural deduction proof into such a graph, and the other way around. Chapter five presents the graph versions of the rewrite steps for natural deduction. Finally, chapter six contains the proofs that show that proof graphs can correctly represent all natural deduction proofs, and vice versa, that all proof graphs represent a valid natural deduction proof.

The results in this thesis are relevant to several areas of the field of cognitive artificial intelligence. Firstly, in the area of computational linguistics a host of formal systems is used, of which several are based on natural deduction. Moreover, many of the other systems face problems related to the presented work. For instance, Discourse Representation Theory uses a semi-formal method to process sentences for which ordinary first-order logic fails due to limitations on variables. Research into the behaviour of variables in formal systems is relevant to this theory and to other linguistic models, as is the investigation into implicit underlying structures within natural deduction.

Also for any area of research connected with data representation, this thesis may prove helpful. As a concrete study into sharing, data structures related to formal systems may benefit from a more compact representation method. Regarding the area of logic itself, the thesis is a study into fundamental representations, and as such results may be applicable to other proof systems, or in fact any representational structure.

1 Natural Deduction

In the first two sections of this chapter a version of natural deduction will be presented that will serve as a basis for developing proof graphs. This version will differ from the one given in Troelstra and Schwichtenberg [1996] only in notation; there will be no difference in the functioning of the rules. We will only be concerned with natural deduction for intuitionistic logic.

The next two sections will focus on the treatment of variables. Some examples will show that this is not optimal in natural deduction. We will give a suggestion for improvement, to be tested in graphs later on. The final section will briefly comment on substitution.

1.1 Building proofs

First of all, the notation has to be explained. The scheme used for naming formulas, variables and the like should be familiar:

- A, B and C are arbitrary formulas,
- P, Q and R are predicates,
- x, y and z are bound variables,
- a, b and c are free variables,
- u, v and w are assumption markers, and
- t is a term

When more than three of a kind are needed, we will use primes ($'$). Furthermore, our intuitionistic logic consists of the binary operators \wedge, \vee and \rightarrow , the quantifiers \forall and \exists and the symbol for contradiction, \perp . Negation of a formula A is expressed by $A \rightarrow \perp$.

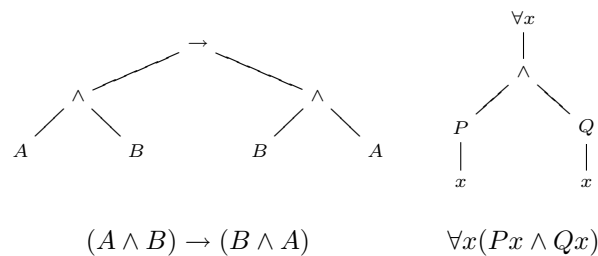


Figure 1.1: Formulas depicted as formula trees

Formulas are built inductively; larger formulas are made by joining smaller formulas with a connective or by adding a quantifier (with a variable) to a formula. Predicates with a number of arguments equal to their arity, which may be zero, are the basic components of a formula. The last connective or quantifier added to a formula is called the *primary* connective. This is the connective that the inference rules of natural deduction operate on. The way a formula is built up is more pronounced when it is presented as a *formula tree*, of which some examples are shown in Figure 1.1; the primary connective of the formula is the root of the (upside-down) tree.

Deductions are formed by introducing a new or duplicate assumption, which is simply a formula with a marker, or by expanding one or more deductions in accordance with the inference rules shown in Figure 1.2. Assumptions with different formulas must have different markers as well. Following these rules, deductions take the form of trees, starting with one or more assumptions as the leaves but having only one conclusion as the root.

$$\begin{array}{c}
\vdots \\
\frac{}{A^u} \quad \frac{\perp}{A} (\perp E) \quad \frac{[A]^u \quad \vdots}{A \rightarrow B} (\rightarrow I, u) \quad \frac{\vdots \quad \vdots}{A \rightarrow B \quad B} (\rightarrow E) \\
\\
\frac{\vdots \quad \vdots}{A \quad B} (\wedge I) \quad \frac{\vdots}{A \wedge B} (\wedge EL) \quad \frac{\vdots}{A \wedge B \quad B} (\wedge ER) \\
\\
\frac{\vdots}{A} (\forall IL) \quad \frac{\vdots}{A \vee B} (\forall IR) \quad \frac{\vdots \quad [A]^u \quad [B]^v}{A \vee B \quad C \quad C} (\vee E, u, v) \\
\\
\frac{\vdots}{\forall x.A} (\forall E) \quad \frac{\vdots}{A[a/x]} (\forall I) \quad \text{Restrictions on } \forall I: a = x \text{ or } \\
\text{\scriptsize } a \text{ is not free in } A; a \text{ is not free} \\
\text{\scriptsize in open assumptions} \\
\\
\frac{\vdots}{\exists x.A} (\exists I) \quad \frac{\vdots \quad [A[a/x]]^u}{\exists x.A \quad C} (\exists E, u) \quad \text{Restrictions on } \exists E: a = x \text{ or } \\
\text{\scriptsize } a \text{ is not free in } A; a \text{ is not free} \\
\text{\scriptsize in } C \text{ or in open assumptions} \\
\text{\scriptsize except } [A[a/x]]^u
\end{array}$$

Figure 1.2: The inference rules of natural deduction

Assumptions are open when they are introduced, but may be closed by instances of the rules $\rightarrow I$, $\forall E$ and $\exists E$ (see Figure 1.2). When these rules are applied, an

identify a deduction with its last inference, when we refer to the minor premise of a deduction, and we identify a deduction with its conclusion when we only refer to the minor premise, while we mean the entire deduction leading up to it as well.

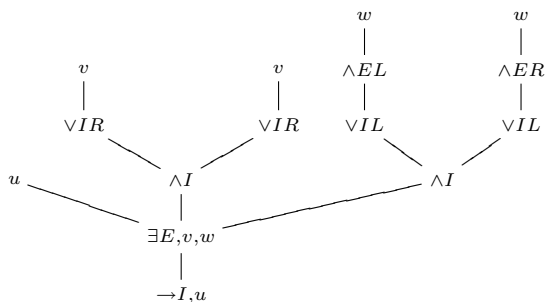


Figure 1.4: Rule tree of the example proof

One of the tools we will use to investigate deductions is what we call *rule trees*. These are trees that show the rule applications used in a deduction, but omit the formulas. Figure 1.4 shows the rule tree of the proof in Figure 1.3.

1.2 Proof rewriting

Natural deduction has both introduction and elimination rules for each connective. When a proof is built from these rules, there is no guarantee that there isn't a more direct proof that reaches the same conclusion. Connectives may be introduced and eliminated at will, possibly without moving any closer to the desired conclusion.

As it turns out, the derivations that follow the most direct route from their assumptions to their conclusion follow a particular scheme. Within these deductions, called *normal* deductions, formulas are first broken down into components by elimination rules and after that reassembled by introduction rules. This sorting of the inferences of a deduction into an elimination-part and an introduction-part is a little obscured by the form of the schemes for disjunction elimination and existential quantifier elimination. We think of these inference schemes as if the closed assumptions were directly below the major premise, which is where the disjunction or existential quantifier is actually removed; nothing happens between the minor premises and the conclusion. The example proof of Figure 1.3 is a normal proof, even though there are introduction rules above the minor premises of the disjunction elimination.

To obtain a normal deduction from any deduction, we transform it step by step, until no elimination rule is below an introduction rule. This process is

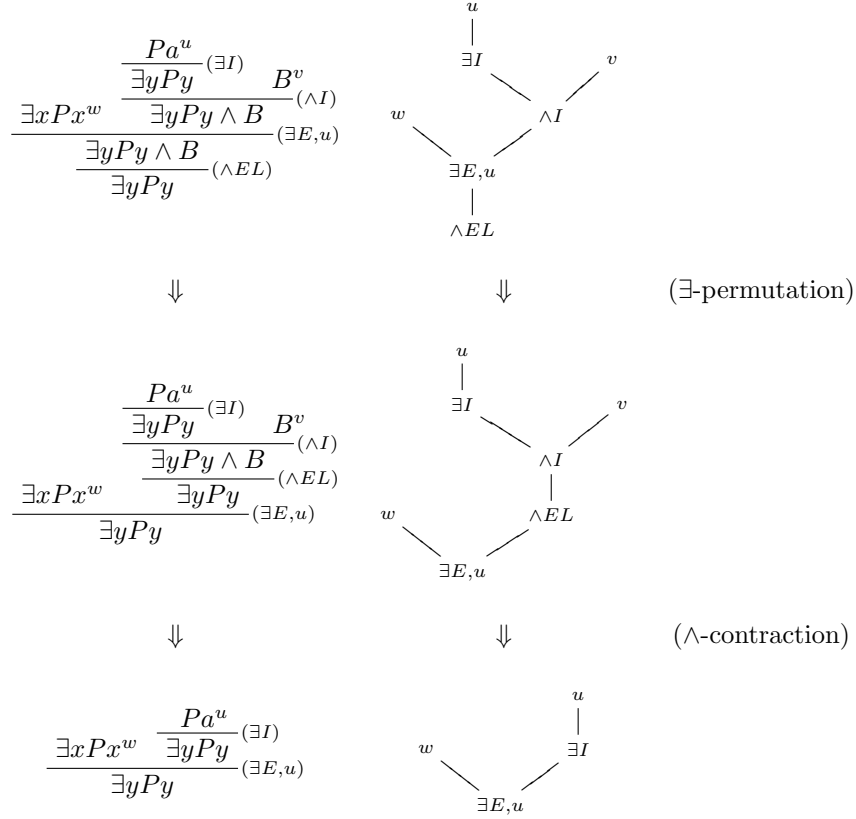


Figure 1.5: An example of a permutation and a contraction

called *normalization*. There are three kinds of transformations: contractions, permutations and simplifications. Contractions are used to remove a consecutive introduction and elimination of the same connective. Simplifications remove parts of a proof that are unused, which occurs when a closed assumption class of an $\exists E$ - or $\forall E$ -application is empty. Permutations are used to shift $\forall E$ - and $\exists E$ -applications down over elimination rules to expose other contractions. Figure 1.5 shows a permutation and a contraction in action, the formal descriptions of the normalization rules can be found in Appendix A.

1.3 Free and bound variables

This section will deal with variables, which will be addressed from both a technical and a philosophical perspective. The issue is this: the variables a , b , x , y and so on that we use in formulas, are really not the variables themselves, but only names for variables, called *variable letters*. The reason this difference is

noticeable is that a variable letter does not uniquely identify a variable.

First we look at bound variables. Observe that the formulas $\forall xPx$ and $\forall yPy$ essentially mean the same thing: ‘ P holds for everything’. The interpretation shows that the choice of variable letter does not affect the meaning of the formula. The way we see this philosophically, is that the identity of the variable is not determined by the letter, but by the quantifier; the letter is simply a necessary marker to link the variable to the quantifier.

Technically the similarity of $\forall xPx$ and $\forall yPy$ is called α -equivalence: two formulas are α -equivalent if they are the same up to the names of their bound variables. This is reflected by the fact that substitution of bound variables is generally considered valid.

As a note on the side, the current context of natural deduction is actually a poor choice for a discussion of α -equivalence: within a natural deduction proof it is generally not allowed to replace a formula with an α -equivalent one.

In many respects, the semantic properties of the logical connectives are determined by the inference rules. Take for instance the symmetry of conjunction: although $(A \wedge B) \rightarrow (B \wedge A)$ is easily proved within natural deduction, a formula $(A \wedge B)$ may not be replaced by $(B \wedge A)$. Likewise, $(\forall xPx) \rightarrow (\forall yPy)$ is a tautology, but bound variables may not be substituted within a proof.

Despite these considerations, the general idea is that α -equivalence shows that variable letters and variables are not the same thing, since bound variables may be renamed without affecting the meaning of the formula.

Another, related point is that the same variable letter may be used for different variables within the same formula, as in for example $(\forall xPx) \wedge (\forall xQx)$, or even in a nested situation such as $\forall x(Px \wedge \forall xQx)$. The different occurrences of x denote different variables, since they are bound by different quantifiers and may be substituted independently.

In the following paragraphs we will apply these insights to assumption markers and free variables on the level of an entire proof, rather than a single formula. The focus will be on the technical side, since intuitions on proofs may not be as strong as those on formulas. Our primary method will be to find out which substitutions yield valid deductions.

$$\frac{A^u}{A \rightarrow A} (\rightarrow I, u) \quad \frac{A^v}{A \rightarrow A} (\rightarrow I, v) \quad \frac{\frac{A^u}{A \rightarrow A} (\rightarrow I, u) \quad A^u}{A} (\rightarrow E) \quad \frac{A}{A \rightarrow A} (\rightarrow I, u)$$

Figure 1.6: Assumption markers

Figure 1.6 illustrates the striking similarity between variables and assumption

markers: α -equivalence exists for assumption markers too, as the first two proofs show. The third proof illustrates that the same marker may be used for different assumption closures.

Formally, which variable is bound by which quantifier, and which assumption is bound by which rule application, is defined by the construction of the formula or deduction. This is best shown with rule trees and formula trees: a variable is bound by the first quantifier with the same letter above it; an assumption is closed by the first $\rightarrow I$ -, $\forall E$ - or $\exists E$ -application with the same marker, found when moving down the rule tree. Since the variable bound by a quantifier does not have to appear in the formula at all, it is not possible to create ‘illegal’ formulas—although one may of course be left with unwanted free variables. We will come back to this later, when we need to apply the restrictions that follow from the definition of assumption binding, to graphs.

We will now focus on the free variables within a proof. Free variables may not be renamed, because the letter of a free variable generally denotes the same variable throughout different formulas within a certain environment, such as a proof. For instance, when Pa and Qa occur as premises of a conjunction introduction, concluding $Pa \wedge Qa$, the letter a denotes the same variable in both premises as well as the conclusion—or at least, renaming it in one but not the other formulas yields an invalid inference, whereas renaming it in all three formulas does not. The question is, how to find out whether occurrences of a variable letter in totally different parts of a proof denote the same variable.

$$\frac{\frac{\frac{\vdots^{(1)}}{A}(\forall I)}{\forall x.A[x/a]}(\forall E)}{A[t/a]} \Rightarrow \frac{\frac{\vdots^{(1)}}{A}}{A} \} [t/a]$$

Figure 1.7: \forall -contraction

It is not so that a free variable may not be renamed at all. Consider the rule for \forall -contraction in Figure 1.7. What has happened on the right side, is that in the entire deduction leading up to A , a is replaced by t . Consider also the two proofs shown in Figure 1.8, that differ only in the name of the free variable.

$$\frac{\frac{Pa^u}{Pa \rightarrow Pa}(\rightarrow I, u)}{\forall x(Px \rightarrow Px)}(\forall I) \quad \frac{\frac{Pb^u}{Pb \rightarrow Pb}(\rightarrow I, u)}{\forall x(Px \rightarrow Px)}(\forall I)$$

Figure 1.8: Are these proofs α -equivalent?

The suggestion rising from Figure 1.7 and Figure 1.8 is that on the level of a proof, the proper variable of a $\forall I$ -application becomes bound. The next paragraph will show some characteristics of natural deduction that point in the

same direction.

1.4 Proper variables

As was pointed out in the previous section, bound variables cannot be identified by their letter alone, since that letter may be used by other variables as well. When looking at a proof as a whole, some free variables suffer from the same problem, as will be demonstrated with the help of Figure 1.9.

$$\frac{\frac{\frac{\forall xPx \wedge Qa^u}{\forall xPx} (\wedge EL)}{Pa} (\forall E)}{\forall yPy} (\forall I) \quad \frac{\frac{\frac{Pa^v}{\exists yPy} (\wedge EL)}{\exists xPx^u} (\wedge I)}{\exists yPy \wedge Qa} (\exists E, v)}$$

Figure 1.9: Some inference rules are too strict

The applications of $\forall I$ and $\exists E$ in the examples of Figure 1.9 are not allowed, because the proper variable, a in both cases, is free in an open assumption, $\forall xPx \wedge Qa^u$ and Qa^w respectively. However, it is not possible to derive a contradiction from these proofs—these are in essence valid inferences. This imperfection is widely recognized, but also quickly dismissed as it does not prohibit us from proving anything: both deductions in Figure 1.9 could just as well have a b as the proper variable, while the free variable a in the assumptions remains in place. The question rises, then, whether the occurrences of the letter a denote the same variable throughout these deductions.

$$\frac{\frac{\frac{\dots}{Qa} (\forall I)}{\forall yQy} (\forall I)}{\frac{\frac{Pa}{\forall xPx} (\forall I)}{Pt} (\forall E)} \Rightarrow \frac{\frac{\dots}{Qt} (\forall I)}{\forall yQy} (\forall I)$$

Figure 1.10: A contraction gone bad

Real problems arise when we take a closer look at normalization. Figure 1.10 shows an abstracted \forall -contraction where the right side ends up with an illegal $\forall I$ -application (universal conclusions may only be drawn from free variables, not terms). Of course, this is because the a in the upper part should not be renamed as it has nothing to do with the a in the lower part, but the inference rules blindly operate on variable letters.

Two different measures to amend this will be discussed here. The first and simplest is to declare the variable a not to be free in the subproof of an inference of which it is the proper variable. This would imply that a is not free in the

upper part of the proofs in Figure 1.10, and would thus not be substituted—only free occurrences of variables participate in a substitution. The other solution is the original one devised by Prawitz.⁵

His solution was, essentially, to rename as few occurrences of a variable as possible, while still retaining a valid proof. The procedure that finds these occurrences starts at the $\forall I$ - or $\exists E$ -application whose proper variable is the one that needs to be renamed. If it is a $\forall I$ -application, the premise is marked—if the proper variable occurs in it—and if it is an $\exists E$ -application the assumptions that it closes are marked—again, only if they contain the proper variable. The marked formula may be part of an inference in three ways: it may be a premise, the conclusion or an assumption closed by the inference. For each inference in which it takes part, we mark those formulas that share a letter with the marked formula in the inference scheme—if they contain a free occurrence of the variable letter. For instance, suppose the left closed formula of a $\forall E$ -application is marked (the A in the inference scheme). Then the major premise, $A \vee B$, is marked; because it contains A in its entirety it also contains an occurrence of a . If the right major premise (the formula B) also contains a free a , that formula is marked in turn. This procedure is repeated until nothing else can be marked. If the variable letter a by that time both occurs in marked and unmarked formulas, it is replaced by a fresh one in all marked formulas simultaneously. The resulting variable letter (the a or the fresh one) is called a *pure* variable.⁶

This renaming procedure may be viewed from a purely technical perspective, in which it simply renames a proper variable while leaving all inferences valid. But since renaming is usually reserved for bound variables, it is a strong indication that also in this view the proper variable of a $\forall I$ - or $\exists E$ -application behaves like a bound variable.

$$\frac{\frac{\frac{\vdots}{Pa^u}(\exists I) \quad \frac{\frac{\frac{\vdots}{Qa} \quad \frac{\vdots}{\forall y Qy}(\forall I)}{\vdots}}{\vdots}}{C}(\exists E, u)}}{C} \Rightarrow \frac{\frac{\frac{\vdots}{Pt} \quad \frac{\frac{\vdots}{Qt} \quad \frac{\vdots}{\forall y Qy}(\forall I)}{\vdots}}{\vdots}}{C}}{C}$$

Figure 1.11: Another contraction gone bad

As a further illustration, Figure 1.11 shows that the problem found with \forall -contraction in Figure 1.10 can be reconstructed for \exists -contraction. The free variable letter a is replaced with term t , which makes the \forall -introduction invalid.

It begins to look as though applications of both \forall -introduction and \exists -elimination

⁵see Prawitz [1965]

⁶This is a quite liberal interpretation of Prawitz' formalism, that actually constructs *sequences* of formulas instead of marking them. Since deductions are trees, there is probably no specific reason to use sequences, and marking the formulas should do equally well.

actually bind the occurrences of their proper variables within their subproofs. The restrictions on those rules, which amount to: ‘after an application, the proper variable may not occur in the conclusion or in any open assumption’ make more sense in that light as well. If an open assumption in which a proper variable occurs is closed by an implication introduction, then that variable will also occur in the conclusion of the implication introduction. However, the variable occurrence in the assumption will be bound by the rule whose proper variable it is, but the variable occurrence in the conclusion of the implication introduction will be free.

1.5 Substitution rules

As the previous section illustrated, variables need to be handled with care. Before we start building graphs, another sensible but implicit restriction should be brought to the surface.

The rules for substitution in general state that, firstly, only free variable occurrences may be replaced, and secondly, no free variable in the replacement formula may become bound in the substitution. The area to which the latter specifically applies is that of the proper terms of $\forall E$ - and $\exists I$ -applications, repeated below.

$$\frac{\vdots}{\forall x.A} (\forall E) \quad \frac{\vdots}{\exists x.A} (\exists I)$$

In principle the term t may be any term, as long as it can be placed in the same positions that a variable may occur. It can be a variable, or a larger term in which many variables occur. The catch is of course, that free variables may not be inserted into a formula without checking whether they become bound. The proof in Figure 1.12 shows the consequences of ignoring the restrictions on substitution.

$$\frac{\frac{\frac{\exists y.Px^u \quad Px^v}{Px} (\exists E, v)}{(\exists y.Px) \rightarrow Px} (\rightarrow I, u)}{\forall x.((\exists y.Px) \rightarrow Px)} (\forall I)}{\frac{(\exists y.Py) \rightarrow Py}{\forall z.((\exists y.Py) \rightarrow Pz)} (\forall E)} (\forall I)$$

Figure 1.12: An incorrect proof

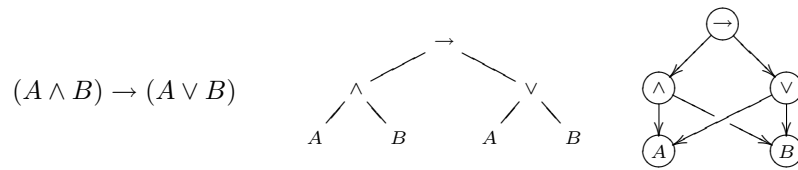


Figure 2.2: The transition from propositional formula to proof graph

and which is the conclusion, edges need to be directed. Although it seems like there is a choice to be made here, it is only an arbitrary one, since the direction of edges relative to each other is decided by the way we implement sharing.

The ‘direction’ of sharing is that rule nodes share premises and conclusions, and formula nodes share subformulas. If we were to direct edges between rule nodes towards conclusions and those between formula nodes towards subformulas, the direction of edges would be along with the direction of sharing within the formula part, but against it within the proof part of a graph. This would prevent us from finding common definitions for dealing with formula and rule nodes.

We choose to direct edges towards the shared parts, which will allow for more convenient definitions later on. That is, edges in proof graphs lead from formulas to subformulas and from rule applications to premises, the latter of which might seem counterintuitive to those who expect edges to represent the direction of inference. The edges that connect the proof part of the proof graph to the formula part lead from the rule nodes to the formula nodes.

Like rules in regular natural deduction, a rule node requires the correct number of premises. Premises of a node are indicated by outgoing edges, but the node also has an edge to a formula. Furthermore, where natural deduction rules discard assumptions, their proof graph counterparts need to do so as well. This implies that the number and function of outgoing edges should be fixed. On the other hand, in order to share subproofs multiple rule nodes must be able to share the same premise. The number of incoming edges on a rule node should therefore be arbitrary. Figure 2.3 shows the connections of an implication introduction node as an example, with the natural deduction counterpart for reference. There is a slight inaccuracy with the premise-port, which does not connect directly to the premise formula, as its name suggests, but instead connects to the rule node whose conclusion is the required premise.

To regulate the outgoing edges on a node we introduce the notion of a *port*. Each node will have a limited number of ports, and each port accommodates exactly one outgoing edge. Ports act as labels on edges, which allows edges to perform different functions in a graph; for example, as illustrated in Figure 2.3, the three ports on the $\rightarrow I$ -node are the closed assumption-port, the premise-port and the conclusion-port. The edge attached to the conclusion-port should point to a formula node that represents the conclusion formula of the implication

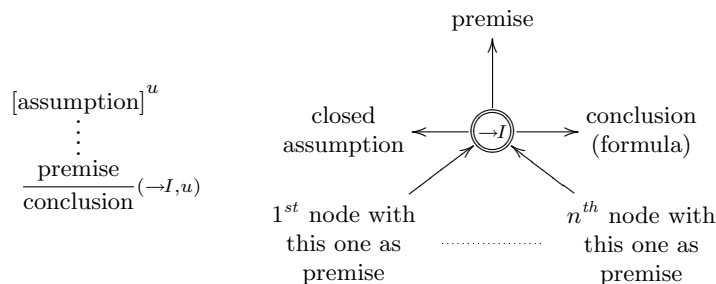


Figure 2.3: The edges of an implication introduction node

introduction, the closed assumption-port connects to the assumption node that is discarded, and so on. What ports a node has depends on the label of that node: for instance, a conjunction node always needs two subformulas and a conjunction introduction node has two premises and a conclusion.

When illustrating the graphs, the choice was made to omit the names of the ports from the pictures. Instead, ports are indicated by the spatial layout of the pictures. As the layout of the rule trees and formula trees of Figure 2.1 and Figure 2.2 is preserved as much as possible, the general direction in which an edge leaves a node is a clear hint as to which port it is meant to use. Which ports are located where is given by illustrations like Figure 2.3 and the left picture of Figure 2.4, the latter of which shows the ports of an implication node. These images can then be used as a reference to identify the function of the edges in the middle and right pictures. When the number of edges is large, ports and edges can be identified by their clockwise order on the node's rim. This can be practised by looking up the entry for disjunction elimination in Appendix B.

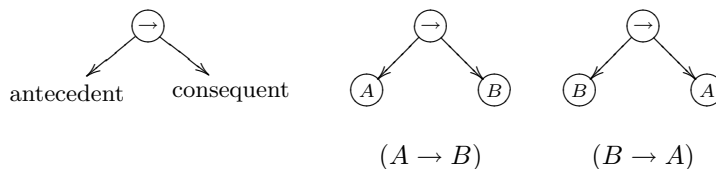


Figure 2.4: Identification of ports

To sum up, we now have two graph frameworks—one which represents the rule applications used in a natural deduction proof, and one which will eventually represent first order formulas, but is limited to propositional logic at the moment. We know how to identify nodes and edges and what features of natural deduction proofs they are meant to represent. Putting it all together, Figure 2.5 shows the completed proof graph of Figure 2.1.

Note that sharing in proof graphs is *implicit*: nodes are simply the target of multiple edges, and there are no special ‘funnel’ nodes to unite multiple edges

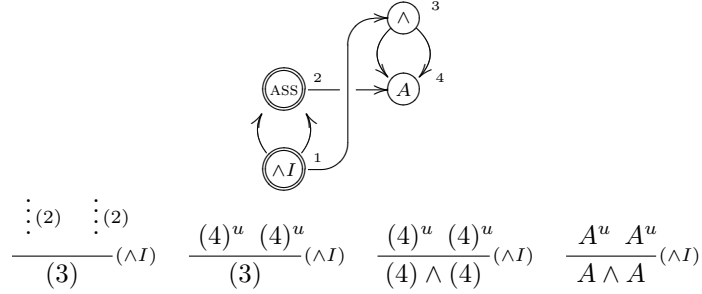


Figure 2.6: Interpreting a proof graph

To create the second deduction, we read from node 2 that the two subdeductions are assumptions whose formula occurrence is represented by node 4. As an assumption marker we have taken u for the moment, because we have yet to add those to the graph framework.

The third and fourth deductions add the interpretations of nodes 3 and 4. Eventually, we see four occurrences of A as a (sub)formula, while the node representing A , node number 4, only has three edges towards it. The edge from node 2 to node 4 is shared, because node 2 is: there are two paths from 1 to 2, and hence also two paths from 1 to 4 through 2. Each occurrence of a subproof or subformula corresponds to a path from the root node to the node that represents it. This important observation will be a great help in formulating definitions later on.

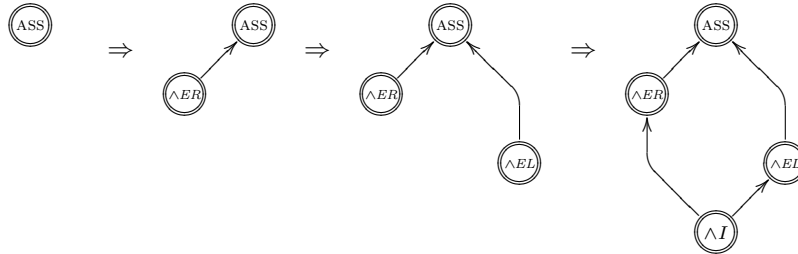


Figure 2.7: Inductive construction (formula nodes omitted)

At the moment nothing prohibits a graph from having more than one root node. If we want to keep open the option of an inductive definition, we have reason not to change this. Figure 2.7 shows what the inductive construction of Figure 2.5 would look like using multiple conclusions. If we would restrict the number of root nodes to one, the third figure could obviously not be constructed. Instead, we would have to create two different graphs, each with a different assumption node; yet to construct the fourth figure from those two graphs, both assumptions

would have to be merged into one again.

With two root nodes, two natural deduction proofs can be read from the third graph of Figure 2.7. This should not be a problem, and in principle there is nothing against a graph containing two proofs that are entirely separate, i.e. a graph with two root nodes that don't share any node.

2.3 Bisimulation

In Figure 2.5 all formulas are shared; there are no two formula subgraphs that represent the same formula. There are two conjunction nodes, but one represents the formula $A \wedge B$ and the other $B \wedge A$ —note that although the graph proves them to be equivalent, they are not the same formula. There will be cases, however, in which the same formula is represented twice in a graph. The problem is how to find out if two different subgraphs represent the same formula or deduction.

The issue that makes the matching of graphs non-trivial is that subgraphs don't have to be shared; for instance, the formula $A \wedge A$ can both be represented by a conjunction node sharing just one A -node, or by a conjunction node with two separate A -nodes.

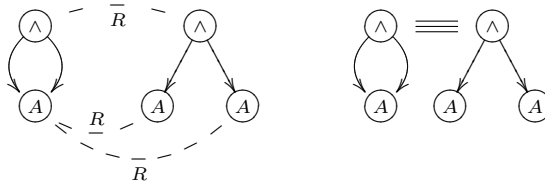


Figure 2.8: A bisimulation R as a witness of bisimilarity

The relation depicted by the dashed lines in Figure 2.8 is a *bisimulation* relation. Two nodes n and m are bisimilar if a bisimulation R can be found that holds between them. A relation R is a bisimulation if it conforms to two restrictions: firstly, it may only hold between two nodes a and b if both have the same label and, consequently, the same ports. Secondly, when each port on node a is paired with the corresponding port on node b , R must also hold between the two targets of each pair of ports.

The bisimulation relation can be characterized in another way: two graphs are bisimilar if they have the same *unfolding*. An unfolding is an adaptation of a graph that contains the same information, but has no shared nodes. It is obtained by duplicating every node that is the shared target of multiple edges, such that each edge gets its own target node. When a graph contains cycles, this process never stops, theoretically leading to infinite unfoldings. Since we are dealing with *acyclic* graphs, the unfolding of our graphs is a tree. In each of

the two images of Figure 2.8, the right graph is the unfolding of the left graph. The unfolding of a formula graph will be a formula tree, and the unfolding of a rule graph stripped of its formulas will be a rule tree.

Summing up, a formula graph unfolds to a formula tree, which represents a formula. A formula graph represents a single formula if it has a single root node. Consequently, two formula subgraphs represent the same formula if there is a bisimulation relation between the two root nodes.

2.4 Backpointers

Next, we look at bisimulations between entire proof graphs. Figure 2.9 shows the two α -equivalent natural deduction proofs of Figure 1.6. The graph translation, shown on the right, is the same, since the assumption markers are replaced by edges.

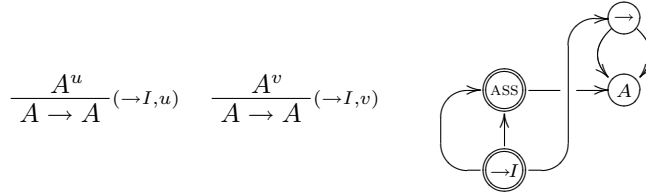


Figure 2.9: Two α -equivalent proofs and their graph translation

Figure 2.10 shows something that, however, does not work correctly in the current configuration. The proof graphs are bisimilar, but the proofs of which they are translations, shown below the graphs, are not the same: the one on the left has an open assumption where the one on the right does not. Upon closer inspection the left proof graph also reveals an open assumption, the lower one. Although the upper assumption is not used as a premise in the graph, this is a correct construction and is in fact the way in which we will deal with ‘empty discharges’, assumption discharges in which zero instances of an assumption are closed, as with the left natural deduction proof in Figure 2.10.

To see that the graphs of Figure 2.10 are indeed bisimilar, first observe that all nodes labeled A have no outgoing edges. Therefore, a bisimulation R may be constructed that holds between all these nodes. Next, all three assumption nodes have only one outgoing edge, towards the A -nodes. Since R holds for those nodes, R may also hold for the assumption nodes. It now easily follows that R holds for the root nodes of both graphs, the implication introduction nodes.

The adopted solution to this problem is the use of *backpointers*, illustrated in Figure 2.11 by the dashed arrow. These additional connections link assumption nodes with assumption discarding nodes, only in the reverse direction—hence

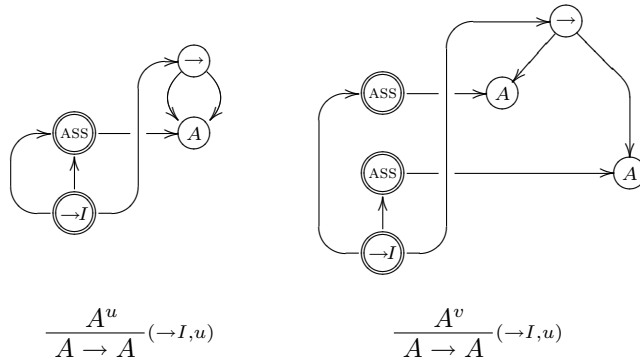


Figure 2.10: Incorrectly bisimilar proof graphs

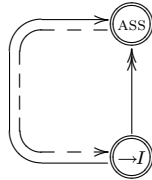


Figure 2.11: A closed assumption with its backpointer

the name. To accommodate our notion of bisimulation for the use of backpointers, we add the following restriction to the description of bisimulations: R may only hold between two nodes if either both have no backpointers, or both have exactly one backpointer and R holds between the two targets of these backpointers. This assures that closed assumptions are different from open ones and that, when comparing two graphs, bisimilar assumptions can only be discarded by bisimilar discarding nodes. When backpointers are added, the graphs of Figure 2.10 are no longer bisimilar: the open assumption within the left graph only has an outgoing edge, while the closed assumption within the right graph has both an edge and a backpointer.

The use of backpointers comes at a cost, however. Without backpointers our graphs were acyclic. When we allow cycles by introducing backpointers, we need additional restrictions and checks to sort between ‘good’ and ‘bad’ cycles, to prevent a rule application from being its own premise. Because of this issue we have given backpointers a separate status: they are only used when checking for bisimilarity and are to be ignored with other operations on these graphs.



Figure 2.12: Retaining the indices of open assumptions

Also, while closed assumptions are now only bisimilar if they are closed by bisimilar rule nodes, open assumptions with the same formula but different labels are still bisimilar. For open assumptions, therefore, we still need assumption markers. For graphs, we will use *indices*, enclosed in a square and attached to the assumption by a dashed arrow, as illustrated by Figure 2.12.

2.5 The closing of assumptions

A difference between deductions and graphs that was pointed at earlier, is that for deductions, which assumption is closed by which rule application is defined inductively. Moving down from an assumption, it is closed by the first application of $\rightarrow I$, $\vee E$, or $\exists E$ with the same marker. This prevents two situations from occurring: one, an assumption that is closed by more than one rule application; and two, a rule application that closes an assumption outside of its direct subproof.

These situations have to be ruled out explicitly in our graph framework. Before we state the restrictions that achieve this, we will demonstrate for both these constructions that allowing them yields unwanted results, and that we are not just throwing away opportunities for sharing. Figure 2.13 shows two graphs, each with an assumption that is closed by two implication introduction nodes (backpointers are omitted).

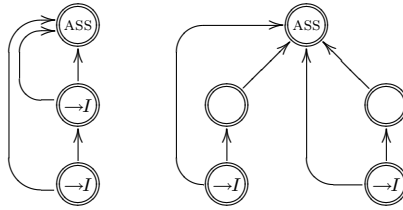


Figure 2.13: Double assumption discharges

To see why the graphs in Figure 2.13 spell trouble, we glance forward at what rewriting for graphs will involve. For natural deduction and graphs alike, when \rightarrow -contraction is applied to a proof, the assumptions that were closed by these rules are substituted by a subproof. If two implication introductions discharged the same assumption, two different contractions could be made. For the right graph both contractions are directly visible, for the left graph, if it is extended by two consecutive $\rightarrow E$ -nodes, one contraction could expose the other. Figure 2.14 illustrates this.

After the first contraction, the assumption node is replaced by a node with another label, in this case $\vee IL$. The second contraction involves a $\wedge ER$ -node that is closed by an implication introduction, and has to be replaced by a $\vee IL$ -node. In the current framework this isn't a legal construction; some radical

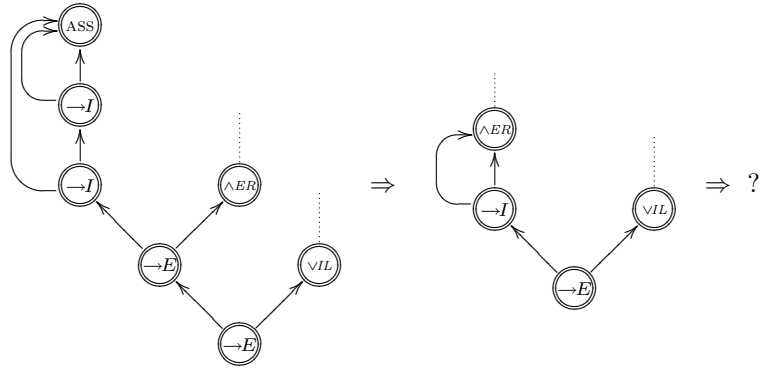


Figure 2.14: One assumption cannot be substituted twice

changes of the definitions for proof graphs might be suggested, but it is clear that without those, the situation is not going to be resolved.

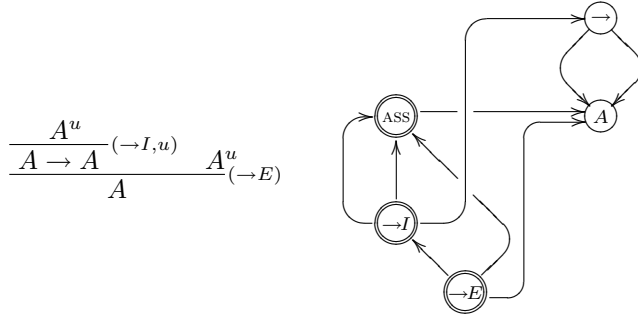


Figure 2.15: Another interesting contraction

The other situation to be addressed is the one in which a rule application closes an assumption outside its subformula, as portrayed in Figure 2.15. Before it can be clear what is going on here, this unusual construction requires an interpretation of what it is for an assumption to be closed. There are two options; the first is, that the path to an assumption decides whether it is considered open or closed, the way the inductive definition of proofs decides whether an assumption is closed or not. In this case, the graph represents the proof on the left, where the assumption A^u on the right is considered open. The problem is then, that since the assumption is considered open, it will have to be closed again to make the deduction into a proof. This results in the situation discussed before, in which two rule nodes close the same assumption.

The second interpretation is that the assumption is closed, since there is a rule node that closes it. The graph then represents the proof on the left, but this time the assumption A^u on the right is considered closed by the implication introduction labeled u . The proof represented by the graph now derives an ar-

bitrary formula A from two closed assumptions, rendering our entire framework useless.

The example uses two different notions of when an assumption is closed. One is a local notion, related to *dependence* on an assumption in natural deduction. It states that a rule node is dependent on a certain assumption node if there is a path from the rule node to the assumption node that does not cross the node that closes the assumption.

The other notion is a global notion, which simply reads that an assumption is closed if there is a node that closes it—equivalent to the notion in natural deduction that an assumption is closed if there is some rule that closes it. The difference between graphs and proofs is that in proofs these notions automatically coincide: an assumption is open if the conclusion of the proof depends on it, and closed if it does not. As we have just seen, for graphs this is not necessarily the case; a situation may be constructed in which a root node depends on a closed assumption.

The restriction that we explicitly impose on graphs, therefore, will be that every path from a root node to an assumption must cross the node that closes the assumption. With the help of an earlier observation, that a path from a to b in a graph corresponds to b being a subproof of a , we can attach a clear intuition to this restriction: it states that the assumptions closed by a rule node must all occur in its direct subproof.

In the next chapter this restriction will be further specified, among other things because the $\forall E$ -node closes two assumptions, which leads to additional difficulties.

2.6 The first-order fragment

The time has come to upgrade our graphs from propositional logic to predicate logic. Figure 2.16 shows how bound variables appear in formula graphs. Predicates (P and Q in the example) are treated as connectives for now, but later on we will take a different approach for the formal definitions.

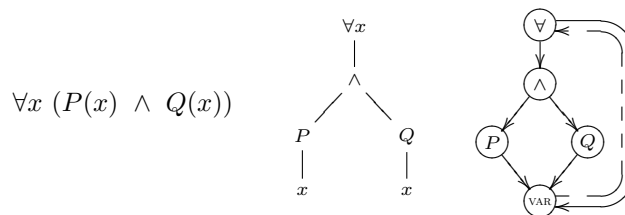


Figure 2.16: Bound variables in formula graphs

As illustrated, variables are to be shared as much as possible. To avoid α -equivalence issues the variable letter of a bound variable, such as the x in the above illustration, is replaced by a backpointer. For free variables the same indices as for assumptions will be used, as Figure 2.17 shows.

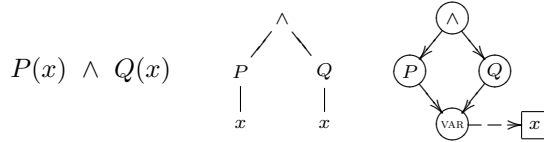


Figure 2.17: Free variables in formula graphs

Variables and assumptions have more in common than just α -equivalence. Like assumptions may only be discharged once, variables may not be bound by more than one quantifier. Also, quantifiers may not bind variables outside their direct subformula, their scope. In a first-order formula these issues are automatically resolved by the way quantifiers bind their variables. By definition, a quantifier binds all variables with a certain variable letter within its subformula, except the ones that are already bound by another quantifier. In graphs these restrictions have to be applied by hand. Yet by making them explicit, we find that they are exactly the same for variable binding as they were for assumption discharge. Figure 2.18 shows that bound variables can be mistreated in the same way that closed assumptions can.

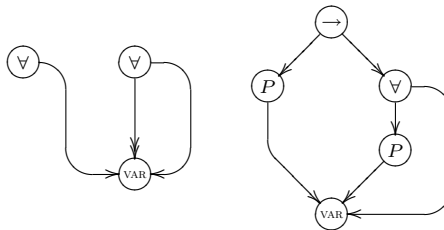


Figure 2.18: How formula graphs may go awry

In the left graph of Figure 2.18 a variable is bound by two quantifiers, analogous to Figure 2.13. The right graph mimicks Figure 2.15: the graph represents the formula $Px \rightarrow \forall xPx$, where the first x is either bound by the universal quantifier, or is free, in which case the variable node has to be bound twice. As a result of the analogy the restriction for bound variables will sound quite familiar: from every root node, all paths to a bound variable must cross the quantifier node that binds it.

To see that the root nodes are indeed the correct places to start looking for paths to variables, remember that there are no formula root nodes—or at least, if there are any, they are not part of a proof within the graph. The formulas represented in the proof are always the conclusion of some rule node. The root

nodes of formulas are thus, essentially, the rule nodes. But since there can be no formula nodes in between rule nodes, we may just as well use the root nodes of a graph, instead of every rule node. That the restriction for assumption closure refers to paths starting at root nodes as well, makes it sensible to do so.

2.7 Substitution

Turning to the natural deduction rules for first-order logic, one of the trickier habits of variables is their tendency to get substituted—for instance, by quantifier introduction and elimination rules. Where the other rules of natural deduction operate conveniently on the primary connective of a formula, which is represented by the root node of a formula subgraph, the quantifier rules affect the variables as well, which are at the leaves of the graph. Since we are sharing subformulas and since variables are different before and after substitution, we cannot share one formula graph as the 'before and 'after of a substitution. Instead, two subgraphs are required that are only different in the variable that is substituted for. Although generating a duplicate graph with substituted variables is easy, we also need to be able to verify the correctness of a substitution in an existing graph. The relation that can identify two formula graphs as the premise and conclusion of a substitution, P and $P[x/a]$, is the *bisimulation modulo identification*, illustrated by the curly triple bars in Figure 2.19.

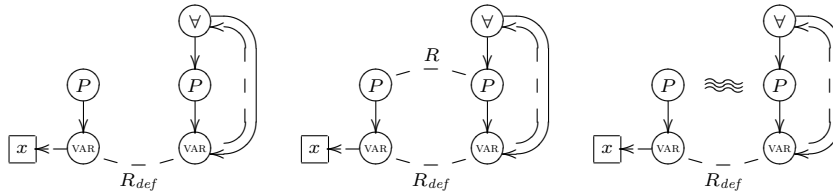


Figure 2.19: The construction of a bisimulation modulo identification-relation

The *bisimulation modulo identification of x and y* is a bisimulation for which the auxiliary relation R is assumed to hold between x and y , without regarding the restrictions on R , as illustrated by the left picture of Figure 2.19. The relation is equivalent to a regular bisimulation when the variable substituted for or the substitute is absent from both graphs. When this relation is applied to verify the correctness of substitution, the presence or absence of variables and substitute formulas has to be checked independently.

As Figure 2.20 illustrates, the graphs for the formulas $R(x, y)$ and $R(y, x)$ are bisimilar modulo identification of x and y ; yet the result of substituting x with y or vice versa will not be that $R(x, y)$ becomes $R(y, x)$. To recognize a substitution of y for x in a graph, we need the additional requirement that x does not occur in the result: x does not occur (free) in the formula $A[y/x]$. If two graphs, simply indicated with 'before' and 'after', are bisimilar modulo identification of

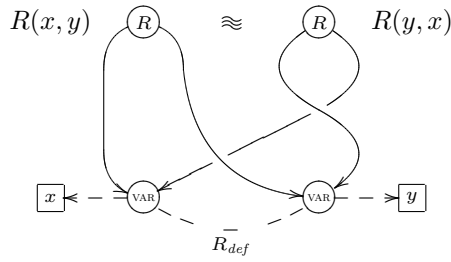


Figure 2.20: The limits of bisimulation modulo identification

x and y , and x may not occur in the after graph, then since nodes representing x are only bisimilar to nodes representing x or y , where x occurs in the before graph there has to be a y in the after graph.

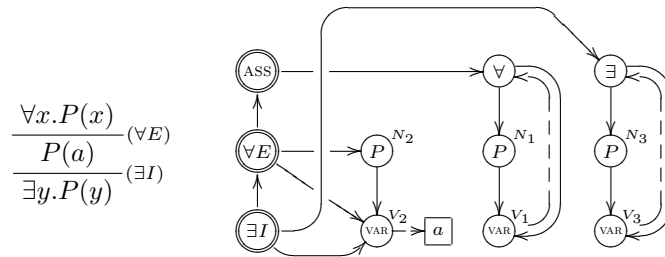


Figure 2.21: A proof with quantifier introduction and elimination rules

Figure 2.21 shows a small proof with quantifiers. Besides the premise-port and conclusion-port the $\exists I$ -node and $\forall E$ -node have a third port, whose edge leaves the node on the lower right side. This is the proper term-port, which connects to the term t in the inference schemes (see Appendix A). When checking the correctness of a proof, this information is used for constructing the bisimulation modulo identification relation.

For example, the $\exists I$ -application in the proof in Figure 2.21 has the conclusion $\exists y.P(y)$ and the premise $P(y)[a/y] = P(a)$; the proper term of the inference is the free variable a . The $P(y)$ in $\exists y.P(y)$ is represented by the subgraph starting at node N_3 , $P(a)$ is represented by N_2 . Variables y and a are represented by nodes V_3 and V_2 . If the inference is correct, N_2 and N_3 are bisimilar modulo identification of V_2 and V_3 , and there is no path from N_2 to V_3 . A generalization of this requirement is illustrated by Figure 2.22.

the variable letters denote the same variables. This implies that the same variable not only occurs in premises and conclusions, but also, for instance, in the conclusion and closed assumption of an $\rightarrow I$ -inference.

$$\frac{\begin{array}{c} Pa^u \\ \vdots \\ A[a/x] \\ \hline \forall x A \end{array} (\forall I)}{\begin{array}{c} \vdots \\ B \\ \hline Pa \rightarrow B \end{array} (\rightarrow I, u)}$$

Figure 2.24: An unrestricted proper variable

The schematic proof in Figure 2.24 shows what may happen without explicit restrictions: the $\rightarrow I$ -application introduces an occurrence of a , the proper variable, below the $\forall I$ -application; outside its direct subproof. Likewise, an occurrence of the variable letter a in the conclusion would allow occurrences outside the scope of the $\forall I$ -application—if that doesn't already count as 'outside' itself. In other words, the restrictions on the $\forall I$ -scheme limit the scope in which occurrences of the proper variable of a $\forall I$ -application may occur to the direct subproof of the application.

It will probably come as no surprise that investigations into the case of existential quantifier elimination lead to a similar conclusion. The restrictions on the inference scheme prohibit the proper variable from occurring outside the proof of the minor premise: in the scheme it may not occur in the major premise ($\exists x A$), the conclusion C or any open assumption on which the minor premise depends.

Returning to graphs, we will treat proper variables of $\forall I$ - and $\exists E$ -applications as bound. Figure 2.25 shows the graph scheme for universal quantifier introduction.

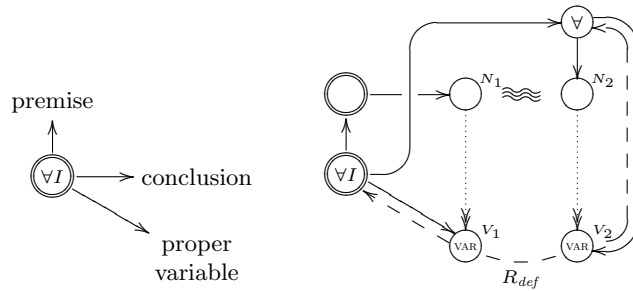


Figure 2.25: Universal quantifier introduction

The scheme in Figure 2.25 requires two nodes to be bisimilar modulo identification: node N_1 which represents the premise, $A[a/x]$ in the inference scheme, and

N_2 , which represents the direct subformula (A) of the conclusion ($\forall xA$). As for the variable nodes identified by the bisimulation, V_1 represents the proper variable and is bound by the $\forall I$ -node, and V_2 is bound by the universal quantifier node that is the primary connective of the conclusion.

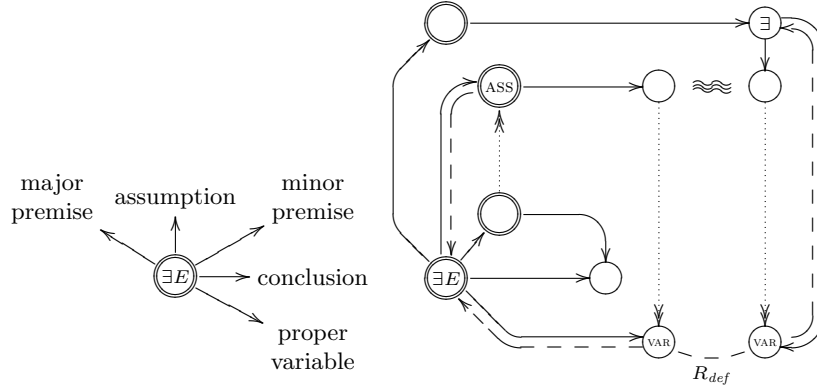


Figure 2.26: Existential quantifier elimination in graphs

The scheme for existential quantifier elimination, portrayed in Figure 2.26, includes all three types of binding: a variable bound by a quantifier, a variable bound by an inference, and a closed assumption.

The restrictions for proofs and formulas that bound variables and closed assumptions must be within the scope of the binder, were made explicit in graphs as the restriction that any path from a root node to a bound node must cross the binder node. For binding by $\forall I$ - and $\exists E$ -nodes, as well as assumption closure by $\forall E$ - and $\exists E$ -nodes, we need to be a little more specific.

As we saw, the proper variable of \forall -introduction in proofs should only occur in the direct subproof, and not in the conclusion. Demanding that a path crosses the $\forall I$ -node does, however, allow the path to use the conclusion-port. The restriction should therefore be that paths from a root node to a bound variable should not only cross the binding $\forall I$ -node, but use the premise-port as well—or the proper variable-port, of course.

To correctly state the bound variable restriction for existential quantifier elimination we have to identify differences between ports as well. The proper variable may only occur in the subproof of the minor premise, and even then not in the conclusion of that subproof. Since the $\exists E$ -node and its minor premise share the same conclusion, we only have to restrict the proper variable from being reached through the conclusion-port. Furthermore, it may not be reachable through the major premise-port, but it may be reached through the closed assumption-port, the premise-port and the proper variable port.

In this setup a minor difference remains between proof graphs and natural de-

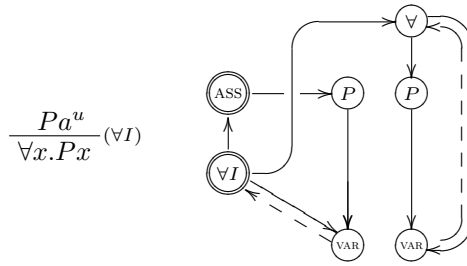


Figure 2.27: An assumption that cannot be discharged

duction. The incomplete proof in Figure 2.27 cannot be constructed within natural deduction since the proper variable of the $\forall I$ -application is open in an assumption. The graph, on the contrary, does comply to all our restrictions, but its open assumption cannot be discharged anymore: the bound variable would be reachable without crossing the $\forall I$ -node. Any added discharging node could never be reachable from the $\forall I$ -node, since all ports within the subproof are already used. The inductive construction procedure would be at a dead end here. However, this is ultimately not an issue for completed proofs, in which all assumptions have to be closed; therefore we will not explicitly prohibit graphs like the one in Figure 2.27.

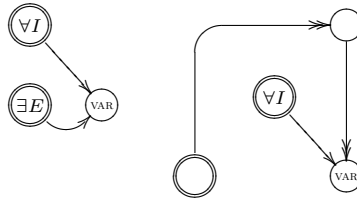


Figure 2.28: Prohibited constructions

Figure 2.28 shows the graph configurations that have been ruled out. The configurations are reminiscent of the prohibited constructions for bound variables and closed assumptions. The left graph illustrates the double binding of a variable, which would in this case correspond to deducing a universal conclusion from an existential premise—two $\exists E$ -nodes or two $\forall I$ -nodes would lead to similar problems. The right graph shows a proper variable of a $\forall I$ -node that is reachable from outside the subproof of the $\forall I$ -node, which would allow derivations like $Pa \rightarrow (\forall xPx)$.

2.9 Dead terms

The central question of the current section is expressed by Figure 2.29. Previously we have seen that if we treat applications of $\forall I$ and $\exists E$ as though they

bind their proper variables, we can eliminate a form of α -equivalence. Let us now take a closer look at the proper term of \forall -elimination and \exists -introduction.

$$\frac{\frac{\forall x.Px}{Pa}^{(\forall E)}}{\exists y.Py}^{(\exists I)} \quad \frac{\frac{\forall x.Px}{Pb}^{(\forall E)}}{\exists y.Py}^{(\exists I)}$$

Figure 2.29: Are these deductions α -equivalent?

The first impression may be that this case is remarkable similar to the situation for universal quantifier introduction. Nonetheless, there are good reasons why the proper terms of $\exists I$ -applications cannot be seen as bound, the first being that they're terms; terms, in general, cannot be bound.

The second reason not to treat proper terms as bound, is that a term or variable may be the proper term of many applications of \forall -elimination or \exists -introduction, because it may still occur in the conclusion of an application, or in its open assumptions. These two typical restraints on the proper variables of $\forall I$ and $\exists E$ obviously don't apply to \forall -elimination, but neither to \exists -introduction, as shown in Figure 2.30.

$$\frac{\frac{Pt \wedge Qt}{\exists y.(Pt \wedge Qy)}^{(\exists I)}}{\exists x \exists y.(Px \wedge Qy)}^{(\exists I)} \quad \frac{\frac{Pa}{\exists x.Px}^{(\exists I)}}{Pa \rightarrow \exists x.Px}^{(\rightarrow I, u)}$$

Figure 2.30: $\exists I$ doesn't have the constraints of $\forall I$

The case portrayed in Figure 2.29, where the proper term does not occur in the conclusion or an open assumption of the $\exists I$ -application, seems to be an exception. In such a particular case, then, it might still be possible to treat the term as bound, since the conditions as they are for \forall -introduction are met. The third reason not to treat proper terms as bound, is that applications of some other rules can meet these same conditions, and should then also be considered as binders. Figure 2.31 shows the case of \wedge -elimination; other inferences that may show this behaviour are $\rightarrow E$, $\exists E$ and $\forall E$, though the latter two only when at least one closed assumption class is empty.

$$\frac{\frac{\forall x.(Px \wedge A)}{Pa \wedge A}^{(\forall E)}}{A}^{(\wedge ER)} \quad \frac{\frac{\forall x.(Px \wedge A)}{Pb \wedge A}^{(\forall E)}}{A}^{(\wedge ER)}$$

Figure 2.31: Another example of a dead term

For these reasons these terms, which we will call *dead* terms, should not be considered bound. Unless the term consists only of a variable, a dead term is a term that does not occur in the conclusion and assumptions of a deduction. If it does consists only of a variable, a dead term is a term that does not occur

in a conclusion or assumption and also isn't the proper variable of some $\forall I$ or $\exists E$ -application.

On the one hand, dead variables are indispensable to the deductions in which they occur. For the typical situations where they occur, illustrated in this section, it is probably impossible to find an equivalent deduction without one. On the other hand, which variable letter or term is used is even less relevant than with bound or proper variables: there is no reference to a binder.

$$\begin{array}{c}
\frac{\frac{\forall x \forall y. Rxy^u}{(\forall x \forall y. Rxy) \wedge (Pa \rightarrow Pa)} (\wedge I) \quad \frac{Pa^v}{Pa \rightarrow Pa} (\rightarrow I, v)}{(\forall x \forall y. Rxy) \wedge (Pa \rightarrow Pa)} (\wedge EL)}{\frac{\forall x \forall y. Rxy}{\forall y. Ray} (\forall E)} (\forall E)}{\frac{Rab}{\forall x. Rxb} (\forall I)} (\forall I)}{\forall y \forall x. Rxy} (\forall I)
\end{array}
\qquad
\begin{array}{c}
\frac{\frac{\forall x \forall y. Rxy^u}{(\forall x \forall y. Rxy) \wedge (Pb \rightarrow Pb)} (\wedge I) \quad \frac{Pb^v}{Pb \rightarrow Pb} (\rightarrow I, v)}{(\forall x \forall y. Rxy) \wedge (Pb \rightarrow Pb)} (\wedge EL)}{\frac{\forall x \forall y. Rxy}{\forall y. Ray} (\forall E)} (\forall E)}{\frac{Rab}{\forall x. Rxb} (\forall I)} (\forall I)}{\forall y \forall x. Rxy} (\forall I)
\end{array}$$

Figure 2.32: Dead terms and proper variables

An example of a more serious construction with dead terms is given in Figure 2.32. The tails of both proofs are mainly there to allow two different proper variables; the difference is in the redundant branch on the top right. Since proper variables are considered bound, it will be very hard to rename only the variables in the top right branch, and not in the rest of the proof.

Given these considerations, it is debatable whether it is really α -equivalence we are dealing with. If, nonetheless, we want each of the pairs of deductions in Figure 2.29, Figure 2.31 and Figure 2.32 to be represented by one and the same graph, another option than binding is simply erasing the name of the variable or term. In fact, this can be done in the deductions themselves as well. In the next few examples, dead terms consisting of variables will be replaced by a special symbol, ' \emptyset '.

$$\frac{\frac{\forall x. Px}{P\emptyset} (\forall E)}{\exists y. Py} (\exists I) \qquad
\frac{\frac{\forall x. (Px \wedge A)}{P\emptyset \wedge A} (\forall E)}{A} (\wedge ER) \qquad
\frac{\frac{\forall x. (Px \rightarrow A)}{P\emptyset \rightarrow A} (\forall E)}{A} (\rightarrow E) \qquad
\frac{\forall x. Px}{P\emptyset} (\forall E)$$

Figure 2.33: Dead terms without names

Figure 2.33 shows the example deductions of Figure 2.29 and Figure 2.31, as well as an example deduction in which a term is removed by an implication elimination, using the special 'dead term' symbol. The intended consequence is that every dead term is represented by one and the same term. To see why that does not cause problems, consider the following argument: firstly, if a proof

contains a dead term that consists of only a variable, that variable might just as well have been a term, since it may not occur as a proper variable of $\forall I$ or $\exists E$.

Secondly, there are only two rules that may operate on terms, $\exists I$ and $\forall E$. The other two quantifier rules affect only their proper variables, and all the other rules add or remove connectives. Obviously, no application of $\forall E$ can be made invalid by changing the proper term, if the new term does not contain any variables. As for $\exists I$, the worst that can happen for an $\exists I$ -application when changing the proper term, is that it becomes identical to some other term in the premise and conclusion. An example of this situation is given by Figure 2.34.

$$\frac{\frac{R(s, t)}{\exists y.R(s, y)} (\exists I)}{\exists x \exists y.R(x, y)} (\exists I) \qquad \frac{\frac{R(\emptyset, \emptyset)}{\exists y.R(\emptyset, y)} (\exists I)}{\exists x \exists y.R(x, y)} (\exists I)$$

Figure 2.34: Existential quantifier introduction with dead terms

In Figure 2.34 the two terms s and t in the left proof are replaced with one and the same term on the right. The intuition may be that the upper inference in the proof on the right is invalid, but that is not the case. The direction of substitution in the scheme for \exists -introduction was chosen such that not all occurrences of the proper term in the premise have to be replaced by the new existentially quantified variable in the conclusion. As a result, changing the proper term cannot render an $\exists I$ -application invalid.

Next, we apply these findings to natural deduction. To find out whether a term in a deduction is dead, checking whether it does not occur in the conclusion or open assumptions will suffice, unless it only consists of a variable. Then it should also be verified that it is not the proper variable of an application of \forall -introduction or \exists -elimination. And to find the proper variables, they have to be made into pure variables, as was demonstrated on page 17.

For graphs, dead terms face the same problems. Although bound variables are indicated with backpointers in graphs, and are thus relieved of issues with variable letters, they can still occur in terms that would otherwise be dead. Figure 2.35 shows such a construction.

The problem presented by Figure 2.35 is a serious one. If a dead term can use any proper variable, then if we want to build a graph system that deals with dead terms, we will need the algorithm for finding pure variables to define proper variables. The other possible definition for proper variables, all free variables in the subdeduction, suddenly looks a lot more appealing, particularly because it corresponds to the definition of regular variable binding and assumption discharge.

The decision to abandon the option of pure proper variables is made definitive by

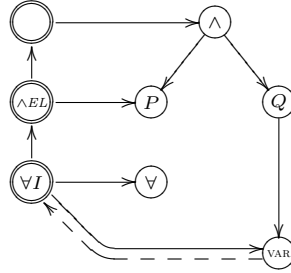


Figure 2.35: A proper variable occurring as a dead term

the deduction in Figure 2.36. An adaptation of Figure 2.32, the only difference is that the redundant branch of the proof has been moved down by two inferences. Although it is hard to say why this should fundamentally matter, this time the algorithm for finding pure variables will not identify the variables in the branch as dead terms.

$$\frac{\frac{\frac{\forall x \forall y. Rxy^u}{\forall y. Ray} (\forall E)}{Rab} (\forall E)}{(Rab) \wedge (Pa \rightarrow Pa)} (\wedge I)}{\frac{Rab}{\forall x. Rxb} (\forall I)} (\wedge EL)}{\frac{\forall x. Rxb}{\forall y \forall x. Rxy} (\forall I)} (\rightarrow I, v)$$

Figure 2.36: Dead terms or no dead terms

2.10 Rewriting

Rewriting will be addressed in a straightforward way: the rewrite schemes for graphs will be the graph translations of the rewrite schemes for natural deduction. The current section will explore which adaptations to this general approach are needed to deal with the particularities of graphs—that is, of course, sharing.

Figure 2.37 shows the contraction schemes for conjunction introduction and left conjunction elimination. In the graph version all formula nodes are omitted, because they play no role in the contraction. For comparison with the natural deduction schemes the subgraphs of the uppermost nodes are sketched with dotted arrows.

While for proofs the property of being a premise of some inference is indicated

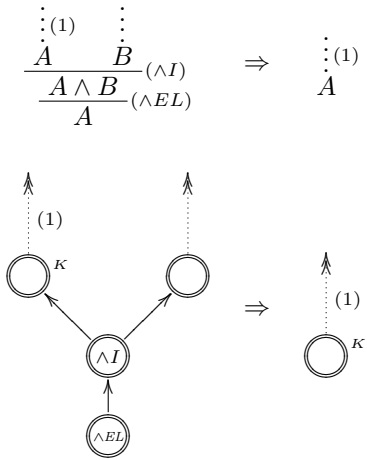


Figure 2.37: Conjunction contraction directly translated for graphs

by being on top of that inference, graphs use edges. When the $\wedge EL$ -node is removed, the edges from its predecessors are left without a target. Obviously, these have to be redirected to the node K . To indicate that all predecessors of one node should become predecessors of another, a dotted edge is used, as in Figure 2.38.

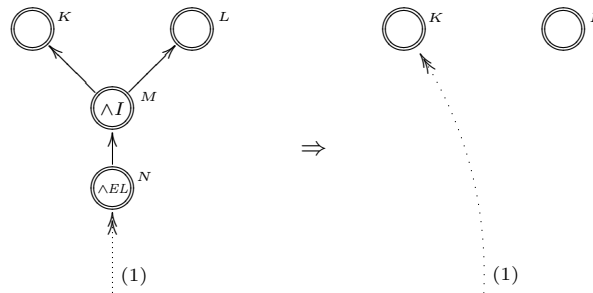


Figure 2.38: Conjunction contraction with predecessors

One other thing has changed in Figure 2.38 as well: the node L has not been removed. The reason behind this is that the $\wedge I$ -node might have shared it with other nodes. Removing it in those circumstances would leave ‘dangling edges’, edges without target.

Actually the same holds for the $\wedge I$ -node itself: if it is shared between the $\wedge EL$ -node and some other nodes, it may not be removed. That it does not reappear in the right side of the contraction is because there are other rewrite steps for which that doesn’t work, as we will see shortly. The approach that does work for all rewrite steps is to copy the introduction node of the contraction first; that way, the elimination node of the contraction and the other nodes that share it

have separate copies.

On the other hand, the possibility exists that all these nodes weren't shared in the first place. What happens then is that redundant nodes without predecessors are left all over the graph, like the node L in Figure 2.38. Each rewrite step will therefore be followed by an operation that gets rid of these nodes, to be defined later.

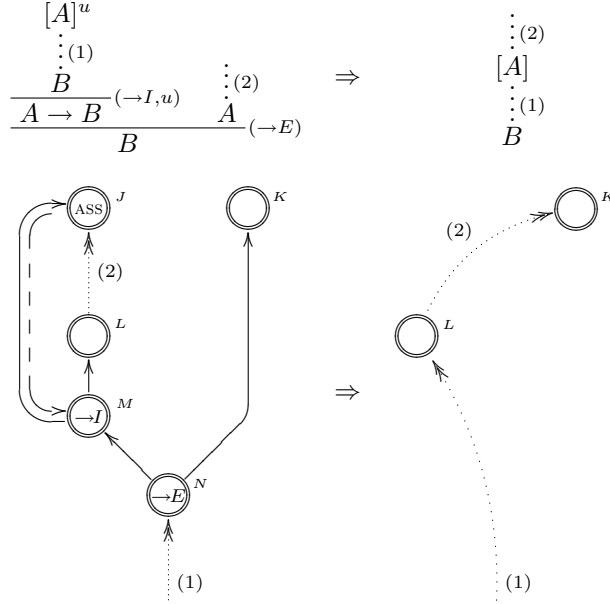


Figure 2.39: The graph translation of implication contraction

The contractions that cannot do without the duplication of the introduction node, are \rightarrow -contraction, \forall -contraction and the permutations.

The assumption of the $\rightarrow I$ -node, J , is replaced in the contraction by the node K . This means that the subgraph of the $\rightarrow I$ -node has changed. For any other node sharing the $\rightarrow I$ -node that change will probably have unwanted consequences, so simply leaving the $\rightarrow I$ -node in the graph is impossible.

So the solution would be to duplicate not only the $\rightarrow I$ -node, but also its entire subgraph, and use one copy for the contraction and one for the other nodes that share it. Figure 2.40 shows a contraction with a shared $\rightarrow I$ -node, and the required duplication step.

In Figure 2.40 the $\rightarrow I$ -node M is shared between nodes N and N' . Before M and N can be contracted, M must be duplicated, and its subgraph as well. More specifically, since closed assumptions may not occur outside the scope of their binders, instead of the entire subgraph of M only its *scope* must be duplicated. Consider for example Figure 2.41.

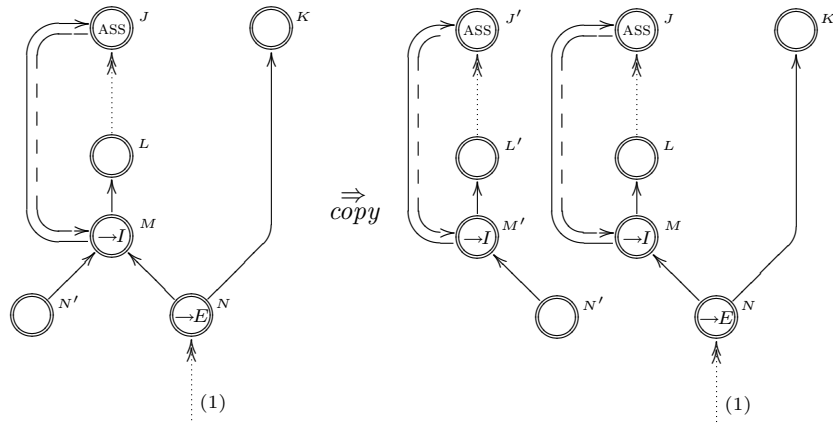


Figure 2.40: Copying the introduction node of the contraction

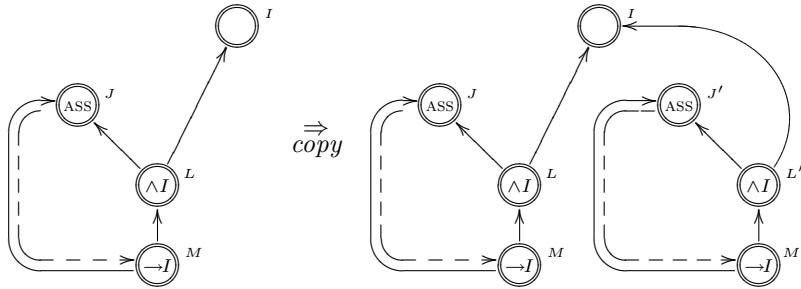


Figure 2.41: Only the scope of the $\rightarrow I$ -node has to be copied

In Figure 2.41 the scope of node M consists of nodes L and J , but not I . When M is duplicated, then so must J and L , but L and its duplicate L' may still share I .

Left out of the discussion so far, the next question is what happens to the conclusions of the rule nodes in a contraction. In proof graphs conclusions are shared as much as possible between an inference and its premises, and we need to verify whether the inference nodes in the result of a translation step still share their conclusions.

Figure 2.42 shows one of the two contraction schemes for conjunction, to which conclusions have been added. In the contraction, the elimination node N is removed, and all edges towards it are redirect to the right premise, L , of the introduction node M . From the perspective of the predecessors of N , however, not much has changed: the node L has the same node as conclusion as did N . At least for this scheme, conclusions are shared in the resulting graph as required.

$$\frac{\frac{\frac{\vdots^{(1)}}{A}}{\forall x.A[x/a]}^{(\forall I)}}{A[t/a]}^{(\forall E)} \Rightarrow \left. \frac{\vdots^{(1)}}{A} \right\} [t/a]$$

Figure 2.44: Universal quantifier contraction for proofs

the nodes A and X no longer occur: A has been replaced with T , and X may not occur outside the scope of M_1 .

In Figure 2.43 the node N has M_0 as a premise before the contraction, and K_0 afterwards. Although K_0 and M_0 don't have the same node as conclusion, their conclusion nodes K_1 and M_1 are bisimilar. In a sloppy interpretation, the bisimulation modulo identification relations read: if A , X and T were the same, then L_1 , the subformula of M_1 , and N_1 would be bisimilar. Since in the resulting graph A has been replaced by T , L_1 and N_1 should be bisimilar; formal proofs, however, will have to wait.

To complete the contraction we somehow need to merge the nodes L_1 and N_1 into one. In Section 4.2 an operation will be introduced that does precisely that.

2.11 Translating proofs to graphs

The translation from deductions to graphs will make use of an intermediate type of graph, which we call 'pre-proof graphs'. In these graphs we will see aspects of both proofs and proof graphs. For instance, bound variables and closed assumptions may be identified both by indices and by backpointers. We will introduce the translation procedure by an elaborate example, illustrated by Figures 2.45 through 2.52.

Pre-proof graphs need, among other things, the ability to represent graphs half-way through a translation. The approach we take is inspired by the idea that every rule node in a graph, by means of its connections, represents a deduction. Applying this idea to an inductive translation procedure, we introduce special 'unfinished' nodes and attach them to the proofs they are going to represent.

Pre-proof graphs, then, are proof graphs containing deductions that have yet to be translated, as an argument of their future root node. These nodes are called *empty* and are, like the other nodes, divided into rule nodes and formula nodes. Empty rule nodes are labeled ' ε_0 ' and linked to a deduction, empty formula nodes get the label ' ε_1 ' and a link to a formula. The graph in Figure 2.45 illustrates the idea.

A pre-proof graph is translated one step at a time. Each step translates one empty node, but in the example some steps will be taken simultaneously. The

possible steps are given in Appendix D.

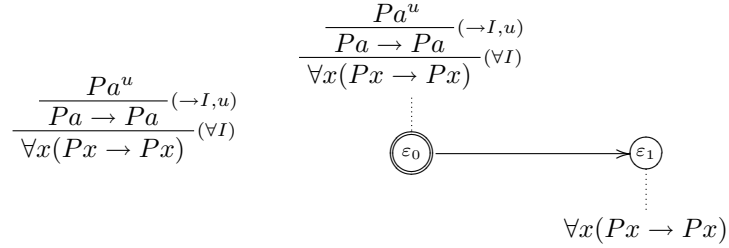


Figure 2.45: The starting proof and initial step

Figure 2.45 shows the proof that we are going to translate, on the left, and the result of the first translation step, on the right. The deduction as a whole has been attached to the left node, an empty rule node; the conclusion has been duplicated and attached to the right node, an empty formula node. The edge from left to right is attached to the conclusion-port of the empty rule node.

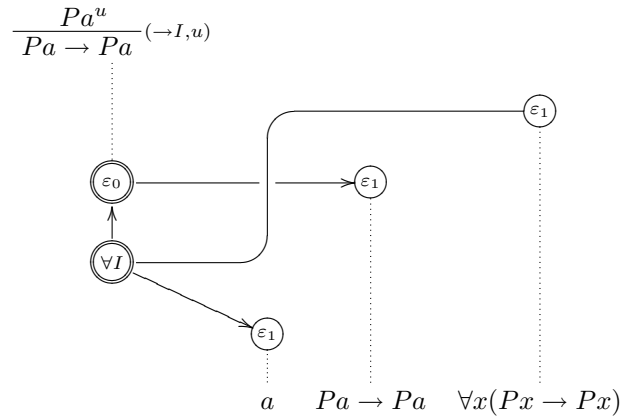


Figure 2.46: The empty rule node was translated

Figure 2.46 shows the result of the next step in the translation. The empty rule node of Figure 2.45 has been given the label $\forall I$, which was the last rule application in the deduction. The premise-port leads to a new empty rule node, that will represent the rest of the deduction. As in the first step, the empty rule node has been given a conclusion. Although it is not that important in this particular case, on other occasions it will allow rule nodes to share parts of the formula graph. The edge on the lower right of the $\forall I$ -node connects to the proper variable, in this case a .

The following step, resulting in Figure 2.47, has translated the middle ϵ_1 -node, representing $Pa \rightarrow Pa$, to an implication node. The other empty formula nodes

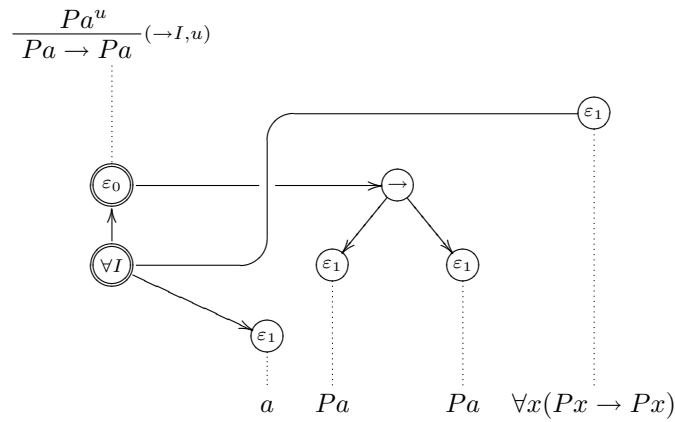


Figure 2.47: The conclusion of the empty rule node had to go first

could have been translated as well, but not the empty rule node. Because sharing of conclusions is enforced, translating an empty node sometimes requires that its conclusion nodes are already present.

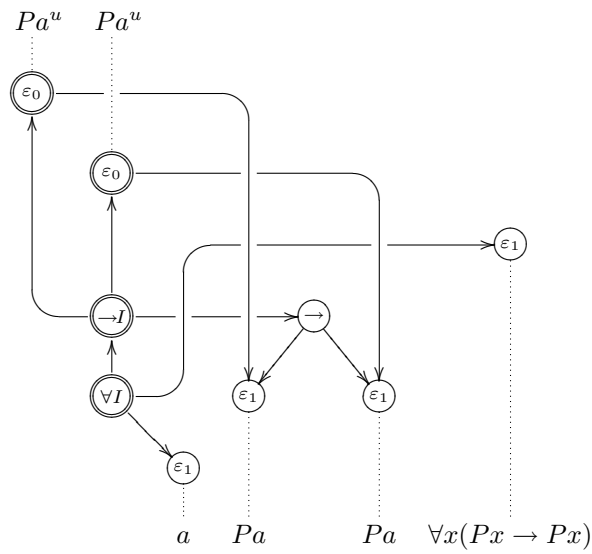


Figure 2.48: Now the empty rule node could be translated

In Figure 2.48 the empty rule node has become an $\rightarrow I$ -node. The upper left empty node is the assumption closed by the $\rightarrow I$ -node; the backpointer will be added later. The conclusions of the new empty rule nodes are the antecedent and consequent of the implication node translated in the previous step.

In Figure 2.49 all formulas have been translated as well, and there are no empty

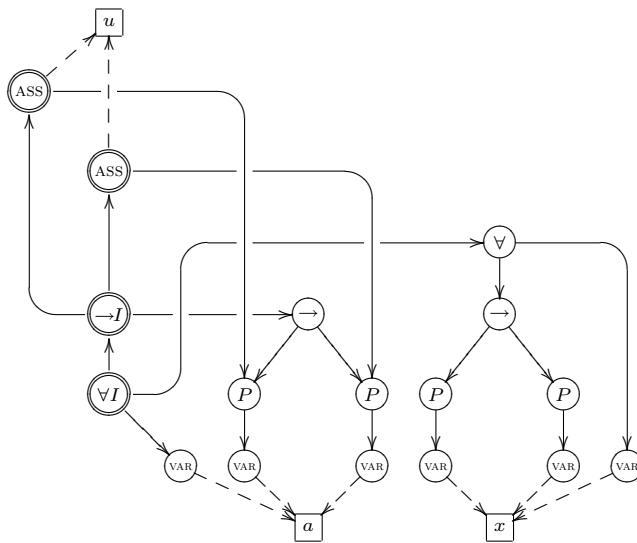


Figure 2.49: The remaining empty nodes have been translated

nodes left. Assumption markers and variable letters have been replaced by indices. The next steps in the translation will merge variables and assumptions with the same index and put backpointers in place.

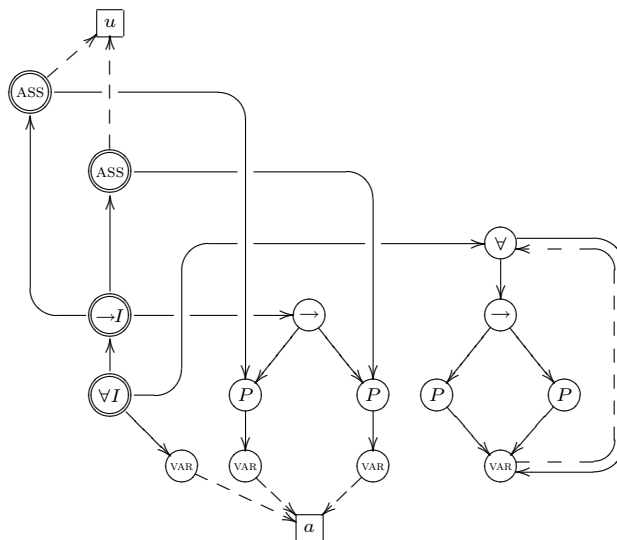


Figure 2.50: The bound variable has been merged

The three variable nodes with index x have been merged in Figure 2.50. The merging procedure is very straightforward: we merge every variable with the

first possible binder, found when traversing the graph against the direction of the edges. A possible binder is in the case of a variable the bound variable of a quantifier node, or the proper variable of a $\forall I$ -node or $\exists E$ -node, with the same index.

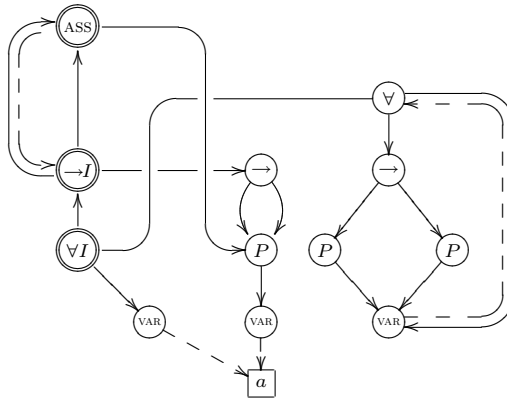


Figure 2.51: The assumptions have been merged

The merging of assumptions involves a little more effort. Variable nodes typically have no outgoing edges, only a backpointer, so merging is straightforward. Assumptions, on the other hand, have a conclusion that has to be merged as well; we will have to develop our procedure so it can work with entire formula graphs.

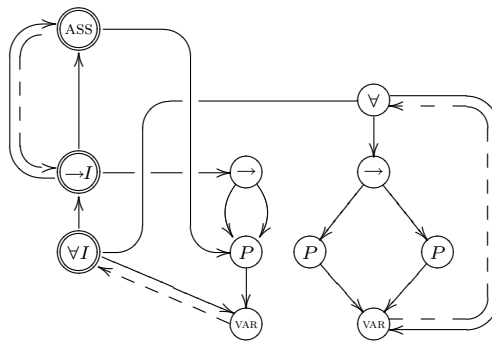


Figure 2.52: The proper variable has been merged

Figure 2.52 shows the completed proof graph. The last step was the merging of the two variables a . This brings a new issue with it; the bound variables x were only reachable by a single path from the quantifier, and likewise there was only one path from the $\rightarrow I$ -node to each assumption. The formula-part and rule-part of the graph were still trees when these merges took place, but the right one of the two variables with index a is shared by two propositions and two rule nodes.

If the variable is bound by the first binder found moving up the graph, we have to show that for each path, this is the same binder. In other words, we have to show that each occurrence of the variable a that we have previously merged, is an instance of the proper variable of the same binding rule node.

3 Formal Definitions

To describe proof graphs mathematically turns out to be quite a job. A lot of the problems faced in the previous chapter lead to exceptions in the formal treatment.

This chapter will describe the formal makeup of proof graphs and that of a larger class of graphs required for incomplete translations, and will define variable binding and assumption closure for those graphs.

3.1 Constraint definition

In this section a constraint-based definition for proof graphs will be given, as well as some basic notions and operations. A large part of the constraint definition—the correctness criteria for the representations of inferences in a graph—is presented in images which are, because of their size, placed in the appendices.

Still, the definition of proof graphs will be quite elaborate due to the multitude of components involved. We need nodes that represent inference rules, nodes that represent formulas, we need edges and special backpointers, and we need a fixed set of ports for each node, dependent on the label; but first we will introduce additional naming conventions:

- \mathcal{D} , \mathcal{E} and \mathcal{F} are deductions
- \mathcal{G} and \mathcal{H} are graphs
- capital letters such as M and N are nodes
- p and q are ports
- γ and δ are edges
- Γ and Δ are paths
- R is a bisimulation relation

The nodes need differentiation between rules and formulas. To achieve this we will use two different sets of labels, not different sets of nodes. Since the inference rules and formulas used in natural deduction are fixed, the sets of rule labels and formula labels are fixed for every graph as well. Although they only differ in label and not in the type of node, for ease of terminology we will refer to nodes with a formula label as formula nodes, and nodes with a rule label as rule nodes.

Definition 3.1 R-LABELS is the set of inference labels:

$$\{\wedge I, \wedge EL, \wedge ER, \vee IL, \vee IR, \vee E, \rightarrow I, \rightarrow E, \perp E, \forall I, \forall E, \exists I, \exists E, \text{ASS}\}$$

F-LABELS is the set of formula labels:

$$\{\wedge, \vee, \rightarrow, \perp, \forall, \exists, \text{VAR}\} \cup \{P_n | n \in \mathbb{N}\}$$

The union of the two sets of labels will be called LABELS. The labels P_i are the predicates, with i indicating the number of ports. Since we want the label of a node to define which and how many ports it has, we need a predicate node for each possible arity. Moreover, since there may be more predicates with the same arity, we will use indices to identify them.

A small note on why predicates are dealt with like this: simply using node labels P and Q for predicates P and Q would lead to problems when two graphs contain a predicate P of different arity. Another option would have been to predefine a set of predicate labels that contains all possible predicates with all possible arities, or to add a set of predicate labels as a parameter to each graph. The current setup represents a predicate Q of arity n by a predicate node P_n with n ports and index Q . Besides using smaller or less sets of entities, it has the added conceptual advantage that predicates are treated the way they would be in second-order logic: they can, theoretically, be bound.

Definition 3.2 PORTS is the set of ports listed on page 106.

For easier reference some ports have been assigned a *type*. Since there is no limit to the arity of the predicates we need to represent, the set contains an infinite amount of argument ports for predicate nodes.

Definition 3.3 *ports* is a function from LABELS to subsets of PORTS, i.e. it assigns a set of ports to each label. The contents of the *ports* function are displayed by the images in Appendix B; exactly how is illustrated by Figure 3.1.

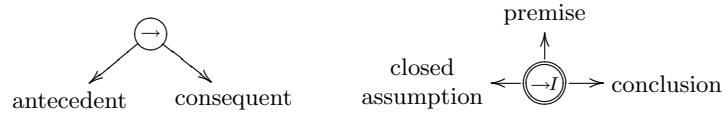


Figure 3.1: Ports

The left image of Figure 3.1 shows the ports of an implication node, the right image the ports of an implication introduction node. The information contained is as follows:

$$\begin{aligned} \text{ports}(\rightarrow) &= \{\text{antecedent}, \text{consequent}\} \\ \text{ports}(\rightarrow I) &= \{\text{closed assumption}, \text{premise}, \text{conclusion}\} \end{aligned}$$

Each proof graph uses the sets LABELS and PORTS, and the *ports* function. Individually, graphs have a set of nodes, a set of indices, and functions that provide the nodes with labels, successors and indices.

Definition 3.4 A proof graph \mathcal{G} is a tuple

$$\langle \text{NODES}, \text{INDICES}, \text{label}, \text{succ}, \text{bind}, \text{index} \rangle$$

where:

NODES a finite set of nodes

INDICES a finite set of indices, disjoint from NODES

label is a function from NODES to LABELS

succ is a function from NODES to partial functions from PORTS to NODES, such that for every node N and every port $p \in \text{ports}(\text{label}(N))$ $\text{succ}(N, p) = N'$ for some node N'

bind is a partial function from the nodes with label VAR or ASS to the nodes labeled $\forall, \exists, \forall I, \exists E, \rightarrow I$ or $\vee E$

index is a partial function from the nodes with label VAR, ASS, or P_i to INDICES, such that $\text{DOM}(\text{bind} \cup \text{index})$ contains all nodes labeled VAR, ASS, or P_i

The *label* function assigns a label to each node. The labels come from one of the two sets of labels; nodes with a label from R-LABELS are called *rule nodes* and nodes with a label from F-LABELS *formula nodes*. The ports of a node N are the ports of its label: $\text{ports}(\text{label}(N))$, which may be abbreviated to $\text{ports}(N)$.

The *succ* function gives a successor for each port on each node. These are the regular connections that contain information such as which nodes are the premises and conclusion of a rule node. The *bind* function links every closed assumption to the rule node that closed it and every bound variable to the quantifier or quantifier inference that binds it. For brevity, from now on the term *binding* will be used for assumption discharge as well as variable binding.

Graphs need to conform to a number of restrictions. To formulate the first one, *acyclicity*, in an intelligible manner, we need the notions of *edge* and *path*. The notion of *backpointer* is provided as well, although we will mostly use the *bind* function itself within definitions.

Definition 3.5 An *edge* is a tuple

$$\langle N \in \text{NODES}, p \in \text{ports}(N), N' \in \text{NODES} \rangle$$

such that $\text{succ}(N, p) = N'$. N is what is called the *source* of the edge, N' the *target*. We use the abbreviation $\langle N, p \rangle$ for $\langle N, p, \text{succ}(N, p) \rangle$.

A *backpointer* is a tuple

$$\langle N \in \text{NODES}, M \in \text{NODES} \rangle$$

such that $\text{bind}(N) = M$.

Definition 3.6 For $M, N \in \text{NODES}$, a path Γ from M to N is a sequence of edges $\langle \gamma_1, \gamma_2, \dots, \gamma_n \rangle$ such that $M = \text{source}(\gamma_1)$ and $N = \text{target}(\gamma_n)$ and for each γ_i ($1 \leq i < n$) $\text{target}(\gamma_i) = \text{source}(\gamma_{i+1})$. The nodes that are on a path, or the nodes that a path *contains*, are the nodes that are the source of the path or the target of some edge within the path.

Since each pair of node and port specifies a unique successor, paths may be abbreviated to a source node and a sequence of ports. A path may be of length zero, but to set it apart from other zero-length paths a source has to be specified.

Restriction 3.7 Proof graphs are *acyclic*, i.e. a proof graph may not contain a path from node N to N consisting of one or more edges. Note that paths solely consist of edges and cannot include backpointers. Since we have a finite amount of nodes, all paths are finite.

Definition 3.8 A root node is a node $N \in \text{NODES}$ such that:

$$\forall M \in \text{NODES} \forall p \in \text{ports}(M) \text{succ}(M, p) \neq N$$

A graph with only one root node is called a *rooted* graph. A rooted path is a path whose source is a root node. In a rooted graph a rooted path can be written as just a sequence of ports. When a graph is required to be rooted, while it isn't, the following definition will save the day:

Definition 3.9 For a node N in a graph \mathcal{G} , \mathcal{G}_N is the subgraph of \mathcal{G} consisting of all nodes reachable from N .

Since backpointers may point to nodes outside of the subgraph \mathcal{G}_N , in some cases it cannot be considered in isolation from \mathcal{G} . When N is a root node, however, Restriction 3.17 will ensure that all backpointers in \mathcal{G}_N point to nodes inside \mathcal{G}_N .

For greater convenience we will use the notion of binding instead of constantly referring to the *bind* relation. This is done by the lemma below; the restriction that follows it ensures a one-to-one correspondence between a binder and the node it binds, and the lemma below that transfers the notion of scope to proof graphs.

Definition 3.10 For all $M, N \in \text{NODES}$ if $\text{bind}(M) = N$ then N *binds* M .

Restriction 3.11 For all $M, N \in \text{NODES}$ if N binds M then $\text{succ}(N, p) = M$ where p is an assumption-type or variable-type port.

Definition 3.12 The *scope* of a node N is the set of all nodes that are on some path from N to a node M that is bound by N .

Disjunction elimination nodes bind two assumption nodes, a left one and a right one. We divide the scope of these nodes into a *left scope* and a *right scope*, which

contain the disjunction elimination node itself and every node on a path from the left minor premise to the left closed assumption, or, respectively, every node on a path from the right minor premise to the right closed assumption. The two scopes of $\exists E$ -nodes are called *assumption scope* and *variable scope*. In both cases ‘the scope’ refers to the union of both scopes.

The bisimulation relation, introduced in Section 2.3, will be described formally below.

Definition 3.13 Two nodes M and N are bisimilar ($M \equiv N$) if there is a relation R such that $R(M, N)$ and for all nodes α, β :

$$\begin{aligned}
R(\alpha, \beta) \Rightarrow & \quad label(\alpha) = label(\beta) \\
& \& \quad \forall p \in ports(\alpha) \ R(succ(\alpha, p), succ(\beta, p)) \\
& \& \quad \text{if } \alpha \in \text{DOM}(bind) \\
& \quad \text{then } \beta \in \text{DOM}(bind) \text{ and } R(bind(\alpha), bind(\beta)) \\
& \& \quad \text{if } \alpha \notin \text{DOM}(bind) \\
& \quad \text{then } \beta \notin \text{DOM}(bind) \text{ and } index(\alpha) = index(\beta)
\end{aligned}$$

For the above definition it doesn’t matter whether M and N are nodes in the same graph or in different graphs. Two graphs are bisimilar if for both graphs every root node is bisimilar to a root node of the other graph.

Individual bisimulation relations, usually indicated as R , need not be reflexive, symmetrical or transitive. Bisimilarity, on the other hand, the fact that a bisimulation can be found, is an equivalence relation.

Lemma 3.14 Bisimilarity is an equivalence relation.

Proof: firstly, the identity relation is a bisimulation, which makes bisimilarity reflexive. Secondly, if R is a bisimulation, then so is the inverse R^{-1} , proving symmetry. Finally, given two bisimulations S and T we show that the composite relation $S \circ T$ is a bisimulation as well.

Suppose that for arbitrary nodes α, β and η we have $S(\alpha, \beta)$ and $T(\beta, \eta)$. Then because $label(\alpha) = label(\beta)$ and $label(\beta) = label(\eta)$ we have $label(\alpha) = label(\eta)$. For all ports p belonging to that label, we have $S(succ(\alpha, p), succ(\beta, p))$ and $T(succ(\beta, p), succ(\eta, p))$, giving us $S \circ T(succ(\alpha, p), succ(\eta, p))$. The case for $bind$ is similar. \square

A very specific notion, the ‘bisimulation modulo identification’ is used to represent substitution within graphs. The relation may be constructed between two nodes that are not bisimilar, and, to mimic transitivity, as well between any two nodes that are bisimilar to the earlier two. The relation inherently represents a partial substitution, i.e. a substitution in which some but not all occurrences of a term or variable are replaced. Because of this and the fact that the relation

is directed, it is not suited to represent multiple substitutions. Fortunately, an inference in natural deduction contains at most one substitution.

Definition 3.15 Two nodes M and N are *bisimilar modulo identification of P and Q* ($M \approx_Q^P N$) if there is a relation R such that $R(M, N)$ (or vice versa), and for all nodes α, β :

$$\begin{aligned}
 R(\alpha, \beta) \Rightarrow \quad & \{\alpha, \beta\} = \{P, Q\} \\
 & \text{—or—} \\
 & \text{label}(\alpha) = \text{label}(\beta) \\
 & \& \forall p \in \text{ports}(\alpha) \ R(\text{succ}(\alpha, p), \text{succ}(\beta, p)) \\
 & \& \text{if } \alpha \in \text{DOM}(\text{bind}) \\
 & \quad \text{then } \beta \in \text{DOM}(\text{bind}) \text{ and } R(\text{bind}(\alpha), \text{bind}(\beta)) \\
 & \& \text{if } \alpha \notin \text{DOM}(\text{bind}) \\
 & \quad \text{then } \beta \notin \text{DOM}(\text{bind}) \text{ and } \text{index}(\alpha) = \text{index}(\beta)
 \end{aligned}$$

Restriction 3.16 The restrictions that ensure that every rule node represents a correct natural deduction inference are given in Appendix B. Figure 3.2 provides an example from that appendix for explanation.

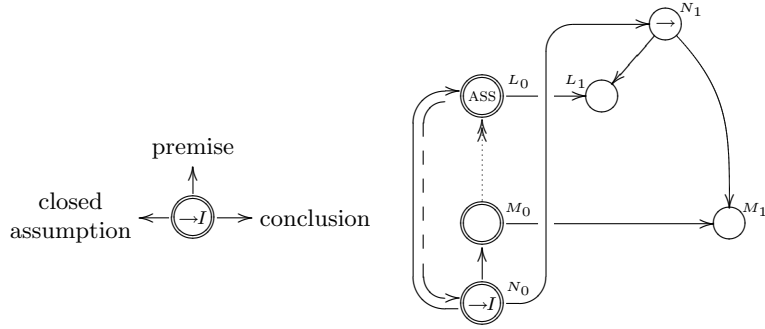


Figure 3.2: Implication introduction

The normal arrows in Figure 3.2 are the edges, the dashed arrow is a backpointer (see also the legend in Appendix B). The dotted arrow roughly indicates the scope of the implication introduction node, but does not impose any restrictions. The left image was already discussed in Figure 3.1, but we need it to know which ports are located where in the right image. The ports of the implication node, which are also required to interpret the illustration, are given in Figure 3.1 as well. The restrictions illustrated by the right image of Figure 3.2 are:

For every (regular) node N , if $label(N) = \rightarrow I$:

$$\begin{array}{ll}
label(M_0) \in \text{R-LABELS} & (M_0 = succ(N_0, \text{premise})) \\
label(L_0) = \text{ASS} & (L_0 = succ(N_0, \text{closed assumption})) \\
label(N_1) = \rightarrow & (N_1 = succ(N_0, \text{conclusion})) \\
bind(L_0) = N_0 & \\
succ(L_0, \text{conclusion}) = L_1 = succ(N_1, \text{antecedent}) & \\
succ(M_0, \text{conclusion}) = M_1 = succ(N_1, \text{consequent}) &
\end{array}$$

The next restriction concerns the correctness of variable binding and assumption closure. For natural deduction the inductive buildup of proofs ensures that assumption closing rules can only close assumptions that are within the subproof(s) of the rule. Likewise, quantifiers can only bind variables within their scope, which is their direct subformula. Since binding and closure within a graph are indicated by a backpointer, we have to verify independently that bound nodes do not occur outside the subgraph of the binding node.

Restriction 3.17 Let N be a binder node and let Γ be the scope of N . Then the following statements must hold:

- for all nodes M, M' such that there is an edge from M to M' , if $M \notin \Gamma$ but $M' \in \Gamma$, then $M' = N$.
- If N is a $\vee E$ -node then the right minor premise and right closed assumption are outside the left scope, and the left minor premise and left closed assumption are outside the right scope of N .
- If N is a $\vee E$ -node or $\exists E$ -node then the major premise of N must be outside Γ .
- If N is a $\forall I$ -node or $\exists E$ -node then its conclusion must be outside Γ .

3.2 Pre-proof graphs

Due to a multitude of factors the translation from a proof to a graph, and the reverse, require several stages. To facilitate this an intermediate notion of graphs is needed, called *pre-proof graphs*. There are two major differences between graphs and pre-proof graphs. The first is that pre-proof graphs may be partial translations of deductions, and in that sense may contain (parts of) deductions and formulas that are still to be translated. These deductions and formulas are contained in the graph by attaching them to so-called *empty nodes*.

To this end there are two extra node labels for pre-proof graphs: ' ε_0 ' (added to the set R-LABELS) and ' ε_1 ' (in F-LABELS). The former has one port, the conclusion port, the latter has none. The *repr* function provides the nodes with these labels with the deduction or formula they are going to represent.

Definition 3.18 A pre-proof graph is a tuple

$$\langle \text{NODES, INDICES, DEDUCTIONS, FORMULAS, } label, succ, bind, index, repr \rangle$$

where NODES, INDICES, *label*, *succ*, *bind* and *index* are as for proof graphs, and:
 DEDUCTIONS is a finite multiset of deductions
 FORMULAS is a finite multiset of formulas
repr is a bijection from empty rule nodes to DEDUCTIONS and from empty formula nodes to FORMULAS

Most of the definitions for proof graphs apply literally to pre-proof graphs, such as the notions of edge, path and root node, but there are a few exceptions. From the restrictions on proof graphs only acyclicity is retained.

Restriction 3.19 Pre-proof graphs are *acyclic*.

Dropping all restrictions except acyclicity means that pre-proof graphs in general do not represent valid natural deduction proofs. This is not a problem because we will only be considered with pre-proof graphs that are the result of the translation procedure to be described later on.

The first definition that requires adaptation is the definition of bisimulation, which has to be expanded to incorporate the *repr* function. There will be no need for bisimulations modulo identification, so that definition will not be replicated for pre-proof graphs.

Definition 3.20 For pre-proof graphs, two nodes M and N are bisimilar ($M \equiv N$) if there is a relation R such that $R(M, N)$ and for all nodes α, β :

$$\begin{aligned} R(\alpha, \beta) \Rightarrow & \quad label(\alpha) = label(\beta) \\ & \& \quad \forall p \in ports(\alpha) \ R(succ(\alpha, p), succ(\beta, p)) \\ & \& \quad \text{if } \alpha \in \text{DOM}(bind) \\ & \quad \text{then } \beta \in \text{DOM}(bind) \text{ and } R(bind(\alpha), bind(\beta)) \\ & \& \quad \text{if } \alpha \notin \text{DOM}(bind) \\ & \quad \text{then } \beta \notin \text{DOM}(bind) \text{ and } index(\alpha) = index(\beta) \\ & \& \quad \text{if } label(\alpha) = \varepsilon_0 \text{ or } label(\alpha) = \varepsilon_1 \\ & \quad \text{then } repr(\alpha) = repr(\beta) \end{aligned}$$

Another definition that cannot be straightforwardly applied to pre-proof graphs is that of binding. Binding in proof graphs is indicated with backpointers, but the problem of placing backpointers is one of the reasons that pre-proof graphs are needed in the first place. Because of this variable letters and assumption markers are retained in pre-proof graphs in the form of indices. Binding in pre-proof graphs may be indicated by backpointers as well as by corresponding indices. However, the definitions required to make this work are complex enough to deserve a section of their own: see Section 3.3. With these definitions, the

notions for proof graphs that depend on the notion of binding, such as *scope*, will be applicable to pre-proof graphs.

A type of pre-proof graph of special interest, are those graphs that do not contain empty nodes or backpointers. We call these *clean* pre-proof graphs.

Definition 3.21 A *clean* pre-proof graph is a pre-proof graph that contains no nodes labeled ε_0 or ε_1 , and for which the *bind* function is empty.

Definition 3.22 The *unfolding* of a clean pre-proof graph \mathcal{G} is a clean pre-proof graph \mathcal{H} such that:

- every rooted path Γ in \mathcal{G} is a node in \mathcal{H} with the same label as $target(\Gamma)$ in \mathcal{G}
- $\langle \Gamma, p, \Delta \rangle$ is an edge in \mathcal{H} whenever Δ is the path in \mathcal{G} that consists of Γ concatenated with the edge $\langle target(\Gamma), p \rangle$ in \mathcal{G}
- $index(\Gamma) = I$ in \mathcal{H} , where the index I is $index(target(\Gamma))$ in \mathcal{G}

Theorem 3.23 A graph \mathcal{G} and its unfolding \mathcal{H} are bisimilar.

Proof: we construct a relation R that holds between every node Γ in \mathcal{H} , which is a path in \mathcal{G} , and the target of Γ , N , which is a node in \mathcal{G} . By the definition of unfolding, the labels of Γ and N are the same, as are any accidental indices $index(\Gamma)$ and $index(N)$.

For every port p associated with the label of Γ and N , the target Δ of the edge $\langle \Gamma, p, \Delta \rangle$ is a path in \mathcal{G} composed of Γ and the edge $\langle N, p, N' \rangle$, where N' is the target of Γ . Now N' is the target of Δ , so that we have $R(\Delta, N')$. \square

Theorem 3.24 Bisimilar clean pre-proof graphs with a single root node have the same unfolding, and such graphs with the same unfolding are bisimilar.

Proof: for one direction, let \mathcal{G} and \mathcal{H} be two clean pre-proof graphs and let R be the bisimulation between them. We adopt the convention of writing rooted paths as sequences of ports. Let \mathcal{G}' and \mathcal{H}' be the unfoldings of \mathcal{G} and \mathcal{H} .

The root of the unfoldings \mathcal{G}' and \mathcal{H}' is the empty path, $\langle \rangle$. This is the basis of the induction.

Let M_R and N_R be the root nodes of \mathcal{G} and \mathcal{H} . By definition, \mathcal{G} and \mathcal{H} are bisimilar only if their root nodes are, therefore R must hold between M_R and N_R .

Let Γ be a rooted path in \mathcal{G} and let Δ be a rooted path in \mathcal{H} such that Γ and Δ are the same sequences of ports. If two nodes are related by a bisimulation, their successors of the same port are related as well. By this observation, the nodes on Γ and Δ , and in specific their targets, are pairwise related by R .

Because of their bisimilarity, the target nodes of Γ and Δ have the same label and, by the definition of unfolding, so do Γ and Δ themselves. Since they have the same label, they have the same set of ports, say p , and thus the same successors, Γ extended with p and Δ extended with p , for all $p \in p$.

For the other direction, let \mathcal{G} and \mathcal{H} have the same unfolding. Then for each node Γ in that unfolding let R hold between $target(\Gamma)$ in \mathcal{G} and $target(\Gamma)$ in \mathcal{H} . By definition their labels and indices correspond, and the successor of the port p in \mathcal{G} is bisimilar to the successor of port p in \mathcal{H} , since both are the target of the same path. \square

3.3 Binding in pre-proof graphs

Within pre-proof graphs there are two ways of binding, with backpointers and with corresponding indices. Binding with backpointers, or rather with the *bind* function, was defined in Definition 3.10. In this section we will define binding with indices.

The different definitions in this section concern binding of variables by quantifiers, binding of assumptions by inference applications, and binding of proper variables by $\forall I$ - and $\exists E$ -applications, adapted for graphs. First we will introduce some convenient notions.

Definition 3.25 For a variable node V , a *candidate binder edge* is any edge $\langle N, p, N' \rangle$ for which one of the following holds:

- $label(N) = \forall$ and the bound variable of N has the same index as V
- $label(N) = \exists$ and the bound variable of N has the same index as V
- $label(N) = \forall I$, $p = \text{premise}$ or $p = \text{proper variable}$, and the proper variable of N has the same index as V
- $label(N) = \exists E$, $p = \text{minor premise}$ or $p = \text{proper variable}$, and the proper variable of N has the same index as V

Definition 3.26 For an assumption node A , a *candidate binder edge* is any edge $\langle N, p, N' \rangle$ for which one of the following holds:

- $label(N) = \rightarrow I$ and the closed assumption of N has the same index as A
- $label(N) = \forall E$, $p = \text{left minor premise}$ or $p = \text{left closed assumption}$, and the left closed assumption of N has the same index as A
- $label(N) = \forall E$, $p = \text{right minor premise}$ or $p = \text{right closed assumption}$, and the right closed assumption of N has the same index as A

- $label(N) = \exists E$, $p =$ minor premise or $p =$ closed assumption, and the closed assumption of N has the same index as A

The reason that edges instead of nodes are specified as candidate binders lies with the last three items in the list above: $\forall E$ -inferences and $\exists E$ -inferences close assumptions in the subproof of some minor premise, and not in all their subproofs; likewise proper variables of $\exists E$ -inferences occur only in the subproof of the minor premise.

Definition 3.27 In a pre-proof graph \mathcal{G} , a node V is bound by the source node of the last candidate binder edge on a path from a root node to V .

This definition is additional to the earlier definition for binding, which applies to nodes with a backpointer to a quantifier or discharging rule node. The next definition, as a counterpart to the previous, defines when variables and assumptions are considered free in graphs.

Definition 3.28 An assumption or variable node V is *free* at a node N if there is a path from N to V that does not contain a candidate binder edge of V , or if V has a backpointer instead of an index, if there is a path from N to V that does not cross the binder of V . An index is called free at a node N if there is a variable or assumption node free at N with that index. A node is free in a graph if it is free at a root node of the graph.

Crucially, each of the above definitions refers to ‘a path’, while sharing is all about having multiple paths towards a node. Later proofs will therefore focus on showing that in the graph translation of a deduction, assumptions and variables are uniquely bound, and never bound and free at the same time and place.

To be thorough, the procedure for finding pure variables from Section 1.4 (see page 17) will also be adapted for graphs.

Definition 3.29 A *v-edge* is an unordered pair of rule nodes $\{N, M\}$ for which one of the following statements is true:

- $label(N) \in \{\wedge I, \wedge EL, \wedge ER, \vee IL, \vee IR, \rightarrow I, \rightarrow E, \perp E, \forall I, \forall E, \exists I\}$, and for some non-conclusion port $p \in ports(N)$ $succ(N, p) = M$
- $label(N) \in \{\forall E, \exists E\}$ and $succ(N, p) = M$ for some $p \in \{\text{left minor premise, right minor premise, minor premise}\}$
- there is some node L such that $label(L) = \rightarrow E$, $succ(L, \text{major premise}) = N$ and $succ(L, \text{minor premise}) = M$
- there is some node L such that $label(L) \in \{\forall E, \exists E\}$, $succ(L, \text{major premise}) = N$ and $succ(L, p) = M$ for some assumption-type port p

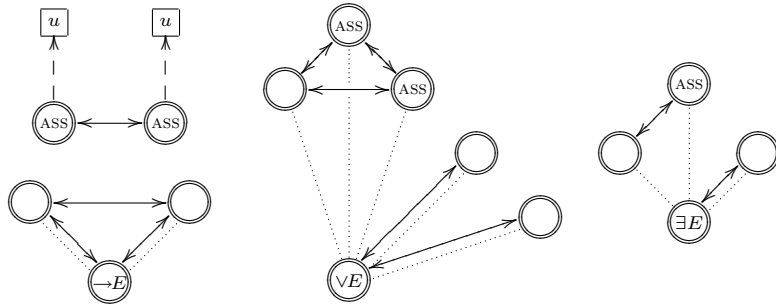


Figure 3.3: The v-edges that differ from regular edges

- N and M are assumption nodes bound by the same rule node

In most cases v-edges correspond to regular edges, albeit undirected. As an illustration of the above definition, Figure 3.3 shows where v-edges also occur: between assumptions of the same class (although the picture only shows that they have the same indices), between the major and minor premise of an implication elimination, and between assumptions and major premises of $\exists E$ -nodes and $\forall E$ -nodes. Regular edges have been drawn as dotted lines.

Definition 3.30 A *v-path* is a sequence of v-edges $\langle \gamma_1 \dots \gamma_n \rangle$ such that each pair of successive v-edges has a node in common, i.e. $\gamma_i \cap \gamma_{i+1} \neq \emptyset$ for $1 \leq i < n$.

Definition 3.31 A variable node V is *purely* bound by a node N if V is the proper variable of N , or if all of the following statements hold:

- N is labeled $\forall I$ or $\exists E$
- N binds V
- there is a v-path Γ from the premise of N , or the minor premise if N is an $\exists E$ -node, to a rule node M , such that
- V is free at M
- at every node on Γ a variable node with the same index as V occurs free
- no node on Γ is labeled $\forall I$ or $\exists E$ and has a proper variable with the same index as V

To summarize what happens above: a v-edge is undirected and links two rule nodes that, if they were inferences in a deduction, would share a formula in their conclusion. Multiple v-edges are linked together in a v-path, simulating the progression of marking formulas. A v-path does not start at the node whose proper variable is sought itself, but at the premise or minor premise. If not, the

v-path could immediately proceed towards a predecessor, in which the proper variable may not occur in a regular deduction.

The following theorems will explain to what extent binding is preserved amongst a graph and its unfolding. The results are independent of the correctness of any binding relation.

Lemma 3.32 Given two edges $\langle N, p, N' \rangle$ and $\langle M, p, M' \rangle$ and an assumption or variable node V , if N and M are bisimilar and either edge is a candidate binder edge for V , then the other is a candidate binder edge for V as well.

Proof: again, bisimilarity requires N and M to have matching labels and bisimilar successors; the closed assumptions of N and M thus have corresponding indices. \square

Theorem 3.33 Let \mathcal{G} and \mathcal{H} be bisimilar rooted graphs, let Γ be a rooted path in \mathcal{G} . Then if the n th node—or the n th edge—is the last candidate binder for $target(\Gamma)$ on Γ in \mathcal{G} , the same holds in \mathcal{H}

Proof: the two paths, Γ in \mathcal{G} and Γ in \mathcal{H} , have bisimilar source nodes, which implies that the nodes on the two paths are pairwise bisimilar; in particular, the n th node. By Lemma 3.32, the n th node or edge is a candidate binder for $target(\Gamma)$ in \mathcal{H} as well. Any candidate binder for $target(\Gamma)$ in \mathcal{G} is a candidate binder for $target(\Gamma)$ in \mathcal{H} , since they have the same index. \square

What the above lemma shows is that, although not true generally, in certain circumstances binding will be preserved under bisimulation. Particularly when dealing with the unfoldings of a graph, in which there is always exactly one rooted path to a node, the lemma will prove useful.

4 Transformations

The following scheme is a summary of the translation process from a deduction to a proof graph.

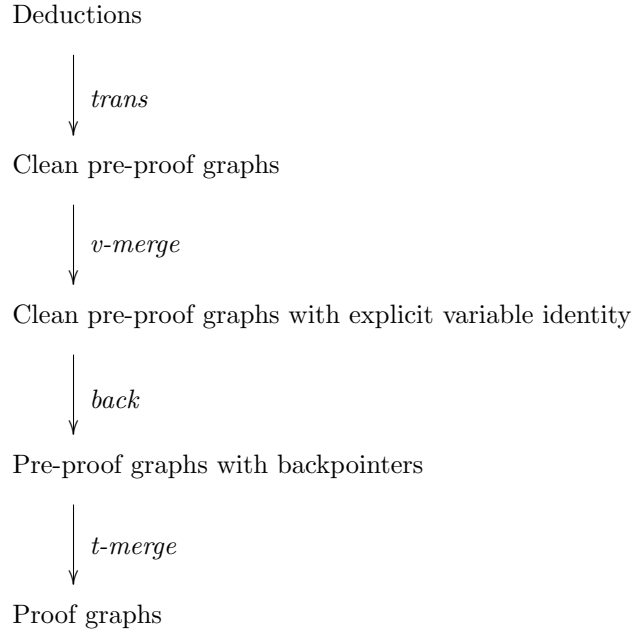


Figure 4.1: Summary of the translation from proofs to graphs

The first stage is the actual translation process, a procedure called *trans*. It turns the deduction into a pre-proof graph in which binding is indicated with indices.

If a deduction that has, say, n occurrences of x is translated by the *trans* procedure, this will most likely result in a graph that has approximately n variable nodes with index x . However, in proof graphs all those variables should be represented by a single node. Therefore, *v-merge* merges separate variable nodes that are bound by the same quantifier node, and merges closed assumption nodes that belong to the same assumption class.

Next, the indices of the merged assumption and variable nodes are replaced by backpointers. This operation is abbreviated as '*back*'. Finally, in the operation called *t-merge*, the proper terms of $\exists I$ -nodes and $\forall E$ -nodes are merged. The bisimulation modulo identification relation, which represents substitution, identifies two nodes, the one that represents the substitute, and the one that is substituted for. To represent the substitution of a variable by a proper term, then, all instances of that term have to be represented by a single node.

The reverse translation, from proof graph to deduction, consists of the following two steps:

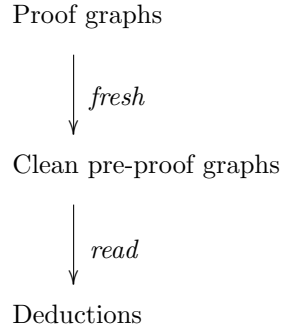


Figure 4.2: Summary of the translation from graphs to proofs

The first, the *fresh* operation, replaces backpointers with fresh indices. The second reads the contents of the graph, starting at the root node. This chapter will present these five operations in detail.

4.1 The *trans* operation

The *trans* operation translates a deduction to a graph one node at a time. The nodes that still need translation are called *empty* nodes, and come in two types: empty rule nodes, labeled ε_0 , and empty formula nodes, labeled ε_1 . The translation step that translates the empty node N is called \textit{trans}_N . Which steps are possible is described by the schemes in Appendix D. The *repr* function gives every empty node a deduction or a formula, whose last inference or primary connective decides which scheme applies. We will explain the schemes in Appendix D with the help of the example depicted in Figure 4.3.

In the example the node N_0 is translated, expressed by the subscript N_0 in \textit{trans}_{N_0} . The *repr* function is indicated by the dotted lines starting at the empty nodes, labeled ε_0 (empty formula nodes would be labeled ε_1). The conclusion node of N_0 , N_1 , already has been translated. However, later on we will show that, for all intents and purposes, we may treat N_1 as the translation of an empty formula node representing $A \rightarrow B$.

The nodes L_1 and M_1 are the antecedent and consequent of node N_1 . They are not necessarily empty nodes, as they may already have been translated. In the images, empty nodes should not be confused with *arbitrary* nodes. Nodes L_1 and M_1 are arbitrary formula nodes, but as we will show we may treat them as empty nodes representing A and B , which is indicated by the A and B in parentheses on the bottom of the picture.

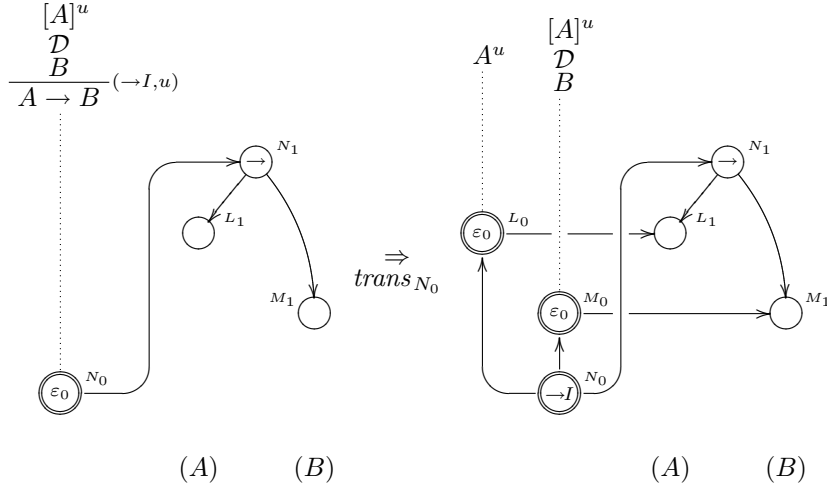


Figure 4.3: The translation scheme for implication introduction

In the translation step the node N_0 is given the label $\rightarrow I$, according to the last inference application in $\text{repr}(N_0)$. New ε_0 -nodes L_0 and M_0 are added, representing the closed assumption class and the premise of the inference. Bookkeeping involves adding A^u and \mathcal{D} to the multiset DEDUCTIONS, and removing an instance of $\text{repr}(N_0)$.

The conclusion-ports of L_0 and M_0 are linked to L_1 and M_1 . Although we don't know what the last inference of $\text{repr}(L_0)$ and $\text{repr}(M_0)$ is, and thus what labels L_0 and M_0 will receive from the future translation steps trans_{L_0} and trans_{M_0} , the conclusion nodes L_1 and M_1 have to be present if we want to share them. On the other hand, all empty nodes have to be new nodes: the translation process cannot take binding into consideration, so although some empty nodes may look like they represent exactly the same formula, binding of some of the variables may change that.

Because trans operates on nodes and not on deductions, we need something extra to get the translation started. The special translation step called trans_0 is defined such that for a deduction \mathcal{D} with conclusion C , $\text{trans}_0(\mathcal{D})$ is a couple of an empty rule node N_0 and an empty formula node N_1 , such that $\text{repr}(N_1)$ is the conclusion C , $\text{repr}(N_0)$ is the deduction \mathcal{D} , and $\text{succ}(N_0, \text{conclusion}) = N_1$, as illustrated in Figure 4.4. The other initial translation step, trans_1 , is provided for the translation of isolated formulas.

A complete translation of a graph requires that each empty node is translated, which is complicated by the fact that each translation step may introduce new empty nodes. To formalize this translation process we introduce the notion of a *translation sequence*: a series of graphs, each of which is translated one step further than the previous.

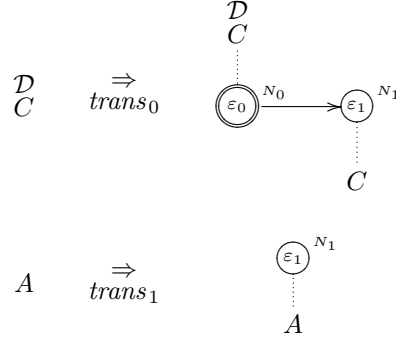


Figure 4.4: The initial translation steps

Definition 4.1 A *translation sequence* is a sequence of pre-proof graphs $\langle \mathcal{G}_1, \dots, \mathcal{G}_n \rangle$ such that for all $1 \leq i < n$, $\mathcal{G}_{i+1} = trans_N(\mathcal{G}_i)$ for some $N \in \mathcal{G}_i$. A translation sequence is *complete* if the first graph in the sequence is $trans_0(\mathcal{D})$ for some deduction \mathcal{D} or $trans_1(A)$ for some formula A , and there is no possible translation step $trans_M(\mathcal{G}_n)$.

Definition 4.2 The *translation* $trans(\mathcal{D})$ of a deduction \mathcal{D} is a graph for which $\langle trans_0(\mathcal{D}), \dots, trans(\mathcal{D}) \rangle$ is a complete translation sequence. The translation of a formula A is a graph $trans(A)$ such that $trans_1(A) \dots trans(A)$ is a complete translation sequence.

The lemma below states some essential features of translation sequences. In particular, each node can only be translated once, and remains unchanged in other translation steps.

Lemma 4.3 Let $\langle \mathcal{G}_1, \dots, \mathcal{G}_n \rangle$ be a translation sequence, and let $\mathcal{G}_{i+1} = trans_N(\mathcal{G}_i)$, where N is a node in \mathcal{G}_i , be a translation step in that sequence. Then for all graphs \mathcal{G}_j , if $1 \leq j \leq i$ and $N \in \mathcal{G}_j$ then N has the same label, successors and represented formula or proof in \mathcal{G}_j as in \mathcal{G}_i ; if $i < j \leq n$, then N has the same label, successors and index in \mathcal{G}_j as in \mathcal{G}_{i+1} .

Proof: in each of the translation schemes in Appendix D, new nodes may be introduced, but nodes are never removed. In a translation step $trans_N$, N is the only node for which the label and successors change. A node may be introduced and translated at most once in a translation sequence, and although it may gain additional predecessors, it remains otherwise unchanged in all other translation steps. \square

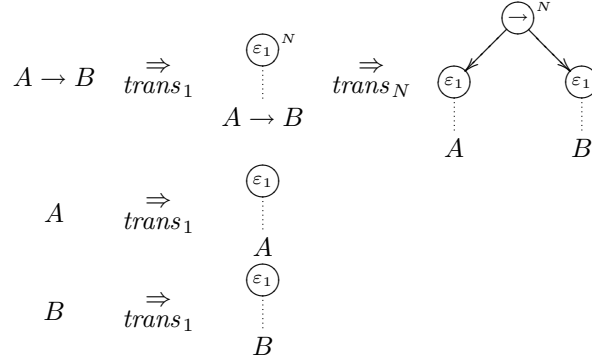
New empty nodes are introduced with their conclusion nodes already in place, to allow the latter to be shared. As a result, the left hand side of each translation

scheme deals with an empty rule node with attached conclusion. Because the scheme itself does not influence the translation of that conclusion, we have to verify independently that it is the right conclusion for the empty node.

More specifically, we will show that the conclusion node of an empty node, when fully translated, is bisimilar to the translation of that nodes represented deduction. In other words, each pair of an empty rule node and a conclusion node may be treated as though they were a direct translation of the represented deduction of the empty node. The translation scheme in Figure 4.3 will serve as an example.

Lemma 4.4 Let $\langle \mathcal{G}_1, \dots, \mathcal{G}_n \rangle$ be a complete translation sequence and let N_0 be a rule node in \mathcal{G}_n . In the graphs \mathcal{G}_i in which N_0 occurs as empty node, let $repr(N_0)$ be a deduction \mathcal{D} . Let N_1 be the conclusion of N_0 . Then the subgraph of N_1 in \mathcal{G}_n is isomorphic to $trans(C)$, with C conclusion the conclusion of \mathcal{D} .

Proof: by induction. We show by the example of Figure 4.3 that the correctness of the represented conclusions is passed on through the graph from the root. The induction hypothesis is that the node N_1 is isomorphic to $trans(A \rightarrow B)$. Below are the first steps in the translations of $A \rightarrow B$, A and B .



Clearly, if N_1 is isomorphic to the node N above, then its antecedent and consequent are isomorphic to $trans(A)$ and $trans(B)$, respectively. \square

Except the conclusion node, all other features of an empty rule node are determined by the represented deduction. Proving that the conclusion matches that in the represented deduction leads to the corollary that nodes in a graph translate the same way as they would in isolation:

Corollary 4.5 If N_0 is an empty rule node in some graph in a complete translation sequence $\langle \mathcal{G}_1, \dots, \mathcal{G}_n \rangle$, then in the result of the translation \mathcal{G}_n , N_0 is isomorphic to $trans(repr(N_0))$. \square

The above lemma and corollary are needed because the translation of a rule node is independent of the translation of its conclusion. The conclusion node

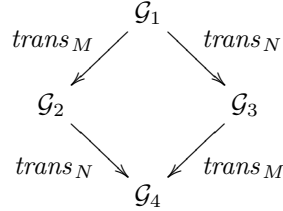
was added to the graph by some predecessor of the rule node, and not by the rule node itself. In this process, the information that it is the right conclusion for the proof that the rule node represents was essentially lost. That information is provided again by the above lemma and corollary.

A further complication in the translation process is that empty nodes may be translated in any order, with the possibility that a different order of translation steps results in a different translation. The next two lemmas intend to show that there is exactly one graph translation for each deduction.

Lemma 4.6 The *trans* operation is confluent, i.e. if for some graph \mathcal{G} there are two complete translation sequences $\mathcal{G} \dots \mathcal{G}'$ and $\mathcal{G} \dots \mathcal{G}''$, then \mathcal{G}' and \mathcal{G}'' are identical up to the names of their nodes (isomorphic).

Proof: let $trans_M$ and $trans_N$ be successive steps in a translation sequence. Three cases may be distinguished: one, $trans_M$ introduces N ; two, N has the label $\wedge I$, $\vee IL$, $\vee IR$ or $\rightarrow I$ and M is the conclusion of N ; three, otherwise. In the first and second case, $trans_M$ and $trans_N$ may not be interchanged, and there are no translation sequences in which N is translated before M . Either N does not exist before the step $trans_M$, or it does not match a translation scheme.

In the third case, the step $trans_M$ leaves node N untouched, and $trans(N)$ may add a predecessor to M , but does not change it. In particular, both N and M are empty nodes in the graph on which $trans_M$ is applied. The following scheme displays the situation:



Although \mathcal{G}_2 and \mathcal{G}_3 are different graphs, the step $trans_M$ may be applied to \mathcal{G}_3 since it is an empty node in that graph. If the nodes introduced in both steps $trans_M$ receive the same names, and the same goes for both steps $trans_N$, then the left and right path in the above schematic produce the same graph \mathcal{G}_4 . \square

Another result needed to show that the *trans* operation is a suitable translation mechanism is that it terminates; and that when it terminates it has produced an acyclic graph, in which all nodes it had to translate are actually translated.

Lemma 4.7 The *trans* operation terminates in an acyclic graph that contains no empty nodes.

Proof: termination is guaranteed by the DEDUCTIONS and FORMULAS multisets: in each translation step, a deduction or formula is erased, and replaced with a finite number of smaller subdeductions or subformulas.

That the resulting graph is acyclic can be deduced from properties of the translation schemes. Each scheme has a root node, and no paths enter that scheme other than through that node. Whenever paths are added to an existing node, there already was a connection to that node in the same direction, from the root of the scheme, so no new connections are made that could form a cycle.

To show that all empty nodes can be translated, we remark that the only restriction on translating an empty node are the requirements imposed on the conclusion nodes in the schemes for $\wedge I$, $\vee IL$, $\vee IR$ and $\rightarrow I$. Considering Lemma 4.4, it is easily seen that these requirements are always met. \square

The definition of proof graphs requires that each variable node or assumption node is bound at most once, and that no node is both bound and free at the same time. The following lemma and theorem show that a freshly translated graph meets these requirements.

Lemma 4.8 In a graph $trans(\mathcal{D})$ for some deduction \mathcal{D} , let N and M be nodes such that M binds N . Then N is not free in \mathcal{G} , and if M' binds N then $M = M'$.

Proof: again we use that each translation scheme has exactly one root node, and that as a consequence the translation process does not allow arbitrary paths to join. If a translated graph contains two paths to a variable or assumption node N , those paths are separated and joined again within one translation scheme.

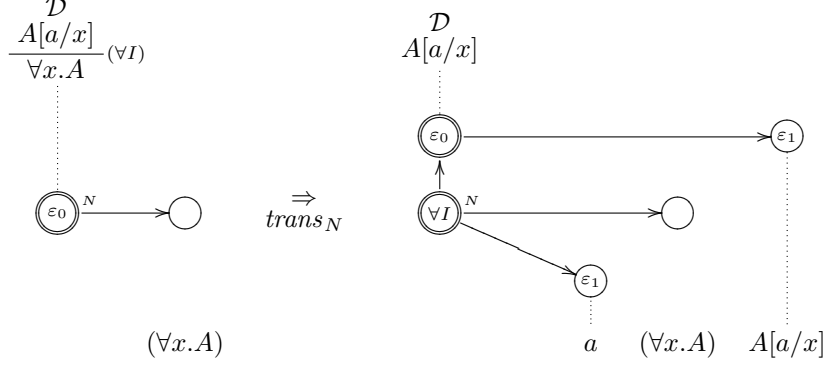
Inspection of the schemes will then show that whenever two paths converge on one node, that node is a formula node, and on the part that they are separate, none of those paths contains a quantifier node or an empty formula node (which might later become a quantifier node). \square

Theorem 4.9 In a pre-proof graph $trans(\mathcal{D})$, let A and A' be two assumption nodes bound by the same node N . Let V and V' be variable nodes with the same index that are free at A and A' respectively. Then if V and V' are bound by M and M' , $M = M'$.

Proof: since V is free in A , let Δ be a path from A to V free of candidate binders for V . By Lemma 4.8 all other paths from A to V are free of candidate binders as well. Let Δ' be such a path from A' to V' . Let Γ be a path from N to A and Γ' from N to A' .

Firstly, M and M' have to be $\forall I$ - or $\exists E$ -nodes, since they have to be predecessors of A and A' . These nodes don't share the relevant conclusion nodes: the only access to their subgraphs is through the node itself. Because of this, if M and M' are on the same path, the first one on the path can't bind variable nodes in

the subgraph of the second one, unless $M = M'$.



Suppose M is on Γ . Then the assumption A with free variable node V is in the subgraph of M , while M is a $\forall I$ -node or $\exists E$ -node whose proper variable has the same index as V . This construction is ruled out by the restrictions on $\forall I$ - and $\exists E$ -applications, which state that the proper variable may not occur free in open assumptions.

By this argument M and M' cannot be on Γ or Γ' , and must be predecessors of N . Since there is only one rooted path to N in the fresh translation $trans(\mathcal{D})$, by the argument above $M = M'$. \square

4.2 The *merge* operations

For technical as well as philosophical reasons explained earlier, we preferred closed assumption classes, and variables bound by the same quantifier or inference, to be represented by a single node. Obviously a graph resulting from the *trans* operation does not yet conform to this requirement. What needs to be done, is that different assumption and variable nodes that are bound by the same binder node, have to be merged into one. Additionally, occurrences of the proper terms of $\forall E$ -nodes and $\exists I$ -nodes have to be represented by a single node, because of the nature of the bisimulation modulo identification.

Collapsing two variable nodes can be as straightforward as redirecting all edges pointing at one node towards the other, and removing the former. Assumption nodes, on the other hand, have a conclusion port that connects to an entire formula subgraph, which cannot be removed since it may be shared, and terms may consist of more than just one node. Of course an assumption node that results after a merger cannot have two separate conclusions: conclusion subgraphs of assumptions and subgraphs that make up a proper term have to be collapsed in their entirety.

Fortunately, the assumptions we need to merge are in one assumption class, and thus have the same formula. The requirement on natural deduction that

assumptions with the same marker must have the same formula, even if they are not bound by the same inference, makes things even simpler.

Lemma 4.10 In a newly translated pre-proof graph, assumption nodes that have the same index are bisimilar.

Proof: consider an assumption A^u . Since natural deduction requires assumptions with the same marker to have the same formula, a proof in which A^u occurs may not contain assumptions B^u (where $B \neq A$). Consequently, a translation sequence that contains empty rule nodes representing A^u cannot contain empty rule nodes representing B^u . Corollary 4.5 guarantees that each empty rule node representing A^u will be bisimilar to $trans(A^u)$ upon completion of the translation. \square

Of course two occurrences of a proper term t will translate to bisimilar graphs as well. What is needed, then, is an operation that collapses a graph along the lines of a certain bisimulation. We will define the *merge* operation to do just that. To complete this section we will show that the different mergers of assumption classes, variables and terms do not interfere with each other.

Definition 4.11 For any pre-proof graph \mathcal{G} and bisimulation R on \mathcal{G} , the graph $merge_R(\mathcal{G})$ is the graph \mathcal{H} such that:

$$\begin{aligned}
NODES_{\mathcal{H}} &= \text{the set of smallest non-empty subsets of } NODES_{\mathcal{G}} \\
&\quad \text{closed under } R^= \\
INDICES_{\mathcal{H}} &= INDICES_{\mathcal{G}} \\
label_{\mathcal{H}} &: label_{\mathcal{H}}(N) = label_{\mathcal{G}}(n) \text{ for all } n \in N \\
succ_{\mathcal{H}} &: succ_{\mathcal{H}}(N, p) = N' \text{ whenever } succ_{\mathcal{G}}(n, p) = n', n \in N \\
&\quad \text{and } n' \in N' \\
bind_{\mathcal{H}} &: bind_{\mathcal{H}}(N) = N' \text{ whenever } bind_{\mathcal{G}}(n) = n', n \in N \\
&\quad \text{and } n' \in N' \\
index_{\mathcal{H}} &: index_{\mathcal{H}}(N) = I \text{ whenever } index_{\mathcal{G}}(n) = I \text{ for } n \in N \\
repr_{\mathcal{H}} &: repr_{\mathcal{H}}(N) = \mathcal{D} \text{ whenever } repr_{\mathcal{G}}(n) = \mathcal{D} \text{ for } n \in N
\end{aligned}$$

Formally, the *merge* operation takes a *quotient* of a graph. We add a few further remarks and basic lemmas to clarify the definition.

The nodes of \mathcal{H} are sets of nodes from \mathcal{G} that R shows to be bisimilar. Although R is not necessarily symmetric or transitive, bisimilarity is, as shown in Lemma 3.14. Instead of taking closure under R we therefore take closure under its reflexive–transitive–symmetric closure, $R^=$. (Technically, the symmetric closure of R would suffice, since set closure is inherently transitive and reflexive.)

The following lemmas are used to show that the *merge* operation creates a valid graph. Even some very basic features have to be demonstrated, such as that the successor and binder functions are indeed functions.

Lemma 4.12 Any node n from the graph \mathcal{G} occurs in exactly one node N in the graph $merge_R(\mathcal{G})$, where R may be any bisimulation on \mathcal{G} .

Proof: n occurs in at least one node N since it occurs in $\{n\}$. Because $NODES_{\mathcal{G}}$ is finite, the subset $\{n\}$ can always be expanded to meet the closure demand, without causing trouble by growing infinite. To see that n occurs in at most one N : let \mathcal{H} be $merge_R(\mathcal{G})$ and suppose two nodes from \mathcal{H} , M and N , have a non-empty intersection containing k , a node from \mathcal{G} . Then for all nodes $m \in M$ and $n \in N$, closure under $R^=$ gives us $R^=(k, m)$ and $R^=(k, n)$. Transitivity gives $R^=(m, n)$, and finally closure under $R^=$ and extensionality imply $M = N$. \square

Lemma 4.13 The successor relation in $merge_R(\mathcal{G})$ is a function (where \mathcal{G} may be any pre-proof graph and R any bisimulation on \mathcal{G}).

Proof: the successors of a node $N \in merge_R(\mathcal{G})$ using port p are specified as all nodes that contain the successor at port p of some $n \in N$. For $merge_R(\mathcal{G})$ to be a valid graph we need to show that the successors of N are unique.

The previous lemma showed that a node from \mathcal{G} is an element of exactly one node in $merge_R(\mathcal{G})$; each individual successor n' occurs in exactly one node N' . Now we show that for a node $N \in merge_R(\mathcal{G})$, for all nodes $n \in N$ the p -successors $succ(n, p) = n'$ are elements of the same node N' .

Let n_1 and n_k be arbitrary nodes in N . Since N is a smallest set closed under $R^=$ we have $R^=(n_1, n_k)$. By Lemma 3.14 $R^=$ is a bisimulation, and therefore $R^=$ must hold between the p -successors of n_1 and n_k as well: for $n'_1 = succ(n_1, p)$ and $n'_k = succ(n_k, p)$ we have $R^=(n'_1, n'_k)$. Consequently, n'_1 and n'_k are elements of the same node N' . \square

Theorem 4.14 For any bisimulation R on a pre-proof graph \mathcal{G} without empty nodes, the pre-proof graph $merge_R(\mathcal{G})$ is well-formed.

Proof: the above lemma can easily be replicated for the other functions, *bind* and *index*. Merging a graph with empty nodes would require a recalculation of the multisets DEDUCTIONS and FORMULAS. \square

Now that it has been proven to work, the next lemma shows that the *merge* operation preserves bisimilarity.

Lemma 4.15 For any bisimulation R on a graph \mathcal{G} , \mathcal{G} and $merge_R(\mathcal{G})$ are bisimilar.

Proof: we prove that the relation $\{\langle n, N \rangle \in NODES_{\mathcal{G}} \times NODES_{merge_R(\mathcal{G})} \mid n \in N\}$ is a bisimulation.

Suppose $n \in \mathcal{G}$ is an element of $N \in merge_R(\mathcal{G})$. By the definition and lemmas above, N has the same label as n . For a port p , by demonstrating that the *succ* relation in $merge_R(\mathcal{G})$ is a function we have shown that for all n , if $succ(n, p) = n'$

and $n \in N$, then $n' \in N'$ where $N' = \text{succ}(N, P)$. Again, the case of the *bind* function is similar.

Moreover, since by Lemma 4.12 each node $n \in \mathcal{G}$ can be an element of only one $N \in \text{merge}_R(\mathcal{G})$, the bisimulation is a function. \square

To summarize, merging assumptions and variables is done by creating a bisimulation with the closed assumption or bound variable of the binder node. The graph can then be collapsed along the lines of those bisimulations. Since merging preserves bisimilarity there is a relative freedom to the order in which different assumptions and variables are merged. We have chosen to do them all at once, in an operation called *v-merge*. The merging of terms, however, is performed separately, in the operation *t-merge*.

Definition 4.16 For a pre-proof graph \mathcal{G} $v\text{-merge}(\mathcal{G})$ is $\text{merge}_R(\mathcal{G})$ where R is the following bisimulation on \mathcal{G} : $R(A, B)$ holds for nodes A and B if there is a node N such that A is bound by N , and B is the the corresponding bound variable, proper variable or closed assumption of N .

Theorem 4.17 Let \mathcal{G} be the graph $\text{trans}(\mathcal{D})$ for some deduction \mathcal{D} . Between \mathcal{G} and $v\text{-merge}(\mathcal{G})$ binding is preserved, and binding in $v\text{-merge}(\mathcal{G})$ is unique.

Proof: the way preservation of binding for graphs without backpointers is expressed, is as follows: if the n th node on a rooted path Γ binds $\text{target}(\Gamma)$ in \mathcal{G} , then the same must hold in $v\text{-merge}(\mathcal{G})$. By Theorem 3.33 this follows directly from the fact that \mathcal{G} and $v\text{-merge}(\mathcal{G})$ are bisimilar.

That binding remains unique as well, in the sense that for a freshly translated graph after the *v-merge* step each node is still bound at most once, is ensured by Theorem 4.9. This theorem stated that free variable nodes with the same index within the same assumption class are bound by the same node. Merging those assumptions thus cannot create the double binding of any of those nodes. \square

The above theorem deals with graphs in which some—if not all—binding is expressed with indices. Binding with indices is not taken into account in a bisimulation relation, but binding with backpointers is—which is exactly what is needed for the merging of terms.

In the rules for \exists -introduction and \forall -elimination a variable x is substituted with the term t , resulting in the formula $A[t/x]$. In principle, all free variables in t remain free in the occurrences of t in $A[t/x]$. This ensures that they can safely be merged in the graph; if a free variable node and a bound variable node would be merged, then the resulting node would be both bound and free.

The catch here is that the formula A may already contain occurrences of t , and nothing prohibits those from having bound variables that are free in t itself. Yet if binding is not taken into consideration, those occurrences of t will be bisimilar

to the ones that are substituted for x . The most straightforward solution is to add backpointers to the graph first, so that binding is taken into account in the bisimulation relation. The t -merge operation is therefore processed after backpointers have been placed.

Definition 4.18 For a pre-proof graph \mathcal{G} , $t\text{-merge}(\mathcal{G})$ is the graph $\text{merge}_R(\mathcal{G})$ where R is the following bisimulation: for each node N labeled $\forall E$, $R(A, B)$ holds whenever A is in the subgraph of the conclusion of N , B is the proper term of N and $A \equiv B$; for each node N labeled $\exists I$, $R(A, B)$ holds whenever A is in the subgraph of the conclusion of the premise of N , B is the proper term of N and $A \equiv B$.

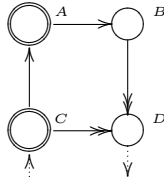
To make this operation viable, it remains to show that the free variables in a substitute term t are indeed bound by the same node, when taking the binding of proper variables into account. This is proven by showing that also in graphs, the free variable occurrences of the same letter in a formula can only be bound all at once.

Theorem 4.19 Let M be a formula node in a pre-proof graph $\text{trans}(\mathcal{D})$ and let V and V' be different variable nodes with the same index, such that V and V' are both free at M . Then if a node N binds V , it also binds V' .

Proof: since N binds V there is a path Γ on which some edge γ is the last candidate binder edge for V . First, suppose M is on Γ . Then γ must lie on the subpath of Γ from N to M , because V is free at M . That subpath combined with the path from M to V' makes that N binds V' .

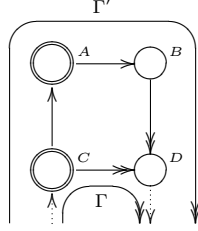
Next suppose that M is not on Γ . Then Γ and the path from M to V , let's call it Δ , are different paths to the same node, V . We will look at the properties of the translation schemes to show that there is a path from N to V .

In the translation schemes all converging paths have split within the same scheme, at a rule node. No paths converge after crossing a quantifier node, so N must be a $\forall I$ -node or $\exists E$ -node. Schematically a scheme can be seen as follows:



Now let the node D be the point at which Γ and Δ converge, which may be V itself. Since the schemes of $\forall I$ and $\exists E$ do not produce converging paths (except at the conclusion of $\exists E$), the node C cannot be the binder node N , nor any other node that is in a position to bind V . Also, the node A cannot be the

binder of N , since both $\forall I$ and $\exists E$ do not bind any nodes in their conclusion.



Let Γ be the path indicated above, from N through C and D to V . We construct a path Γ' , from N through C , A , B and D to V (it may happen that Γ and Γ' are switched). Now either M is on Γ , proving the theorem, or it is not, in which case the argument can be repeated for Γ' and Δ . Since Δ is finite, and with each repetition the new path Γ^i shares a larger part of its tail with Δ , eventually Δ will be a subpath of some path Γ^n and M will be on Γ^n . \square

Finally, we will show that also under the t -merge operation, binding is preserved.

Theorem 4.20 If the n th node on a rooted path Γ binds $target(\Gamma)$ in \mathcal{G} , then the same holds in t -merge(\mathcal{G}).

Proof: let M be the n th node on Γ and let M' be $bind(target(\Gamma))$ in t -merge(\mathcal{G}), where both are the same node m in \mathcal{G} . The bisimulation between both graphs is the set membership relation. Since the nodes on Γ are pairwise bisimilar, we have $m \in M$; for both targets of Γ their binders are required to be bisimilar as well, providing $m \in M'$. By Lemma 4.12 $M = M'$.

This suffices to show that if \mathcal{G} conforms to Restriction 3.17, then so does t -merge(\mathcal{G}). The requirements of that restriction are, roughly, that every rooted path to a node that is bound crosses the binder. \square

4.3 Adding and removing backpointers

The actual step in the translation that adds the backpointers is rather un spectacular. In the graph resulting from the v -merge procedure each binding node has only one bound node, to which it is connected through the closed assumption port, bound variable port or deactivated variable port. It now suffices to remove the index of that variable or assumption node, and add a backpointer towards the binder node.

Definition 4.21 For any pre-proof graph \mathcal{G} $back(\mathcal{G})$ is the graph obtained by removing the index of a node M and adding $bind(M) = N$ whenever M is bound by N .

Optionally, tidying up would involve removing obsolete indices from the INDICES set. As with the *trans* and *merge* operations, we will prove that the *back* operation preserves binding.

Theorem 4.22 Let N and M be nodes in a pre-proof graph \mathcal{G} with unique binding. Then N binds M in $\text{back}(\mathcal{G})$ if and only if N binds M in \mathcal{G} .

Proof: by definition. Unique binding is required if the *bind* relation is to be a function in $\text{back}(\mathcal{G})$. \square

Theorem 4.23 Let \mathcal{G} be a clean pre-proof graph in which variables and assumptions are uniquely bound, and in which no node is both bound and free. Then $\text{back}(\mathcal{G})$ conforms to Restriction 3.17 (repeated below).

Proof: let the node N bind the node V in \mathcal{G} . Then on all rooted paths to V in \mathcal{G} a candidate binder edge γ with source N can be found. Let M and M' be nodes such that M is outside and M' is inside the scope of N ; which means M is not on a path from N to V , but M' is. Furthermore, let δ be an edge from M to M' .

If M is not on any path from N to V , then there is a rooted path to M that does not cross N , which we will call Δ . Since M' is on a path from N to V , there is a path from M' to V that does not cross N , called Δ' , unless $M' = N$. There will thus be a rooted path to V , consisting of Δ , δ and Δ' , by which V is either free or bound by some other node than N , a contradiction.

As for the other three conditions in the restriction: candidate binder edges have been chosen such that they exclude specific nodes from the scope of their source node, matching the demands of Restriction 3.17. \square

For easier reference the restriction on binding in proof graphs is repeated here:

Restriction 3.17 Let N be a binder node and let Γ be the scope of N . Then the following statements must hold:

- for all nodes M, M' such that there is an edge from M to M' , if $M \notin \Gamma$ but $M' \in \Gamma$, then $M' = N$.
- If N is a $\vee E$ -node then the right minor premise and right closed assumption are outside the left scope, and the left minor premise and left closed assumption are outside the right scope of N .
- If N is a $\vee E$ -node or $\exists E$ -node then the major premise of N must be outside Γ .
- If N is a $\forall I$ -node or $\exists E$ -node then its conclusion must be outside Γ .

The operation that removes backpointers and replaces them with fresh indices is dubbed '*fresh*', and will be presented next.

Glancing forward, when fresh indices are appointed to bisimilar formula graphs, reading back these graphs is anticipated to result in α -equivalent formulas. Yet corresponding subformulas in the premises and conclusion of inferences in a proof are required to be identical, not merely α -equivalent. For most inference schemes in graphs this is solved by sharing the same formula graph. However, for the quantifier schemes this was impossible due to the substitutions involved. Instead of sharing the same formula graph, $\forall I$ -, $\forall E$ -, $\exists I$ - and $\exists E$ -schemes have formula graphs that are bisimilar modulo identification of two nodes.

The solution will be to find the variables that should have the same index with the help of the bisimulation modulo identification relation.

Definition 4.24 In a proof graph \mathcal{G} , let R be the union of the all the smallest bisimulation modulo identification-relations that satisfy Restriction 3.16 (compliance to the schemes in Appendix B). The relation *varlink* is then the relation that holds for two variable nodes V and V' whenever $R(\text{bind}(V), \text{bind}(V'))$ holds.

Definition 4.25 For a proof graph \mathcal{G} the clean pre-proof graph $\text{fresh}(\mathcal{G})$ is identical to \mathcal{G} in all respects except the following:

- The *bind* function in $\text{fresh}(\mathcal{G})$ is empty.
- For all assumption and variable nodes V in \mathcal{G} , if $V \in \text{DOM}(\text{bind})$ in \mathcal{G} then $\text{index}(V) = I$ in $\text{fresh}(\mathcal{G})$ for some $I \notin \text{INDICES}_{\mathcal{G}}$.
- For two nodes $V, V' \in \text{DOM}(\text{bind})$ in \mathcal{G} , $\text{index}(V) = \text{index}(V')$ only if *varlink*(V, V') holds in \mathcal{G} .

Again, we need to prove that binding is preserved, under the *fresh* operation. We will also show that binding remains unique in the resulting graph. That is, however, dependent on certain conditions. In the *fresh* operation some quantifiers will have bound variable nodes that receive the same index. Potentially one of those quantifier nodes may get ‘in between’ a node and its intended binder, which variable node then ends up with a different binder than before the operation.

To show that this scenario cannot occur with proof graphs, we need some more results on the bisimulation modulo identification relation. To get into the right mood, the same proof is first presented for regular bisimulations.

Lemma 4.26 In a proof graph there is no path (of length greater than zero) between two bisimilar nodes.

Proof: let N and M be bisimilar nodes and let Γ be a path from N to M . Then there must be a path Γ' from M to a node M' , consisting of the same sequence of ports as Γ , only traversing different nodes.

Since the nodes on Γ and Γ' are pairwise bisimilar, M and M' are bisimilar. Consequently there must be a path Γ'' from M' to a bisimilar node M'' , and so on indefinitely. The concatenation of all paths Γ, Γ' etc. forms an infinite path, which is impossible in a finite acyclic graph. \square

Lemma 4.27 In a proof graph, when $N \approx_Q^P M$ and there is a non-zero length path from N to M , then there is a path from P to Q or from Q to P .

Proof: let Γ be the path from N to M . If N and M were regularly bisimilar, there would be two infinite paths starting at each of them (and overlapping most of the way). Let $\Delta = \langle \Gamma, \Gamma', \Gamma'', \dots \rangle$ be that hypothetical infinite path starting at N , and let $\Delta' = \langle \Gamma', \Gamma'', \Gamma''', \dots \rangle$ be the same path starting at M .

Every n th pair of nodes N' on Δ and M' on Δ' , $N' \approx_Q^P M'$. Since proof graphs are finite, some N' and M' must be P and Q . Since all nodes on Δ (except those on Γ) are also on Δ' , there is a path from P to Q or vice versa. Of course, if one of P and Q is on Γ , then there is a path from that node to M and from M to the other, Q or P . \square

A remark should be made regarding the previous lemma. To prevent the two paths Δ and Δ' from being infinite, they must end at P and Q , at the same point relative to each other. There is no other way to end them, since each edge is unique at each node. If, at some node on Δ , a different path would branch off towards P , then from that node Δ still continues, and what would be required of Δ' is that it branches off at exactly the same point, *and* continues in its original direction.

The significance of the result becomes clear when we look at the inference schemes that require the presence of some bisimulation modulo identification. Each involves at least one variable node that is bound (just) outside the range of the required relation. Since variable nodes have no edges, there can be no path from that node to the one it is identified with. And there can also be no path in the other direction, since that would create a path to the variable node that goes around its binder.

Theorem 4.28 Let N and M be nodes in a proof graph \mathcal{G} . Then N binds M in $fresh(\mathcal{G})$ if and only if N binds M in \mathcal{G} .

Proof: by example. Suppose N is a $\vee E$ -node and M is the left closed assumption of N . Since the new indices are fresh, the node M has a unique index, say, u . Thus N is the only node that could bind M , since it is the only node whose closed assumption has index u .

That M may not be free means that every rooted path in $fresh(\mathcal{G})$ to M has to cross the edge $\langle N, \text{left minor premise} \rangle$ —ignoring the closed assumption-edge itself for simplicity. The left scope of N consists of all nodes on any path from N to M , including M itself. Any path from a root to M shares its final part with some path from N to M , even if it is only the node N itself. The first

requirement of Restriction 3.17 is that a path may only enter the scope of N at N itself; in other words, any path from a root node to M must cross N . The second and third requirement of Restriction 3.17 ensure that the correct port is used on that path: the major premise, right minor premise and right closed assumption must be outside the scope of N . Any path crossing one of those will never reach M , since acyclicity prevents N from reoccurring on that path.

For the other assumption-closing nodes and for variable binding the above argument can be reproduced straightforwardly, except that some variable nodes receive the same index. This is exactly the case for two variable nodes V and V' when $varlink(V, V')$ holds, which, in turn, happens when the quantifiers binding V and V' are required to be bisimilar or bisimilar modulo identification of some nodes P and Q . Yet because of Lemma 4.27 such quantifiers cannot be on the same path, and thus cannot interfere with each other. \square

4.4 The *read* operation

After the backpointers in a proof graph have been replaced by fresh indices, the result is a clean pre-proof graph. The *read* operation will read the information contained in the graph; to help with the correctness proofs it will also be able to interpret empty nodes.

Definition 4.29 The *read* : NODES \rightarrow deductions function is defined as follows: for all nodes N with $label(N) \in \text{R-LABELS}$, if $label(N) = \text{ASS}$ then

$$read(N) = read(C)^{index(N)},$$

if $label(N) = \varepsilon_0$ then

$$read(N) = repr(N),$$

and otherwise

$$read(N) = \frac{read(N_{m+1}) \dots read(N_n)}{read(C)} (label(N), index(N_1) \dots index(N_m)),$$

where: $C = succ(N, \text{conclusion})$ and $N_i = succ(N, p_i)$ for assumption-type ports $p_1 \dots p_m \in ports(N)$ and premise-type ports $p_{m+1} \dots p_n \in ports(N)$, in the same order as in the list on page 106.

For nodes N with $label(N) \in \text{F-LABELS}$, the following table gives *read*(N) for

each different label:

$label(N)$	$read(N)$
\wedge	$(read(N_1)) \wedge (read(N_2))$
\vee	$(read(N_1)) \vee (read(N_2))$
\rightarrow	$(read(N_1)) \rightarrow (read(N_2))$
\perp	\perp
\forall	$\forall index(N_1)(read(N_2))$
\exists	$\exists index(N_1)(read(N_2))$
VAR	$index(N)$
P_n	$index(N)(read(N_1), \dots, read(N_n))$
ε_1	$repr(N)$

where $N_i = succ(N, p_i)$ for ports $p_1 \dots p_n \in ports(N)$, in the order of the list on page 106.

For an entire graph \mathcal{G} , we define $read(\mathcal{G})$ to be the set

$$\{read(N) \mid N \text{ is a root node of } \mathcal{G}\}$$

When a node N in \mathcal{G} is shared, we have multiple subdeductions $read(N)$ in $read(\mathcal{G})$. For instance, suppose N is the target of $\langle M, \text{major premise}, N \rangle$ and $\langle M', \text{right premise}, N \rangle$, then both the major premise of $read(M)$ and the right premise of $read(M')$ will be the deduction $read(N)$. We will refer to each as an *occurrence* of $read(N)$.

In general, we want two bisimilar graphs to result in the same proof when they are read, but the replacement of backpointers by fresh indices prevents this. Lemma 4.30 shows that graphs that are bisimilar after the assignment of fresh indices, do produce one and the same proof.

Lemma 4.30 For two clean pre-proof graphs \mathcal{G} and \mathcal{H} , if two nodes $N \in \mathcal{G}$ and $M \in \mathcal{H}$ are bisimilar ($N \equiv M$), then $read(N) = read(M)$.

Proof: (we show only the case that N and M are rule nodes other than empty or assumption nodes, other cases are similar) let R be the bisimulation such that $R(N, M)$. Then, firstly, $label(N) = label(M)$. Secondly, for all ports $p_i \in ports(N)$, if N_i and M_i are the nodes such that $succ(N, p_i) = N_i$ and $succ(M, p_i) = M_i$, then $R(N_i, M_i)$ holds (for $1 \leq i \leq n$ with n the number of ports in $ports(N)$). Also, if N_i and M_i are variable or assumption nodes, we have ($index(N_i) = index(M_i)$). Since N and M are rule nodes, $read(N)$ and $read(M)$ are respectively:

$$\frac{read(N_{m+1}) \dots read(N_{n-1})}{read(N_n)} (label(N), read(bind(N_1)) \dots read(bind(N_m)))$$

$$\frac{read(M_{m+1}) \dots read(M_{n-1})}{read(M_n)} (label(M), read(bind(M_1)) \dots read(bind(M_m)))$$

If for all $1 \leq i \leq n$ $read(N_i) = read(M_i)$, and $read(bind(N_i)) = read(bind(M_i))$ whenever N_i and M_i are assumption or variable nodes, $read(N) = read(M)$. \square

The ports used in the proof graph framework have been named after the connection they represent. For instance, the major premise-port of a $\exists E$ -node links that node to the inference that is its major premise—or at least, whose conclusion is that major premise. These names were chosen to correspond not just for convenient interpretation (port names are omitted from most pictures) but to allow the paths in a graph to be traced through a deduction as well.

Some information that was implicit in deductions has been made explicit in graphs, such as proper variables and proper terms. Also, in natural deduction the assumption class closed by an inference is indicated only by its marker, where in graphs it is indicated by an assumption node, which has both the marker and the formula. Apart from these differences, paths in graphs can be directly applied to proofs.

Lemma 4.31 Let Γ be a path in a graph \mathcal{G} . Then the target of Γ when it is traced through $read(\mathcal{G})$ is an occurrence of the deduction $read(target(\Gamma))$.

Proof: by induction. We take an implication introduction node N as example. The properties of N in \mathcal{G} are as follows:

$$\begin{aligned} label(N) &= \rightarrow I \\ succ(N, conclusion) &= N_1 \\ succ(N, premise) &= M \\ succ(N, closed\ assumption) &= A \end{aligned}$$

The deduction read from N is then:

$$read(N) = \frac{read(M)}{read(N_1)}_{(\rightarrow I, index(A))}$$

Let Γ be a path to N in \mathcal{G} . Then, since Γ as a node in the unfolding of \mathcal{G} is bisimilar to N , $read(\Gamma) = read(N)$. Next, suppose that the subdeduction found at the end of Γ in $read(\mathcal{G})$ is $read(N)$ (this is the induction hypothesis). Let Γ' be the path Γ with the added edge $\langle N, conclusion \rangle$, and let Γ'' be Γ with $\langle N, premise \rangle$ added. Then $target(\Gamma') = N_1$ while the conclusion of $read(N)$ is $read(N_1)$, and $target(\Gamma'') = M$ while the subdeduction at the premise of N is $read(M)$. \square

Theorem 4.32 The unfolding of a clean pre-proof graph \mathcal{G} and the deduction that can be read from it, $read(\mathcal{G})$, can be mapped onto each other in a one-to-one fashion (i.e. they are isomorphic).

Proof: we map assumption nodes onto assumptions, other rule nodes onto inferences, and formula nodes onto connectives, quantifiers, propositions and variable occurrences. For a node Γ in the unfolding of \mathcal{G} with label L , if Γ is a rule node

the last inference in $read(\Gamma)$ is of the type L ; if Γ is a formula node then the primary connective in $read(\Gamma)$ is of type L . If Γ is a variable node or assumption node the variable letter or assumption marker of $read(\Gamma)$ corresponds to the index of Γ .

This, together with the previous lemma, ensures that the node Γ in the unfolding of \mathcal{G} can be mapped onto the inference or connective found at Γ in $read(\mathcal{G})$. \square

The last, important result of this section will be that the bisimulation modulo identification relation can represent a substitution.

Theorem 4.33 In a proof graph \mathcal{G} , let N and M be two formula nodes that are bisimilar modulo identification of nodes X and Y , and let no path exist from M to X . Then for nodes N , M , X and Y in $fresh(\mathcal{G})$ the following statement will hold:

$$read(M) = read(N)[read(Y)/read(X)]$$

Proof: only the cases that differ from the theorem above will be discussed. Let R be the minimal witness for $N \approx_X^Y M$. For a path Γ from M to M' and from N to N' , either $M' = Y$ and $N' = X$, or M' and N' are variable nodes and R holds between their binders—other cases are similar to the previous theorem.

In the first case, there will be an occurrence of $read(Y)$ in $read(M)$ at the same location where there is an occurrence of $read(X)$ in $read(N)$. In the second case, the relation *varlink* will hold between the variable nodes M' and N' , so they will be assigned the same index in $fresh(\mathcal{G})$. \square

5 Graph Rewriting

5.1 Rewrite steps

A rewrite step for proof graphs consists of several smaller operations. Some rewrite steps require duplication of parts of the graph, others require nodes to be merged, and after most steps some obsolete nodes have to be removed. In this section we will explain the schemes found in Appendix C and formalize the supporting operations.

The additional operations are *copy* and *clean*, and the rewrite operations themselves are divided into contractions, permutations and simplifications—although ‘contraction’ will informally be used for all three.

Definition 5.1 A *rewrite step* is a sequence of operations conforming to one of the following three templates:

$$\begin{aligned}
 1) \quad \mathcal{G} &\xRightarrow{\text{copy}_N^M} \mathcal{G}' \xRightarrow{\text{contract}_N^M} \mathcal{G}'' \xRightarrow{\text{clean}_R^S} \mathcal{H} \\
 2) \quad \mathcal{G} &\xRightarrow{\text{copy}_N^M} \mathcal{G}' \xRightarrow{\text{permute}_N^M} \mathcal{G}'' \xRightarrow{\text{clean}_R^S} \mathcal{H} \\
 3) \quad \mathcal{G} &\xRightarrow{\text{simplify}_N} \mathcal{G}' \xRightarrow{\text{clean}_R^S} \mathcal{H}
 \end{aligned}$$

where \mathcal{G} is a rooted proof graph, and *contract*, *permute* and *simplify* are the steps described by the schemes in Appendix C, in which N and M are the elimination and introduction nodes of the contraction. In each line R is the root node of \mathcal{G} , and S is the target of the dotted arrow labeled (1) in the right hand side of the contraction scheme.

The two additional operations are defined below.

Definition 5.2 The *copy* operation is defined as follows:

$\text{copy}_N^M(\mathcal{G})$ is a graph \mathcal{H} such that

- $\text{NODES}_{\mathcal{G}} \subseteq \text{NODES}_{\mathcal{H}}$
- ORIGINALS is the smallest subset of $\text{NODES}_{\mathcal{G}}$ containing N , closed under ‘scope’, i.e. if $K \in \text{ORIGINALS}$ and L is in the scope of K , then $L \in \text{ORIGINALS}$
- DUPLICATES is the set $\text{NODES}_{\mathcal{H}} - \text{NODES}_{\mathcal{G}}$
- R_{copy} is a bisimulation consisting of a bijection from ORIGINALS to DUPLICATES and the identity relation on $\text{NODES}_{\mathcal{G}} - \text{ORIGINALS}$

- for all nodes $N' \in \mathcal{G}$ the label, successor, index and bind functions remain the same in \mathcal{H} , except when $\text{succ}(N', p) = M$ and $N' \neq N$; then $\text{succ}(N', p) = M'$ in \mathcal{H} for the node M' for which $R_{\text{copy}}(M, M')$ holds

Definition 5.3 The *clean* operation is defined as follows:

$$\text{clean}_R^S(\mathcal{G}) = \begin{cases} \mathcal{G}_R & \text{if } R \in \mathcal{G}, \text{ and} \\ \mathcal{G}_S & \text{otherwise} \end{cases}$$

The *clean* operation simply confines the resulting graph to all the nodes reachable from the original root node. The node S featuring in the definition is a provision made in case the original root node was the elimination node of the contraction, and has been removed from the graph.

Figure 5.5 shows the scheme for existential quantifier contraction. The only other contraction that requires merging, universal quantifier contraction, was informally discussed in section 2.10, so with this example all cases are covered.

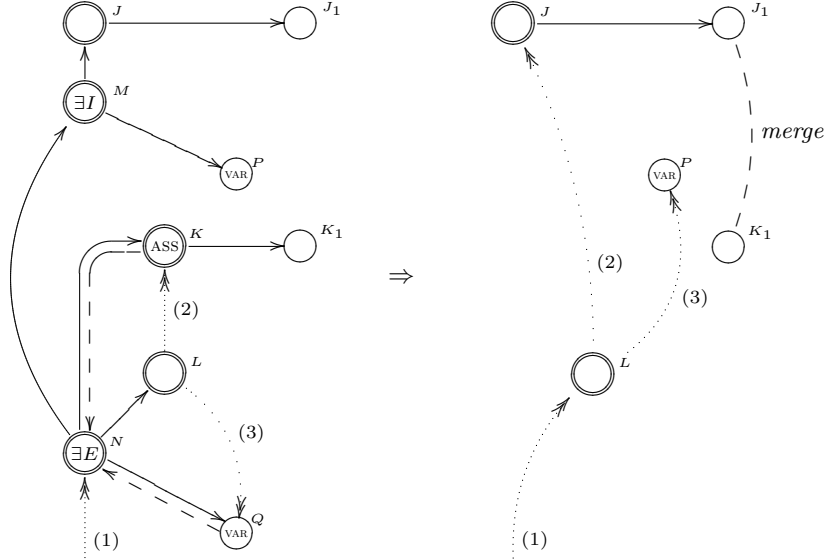


Figure 5.5: \exists -contraction

The left graph in Figure 5.5 will be called \mathcal{G} and the right graph \mathcal{H} . Throughout the schemes in Appendix C the introduction and elimination node of the contraction have been named M and N . The functions of the other nodes are, as usual, given by the direction of the edges upon leaving the node; for example, L is the minor premise of N , because the edge towards it leaves N at the upper right side.

Each dotted arrow represents a set of predecessors. The dotted arrow labeled (1), in all schemes, represents the predecessors of the elimination node of the

contraction—if there are no predecessors to that node, the arrow may be taken to indicate the intended root node of the resulting graph. In \mathcal{H} , (1) points to the node L , which means that all predecessors of N in \mathcal{G} have become predecessors of L in \mathcal{H} :

$$\forall \alpha, p \text{ if } \text{succ}(\alpha, p) = N \text{ in } \mathcal{G} \text{ then } \text{succ}(\alpha, p) = L \text{ in } \mathcal{H}.$$

The node K is the assumption closed by M . All occurrences of that assumption should be substituted by I , the premise of the $\exists I$ -node J :

$$\forall \alpha, p \text{ if } \text{succ}(\alpha, p) = K \text{ in } \mathcal{G} \text{ then } \text{succ}(\alpha, p) = J \text{ in } \mathcal{H}.$$

The nodes Q and P are the proper variable of M , and the proper term of J , respectively. All occurrences of Q in the subproof of L , and those are all occurrences in the graph \mathcal{G} , should be replaced by P :

$$\forall \alpha, p \text{ if } \text{succ}(\alpha, p) = Q \text{ in } \mathcal{G} \text{ then } \text{succ}(\alpha, p) = P \text{ in } \mathcal{H}.$$

The nodes whose predecessors have been removed do not return in the right graph, which means that they do not occur in \mathcal{H} . They may be manually deleted or one can wait for the *clean* step to handle it.

The dashed arrow labeled *merge* indicates that the nodes K_1 and J_1 , which are the conclusions of K and J , should be merged (let \mathcal{G}' be the result of the above three edge redirecting steps):

$$\mathcal{H} = \text{merge}_R(\mathcal{G}') \text{ where } R \text{ is the bisimulation between } J_1 \text{ and } K_1$$

Of course it needs to be shown that there is such a bisimulation:

Lemma 5.4 For three graphs \mathcal{G} , \mathcal{G}' , \mathcal{G}'' and three nodes α , β and γ , if $\mathcal{G} \approx_{\beta}^{\alpha} \mathcal{G}'$, $\mathcal{G}' \approx_{\gamma}^{\beta} \mathcal{G}''$ and no node bisimilar to β occurs in \mathcal{G} and \mathcal{G}'' , then $\mathcal{G} \approx_{\gamma}^{\alpha} \mathcal{G}''$.

Proof: let R be the bisimulation modulo identification from \mathcal{G} to \mathcal{G}' and let S be the one from \mathcal{G}' to \mathcal{G}'' . Let N , N' and N'' be nodes in \mathcal{G} , \mathcal{G}' and \mathcal{G}'' such that $R(N, N')$ and $S(N', N'')$.

For R two cases may be distinguished: either N and N' have the same labels and indices and R holds between their successors and binders, or $N = \alpha$ and $N' = \beta$. For S a similar distinction can be made: N' and N'' may have matching labels and indices while S holds for their successors and binders, or $N' = \beta$ and $N'' = \gamma$. Since N and N'' may not be bisimilar to β , the first case for R and the second case for S are mutually exclusive, and vice versa.

Focusing on N and N'' , the two options left are that $N = \alpha$ and $N'' = \gamma$, or that N and N'' have matching labels and indices and that $R \circ S$ holds between their successors and binders. Thus, $R \circ S$ is a bisimulation modulo identification of α and γ , and $\mathcal{G} \approx_{\gamma}^{\alpha} \mathcal{G}''$ holds. \square

Perhaps unsurprisingly, the previous lemma covers precisely the cases of \forall -contraction and \exists -contraction. Figure 5.6 shows the formula nodes of the \exists -contraction in Figure 5.5. The case for \forall -contraction is similar (see also Figure 2.43).

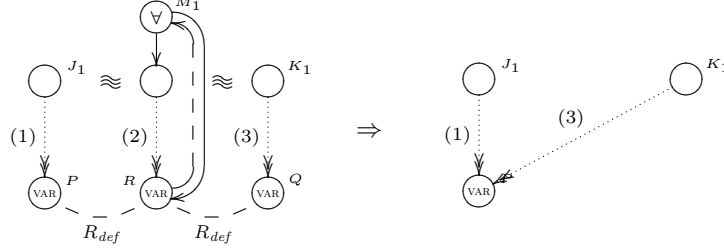


Figure 5.6: The conclusions of a \forall -contraction

The subgraph of J_1 , the subgraph of the subformula of M_1 and the subgraph of K_1 in Figure 5.6 correspond to the graphs \mathcal{G} , \mathcal{G}' and \mathcal{G}'' in the previous lemma; the variable nodes P , R and Q correspond to α , β and γ . The binding of R by M_1 ensures that the subgraphs of J_1 and K_1 cannot contain nodes bisimilar to R : R itself is not reachable from outside the scope of M_1 , and since J_1 and K_1 are the conclusion nodes of some of the rule nodes in the contraction, any quantifier binding P and Q as a predecessor of J_1 or K_1 would also violate scope regulations. In this particular case, from Lemma 5.4 follows $J_1 \approx_Q^P K_1$.

Figure 5.6 also shows the result of the contraction. Most importantly, all edges to Q have been redirected towards P . Since in the left graph J_1 and K_1 are bisimilar modulo identification of P and Q , and in the right graph all occurrences of Q have become occurrences of P , J_1 and K_1 are bisimilar.

5.2 Confluence

In this section we will explore how rewriting in graphs relates to rewriting in deductions. With the help of Figure 5.7

Lemma 5.5 If the root node N of a proof graph \mathcal{G} forms a contraction, permutation or simplification, together with a node M , then the last inferences in $\text{fresh} \circ \text{read}(\mathcal{G})$ form a contraction, permutation or simplification as well. If \mathcal{H} is the result of contracting N and M , then $\text{fresh} \circ \text{read}(\mathcal{H})$ is the result of contracting the last inferences in $\text{fresh} \circ \text{read}(\mathcal{G})$.

Proof: let \mathcal{G} and \mathcal{H} be the left and right graph of the universal quantifier contraction scheme shown in Figure 5.7. Reading back \mathcal{G} and \mathcal{H} gives (contrasted

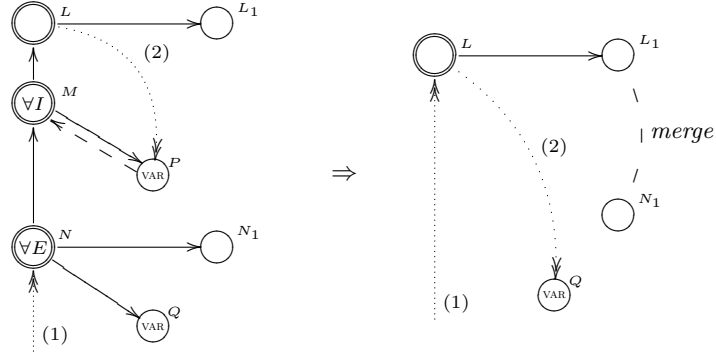


Figure 5.7: Universal quantifier contraction

with the desired result):

$$\begin{array}{c}
 read(\mathcal{G}) = \\
 \frac{\frac{read(L)}{read(M_1)}^{(\forall I)}}{read(N_1)}^{(\forall E)} \\
 \vdots^{(1)} \\
 \frac{A}{\forall x.A[x/a]}^{(\forall I)} \\
 \frac{}{A[t/a]}^{(\forall E)}
 \end{array}
 =
 \begin{array}{c}
 read(\mathcal{H}) = \\
 read(L)[read(Q)/read(P)] \\
 \vdots^{(1)} \\
 A \left. \vphantom{\frac{A}{\forall x.A[x/a]}} \right\} [t/a]
 \end{array}$$

where the node M_1 is the conclusion of M (not present in the illustration). Note that $read(M_1)$ and $read(N_1)$ are formulas, where $read(L)$ is a deduction. Since P is bound in \mathcal{G} it receives a fresh index in $fresh(\mathcal{G})$, which ensures that all occurrences of that index as a variable letter in $read(L)$ have been replaced by occurrences of the index of Q after the contraction. \square

The trouble is, however, that in a graph a contraction may be shared. In that case it does not correspond to a single contraction in a deduction. If, in a graph \mathcal{G} , the node N is the root of a contraction, and no other node than N reads back to the same deduction as N , then contracting N corresponds to contracting all occurrences of the subdeduction $read(N)$ in $read(\mathcal{G})$.

Unfortunately there is no guarantee for different nodes N and M that $read(N)$ is different from $read(M)$. We will need to pinpoint the precise subdeductions that have actually been read from the node N itself. This can be done by tracing all rooted paths to N and mapping them onto the deduction.

Let N be the elimination node of the contraction in Figure 5.8. The step labeled (1) then represents the contraction of all contractions in \mathcal{D} that are at the end

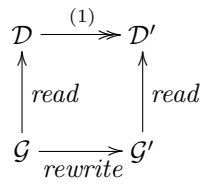


Figure 5.8: Confluence for proof graph rewriting

of a path to N in \mathcal{G} .

By Lemma 4.31 the deduction occurring in \mathcal{D} at the end of the equivalent of a rooted path to N in \mathcal{G} , is $\textit{read}(N)$. The contraction in the step (1) is of the same type as the graph contraction in the step labeled $\textit{rewrite}$ in Figure 5.8. This, and the acyclicity of proof graphs, guarantees that the contractions in \mathcal{D} do not interfere with each other.

6 Correctness and Completeness

In this chapter we will show that proof graphs are a correct and complete representation of natural deduction. It will be demonstrated that each deduction translates to a valid proof graph, and vice versa, that reading a proof graph always results in a correct deduction. Furthermore the reversibility of the translation and reading procedures will be shown.

Regarding the reversibility of the translation procedure, the diagram in Figure 6.1 shows the different steps in the translation of a deduction \mathcal{D} . The first step *trans* produces a clean pre-proof graph \mathcal{G}_1 , merging assumptions and variables in \mathcal{G}_1 results in a clean pre-proof graph \mathcal{G}_2 , adding backpointers to \mathcal{G}_2 produces \mathcal{G}_3 , and finally merging proper terms results in \mathcal{G}_4 . The result of the translation, \mathcal{G}_4 , is a proof graph, indicated by the circle in the diagram.

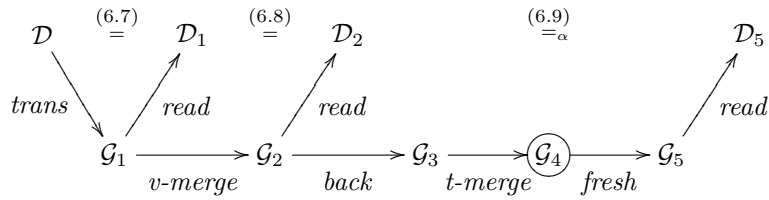


Figure 6.1: Relating the results of reading back the stages of the translation

For the three clean pre-proof graphs in the translation process, the deduction \mathcal{D}_i is the result of reading back the graph \mathcal{G}_i . The other graphs, \mathcal{G}_3 and \mathcal{G}_4 , contain backpointers, which have to be removed before the *read* function can be applied. The importance of the diagram lies in the relations between the deductions. Above each is the number of the theorem that proves it; the three theorems referred to can be found later in this chapter.

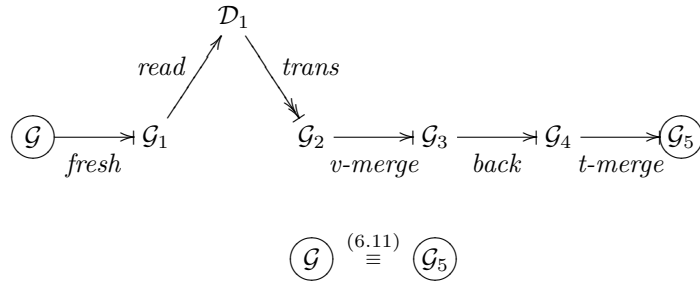


Figure 6.2: Translating an arbitrary graph

Figure 6.2 shows the other side of the story. Starting with an arbitrary proof graph \mathcal{G} , first the backpointers are removed to obtain the clean pre-proof graph \mathcal{G}_1 . From that graph the deduction \mathcal{D}_1 is read. The complete translation proce-

ture, consisting of the consecutive operations *trans*, *v-merge*, *back* and *t-merge* is then applied to \mathcal{D}_1 , resulting in a proof graph \mathcal{G}_5 . If all is well, this proof graph should be bisimilar to the proof graph that we started out with, i.e. $\mathcal{G} \equiv \mathcal{G}_5$.

6.1 Building a proof graph

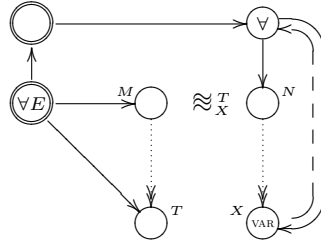
Nearly all the required premises have been proved to show that the translation from proofs to graphs produces a valid proof graph. The missing link is the proof that the translation of a substitution actually results in two graphs that are bisimilar modulo identification.

Lemma 6.1 In the complete translation of a deduction the requirements for bisimulations modulo identification, in the schemes for $\forall I$, $\forall E$, $\exists I$ and $\exists E$, are met.

Proof: by the example below. For the following translation:

$$trans \circ v\text{-merge} \circ back \circ t\text{-merge} \left(\frac{\forall x.A}{A[t/x]} (\forall E) \right)$$

we will show that $N \approx_X^T M$ holds, where M is the conclusion of the $\forall E$ -node, which is the root node, and N is the subformula of the conclusion node (the \forall -node) of the premise of the $\forall E$ -node; the node X is the bound variable of the \forall -node, and the node T the proper term of the $\forall E$ -node; see also the graph below.



In $trans(\mathcal{D})$ (where \mathcal{D} is the deduction above) $M = trans(A[t/x])$ and $N = trans(A)$. Let Γ be a path at the end of which a variable x occurs free in A . Then at the end of Γ in $A[t/x]$ the term t will occur.

By the isomorphism between deductions and unfoldings, any free occurrence of x in A is matched by a free occurrence of a variable node with index x in the unfolding of $trans(A)$. In $trans(\mathcal{D})$ if $source(\Gamma) = N$ then $target(\Gamma)$ is a variable node with index x ; similarly if $source(\Gamma) = M$ then $target(\Gamma)$ is the root node of a graph $trans(t)$. Moreover, the proper term of the $\forall E$ -node will also be a graph $trans(t)$, and the bound variable of the \forall -node will be a variable node with index x .

After the *v-merge* step, all free occurrences of x at N will be merged with its bound variable, to form the node X . Since all variables free in t must remain free in all occurrences of t in $A[t/x]$, those variable occurrences will also be free at M . By Theorem 4.19 free variable nodes in the subgraph of M with the same index will be bound by the same node, if they become bound when $trans(\mathcal{D})$ is a subgraph of some larger graph. This ensures that all occurrences of $trans(t)$ as $target(\Gamma)$ in $trans(A[t/x])$ are bisimilar to the proper term of the $\forall E$ -node, and that they are all merged to form T during the *t-merge* step.

The bisimulation modulo identification can now be constructed between the targets of each path Γ starting at M and Γ starting at N . \square

The theorem below proves that any deduction can be translated to a proof graph.

Theorem 6.2 For any deduction \mathcal{D} $trans \circ v\text{-merge} \circ back \circ t\text{-merge}(\mathcal{D})$ is a proof graph.

Proof: $trans \circ v\text{-merge} \circ back \circ t\text{-merge}(\mathcal{D})$ must be shown to obey the restrictions on proof graphs. Each will be listed and dealt with in turn.

Restriction 3.7 (acyclicity): by Lemma 4.7 $trans(\mathcal{D})$ is acyclic. Both *v-merge* and *t-merge*, since they are based on a bisimulation relation, cannot create new cycles, and in the *back* operation no edges are changed, only indices and backpointers.

Restriction 3.11 (one-to-one correspondence of binder nodes and bound nodes): in $trans(\mathcal{D})$ nodes are uniquely bound, and the *v-merge* procedure unifies all bound nodes of the same binder into one.

Restriction 3.16 (correctness of represented inferences): the similarity between the schemes in Appendix B and those in Appendix D ensures that $trans \circ merge \circ back(\mathcal{D})$ conforms to the former when regular edges are concerned. The correctness of the backpointers in the schemes is implied by the correctness of binding, shown above and below, and the possibility of constructing the required bisimulations modulo identification is guaranteed by Lemma 6.1.

Restriction 3.17 (correctness of binding): by Theorem 4.8, binding is unique in $trans(\mathcal{D})$; by Theorem 4.17 the same holds for $trans \circ v\text{-merge}(\mathcal{D})$. The link between uniqueness of binding and Restriction 3.17 is expressed in Theorem 4.23, which also shows that $trans \circ v\text{-merge} \circ back(\mathcal{D})$ conforms to that restriction. Finally, Theorem 4.20 demonstrates that the *t-merge* step preserves conformation to that restriction as well. \square

6.2 Reversibility of the translation

This section will mostly concentrate on showing the reversibility of the *trans* operation, in the sense that first translating a deduction, and then reading back the resulting graph using the *read* function, gives the original deduction again. The proof consists of several parts; reversibility will be proven separately for the initial steps $trans_0$ and $trans_1$, for formulas, and finally for deductions.

After dealing with the *trans*, the reversibility of the *v-merge*, *back* and *t-merge* operations will be shown, to complete the whole translation sequence.

Lemma 6.3 For any deduction \mathcal{D} $read(trans_0(\mathcal{D})) = \mathcal{D}$, and for any formula A $read(trans_1(A)) = A$.

Proof: the $trans_0$ -step for the deduction \mathcal{D} with conclusion C and the $trans_1$ -step for the formula A are shown below.

$$\begin{array}{ccc}
 \mathcal{D} & & \\
 \vdots & & \\
 trans_0(\mathcal{D}) = & \begin{array}{c} \textcircled{\varepsilon_0}^{N_0} \longrightarrow \textcircled{\varepsilon_1}^{N_1} \\ \vdots \\ C \end{array} & trans_1(A) = \begin{array}{c} \textcircled{\varepsilon_1}^{M_1} \\ \vdots \\ A \end{array}
 \end{array}$$

$$\begin{array}{l}
 read(trans_0(\mathcal{D})) = read(N_0) = repr(N_0) = \mathcal{D} \\
 read(trans_1(A)) = read(M_1) = repr(M_1) = A \quad \square
 \end{array}$$

Lemma 6.4 For an empty formula node N in a pre-proof graph \mathcal{G} , $read(N)$ for $N \in \mathcal{G}$ is the same formula as $read(N)$ for $N \in trans_N(\mathcal{G})$.

Proof: by the example below, the translation scheme for implication.

$$\begin{array}{ccc}
 \begin{array}{c} \textcircled{\varepsilon_1}^N \\ \vdots \\ A \rightarrow B \end{array} & \Rightarrow_{trans_N} & \begin{array}{c} \textcircled{\rightarrow}^N \\ \swarrow \quad \searrow \\ \textcircled{\varepsilon_1}^{N'} \quad \textcircled{\varepsilon_1}^{N''} \\ \vdots \quad \quad \quad \vdots \\ A \quad \quad \quad B \end{array}
 \end{array}$$

$$\begin{array}{l}
 \text{in } \mathcal{G} : \quad read(N) = repr(N) = A \rightarrow B \\
 \text{in } trans_N(\mathcal{G}) : \quad read(N) = read(N') \rightarrow read(N'') \\
 \quad \quad \quad = repr(N') \rightarrow repr(N'') = A \rightarrow B \quad \square
 \end{array}$$

In a translation step $trans_N$ nothing changes for any already existing nodes other than N , except that predecessors may be added. Since the *read* function only looks at successors, the reading back of other nodes than N remains unchanged in a step $trans_N$ as well. Combining the two previous lemmas then gives the following theorem:

Theorem 6.5 For any formula A $trans \circ read(A) = A$. □

To prove the same for deductions, the example of implication introduction is used, repeated in Figure 6.3.

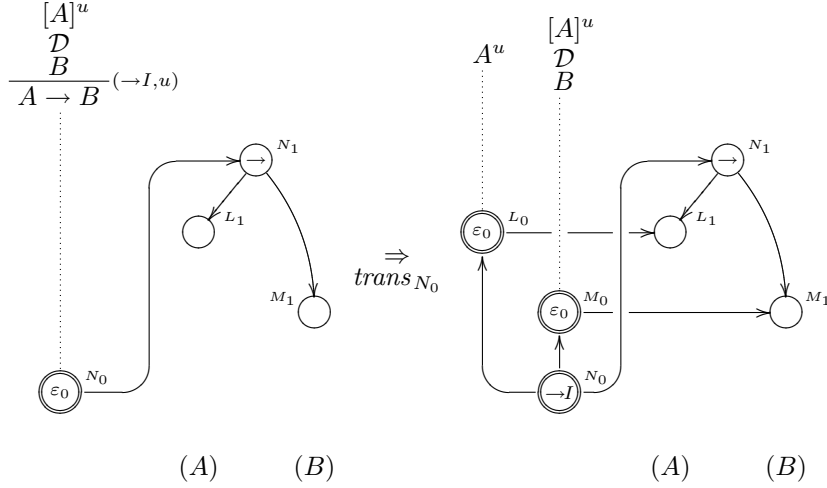


Figure 6.3: The translation scheme for implication introduction

Lemma 6.6 For an inference node N in a pre-proof graph \mathcal{G} , $read(N)$ for $N \in \mathcal{G}$ is the same deduction as $read(N)$ for $N \in trans_N(\mathcal{G})$.

Proof: by the example in Figure 6.3. We show that the deduction read from the node N_0 is the same before and after its translation. Below are the deductions as they are read from the scheme, where \mathcal{G} is the graph on the left.

in \mathcal{G} :

$$read(N_0) = \frac{[A]^u}{\frac{D}{B} \quad A \rightarrow B}^{(->I,u)}$$

in $trans_{N_0}(\mathcal{G})$:

$$read(N_0) = \frac{read(M_0)}{read(N_1)}(label(N_0), index(L_0)) = \frac{[A]^u}{\frac{D}{read(M_1)} \quad read(L_1) \rightarrow read(M_1)}^{(->I,u)}$$

It needs to be shown that $read(L_1) = A$ and that $read(M_1) = B$. Earlier, in Lemma 4.4, it was shown that for an empty node that represents a proof with conclusion C , upon completion of the translation the conclusion node is isomorphic to the direct translation of the formula C . In this case, that means that N_1 is isomorphic to $trans(A \rightarrow B)$, and that L_1 and M_1 are isomorphic

to $trans(A)$ and $trans(B)$, respectively. Applying the previous theorem gives $read(L_1) = A$ and $read(M_1) = B$. \square

Straightforward combination of the above lemmas yields the desired result of reversibility of the $trans$ operation.

Theorem 6.7 For any deduction \mathcal{D}

$$trans \circ read(\mathcal{D}) = \mathcal{D}. \quad \square$$

The following theorem will show the reversibility of the v -merge operation, using results obtained earlier.

Theorem 6.8 For a rooted clean pre-proof graph \mathcal{G}

$$v\text{-merge} \circ read(\mathcal{G}) = read(\mathcal{G}).$$

Proof: by lemma 4.17 $v\text{-merge}(\mathcal{G})$ and \mathcal{G} are bisimilar. By lemma 4.30, which states that bisimilar clean pre-proof graphs yield the same deduction when read, $read(v\text{-merge}(\mathcal{G})) = read(\mathcal{G})$. \square

The remaining part of the reversibility proof will be compressed into one theorem. This is because the intermediate stages cannot be read back directly, as they require the replacement of backpointers by fresh variables first.

Theorem 6.9 For a rooted clean pre-proof graph \mathcal{G}

$$read(\mathcal{G}) =_{\alpha} back \circ t\text{-merge} \circ fresh \circ read(\mathcal{G}).$$

Proof: regular edges and binding are treated separately. Regular edges and paths, as well as the properties of nodes, remain untouched in the $back$ and $fresh$ operations, and thus remain unchanged between (\mathcal{G}) and $back(\mathcal{G})$, and between $back \circ t\text{-merge}(\mathcal{G})$ and $back \circ t\text{-merge} \circ fresh(\mathcal{G})$. Bisimilarity ensures that edges and paths are preserved between $back(\mathcal{G})$ and $back \circ t\text{-merge}(\mathcal{G})$.

By the isomorphism between the unfolding of a graph and the deduction read from it, expressed by Theorem 4.32, regular edges are also preserved between \mathcal{G} and $read(\mathcal{G})$, and between $back \circ t\text{-merge} \circ fresh(\mathcal{G})$ and $back \circ t\text{-merge} \circ fresh \circ read(\mathcal{G})$. This completes the chain and shows that rooted paths are preserved between $read(\mathcal{G})$ and $back \circ t\text{-merge} \circ fresh \circ read(\mathcal{G})$, and that the same path traced through both graphs ends at matching inferences or connectives.

The preservation of binding was also shown for each operation. Theorem 4.32 showed it for the $read$ function, Theorem 4.22 for the $back$ operation, Theorem 4.20 shows it for the $t\text{-merge}$ operation, and Theorem 4.28 for the $fresh$ operation. It should be emphasized that the property preserved in all these theorems is of the form ‘the n th node on a path Γ binds $target(\Gamma)$ ’. \square

6.3 Reading back a proof graph

The core of the reversibility proof for the reading of a deduction from a graph will be the theorem below. It shows that bisimilar proof read back to α -equivalent deductions.

Theorem 6.10 Two proof graphs \mathcal{G} and \mathcal{H} are bisimilar if and only if $fresh \circ read(\mathcal{G}) =_{\alpha} fresh \circ read(\mathcal{H})$ (where α -equivalence may include renaming of proper variables).

Proof: first let \mathcal{G} and \mathcal{H} be bisimilar. Then for any rooted path Γ the label of $target(\Gamma)$ will be the same in \mathcal{G} and \mathcal{H} ; consequently, in $fresh \circ read(\mathcal{G})$ and $fresh \circ read(\mathcal{H})$ all inferences, connectives, free variables and open assumption markers will correspond.

For a rooted path Γ , if $target(\Gamma)$ is bound then by Restriction 3.17 it is bound by some node on Γ , say the n th node. For the same path Γ in \mathcal{H} , the exact same is true.

By Theorem 4.28 the n th node on Γ (say N) binds $target(\Gamma)$ in $fresh(\mathcal{G})$. By Lemma 4.26 N cannot occur on Γ again. If any other binder node M in $fresh(\mathcal{G})$ has a closed assumption, bound variable or proper variable with the same index, then that node is bisimilar modulo some identification to N . By Lemma 4.27 M is then not on Γ . Therefore the n th edge γ is also the last candidate binder on Γ for $target(\Gamma)$ in both $fresh(\mathcal{G})$ and $fresh(\mathcal{H})$.

By Theorem 3.33 the presence of a candidate binder edge on a path is preserved between bisimilar clean pre-proof graphs. Thus the n th edge is the last candidate binder on Γ for $target(\Gamma)$ in the unfoldings of $fresh(\mathcal{G})$ and $fresh(\mathcal{H})$ as well. Finally the correspondence between unfoldings and readings of a graph, expressed by Theorem 4.32, implies that again the n th inference or connective on Γ binds the assumption or variable occurrence at the end of Γ in $fresh \circ read(\mathcal{G})$ and $fresh \circ read(\mathcal{H})$.

For the other direction, suppose \mathcal{G} and \mathcal{H} are not bisimilar. Then for some path Γ $target(\Gamma)$ in \mathcal{G} and $target(\Gamma)$ in \mathcal{H} differ in their label or index, or have non-bisimilar binders. Should the difference lie with the label or index, then $fresh \circ read(\mathcal{G})$ and $fresh \circ read(\mathcal{H})$ will surely not be α -equivalent.

Although not bisimilar, the binders of $target(\Gamma)$ in \mathcal{G} in \mathcal{H} must still lie on Γ . Then in \mathcal{G} $target(\Gamma)$ is bound by the n th node, and in \mathcal{H} $target(\Gamma)$ is bound by the m th node on Γ , where $n \neq m$. By the same argument as above, then in $fresh \circ read(\mathcal{G})$ $target(\Gamma)$ is bound by the n th inference or connective on Γ , while in $fresh \circ read(\mathcal{H})$ $target(\Gamma)$ is bound by the m th inference or connective. Again, $fresh \circ read(\mathcal{G})$ and $fresh \circ read(\mathcal{H})$ are not α -equivalent. \square

For the next theorem, the above theorem and the results on the translation procedure are combined to show the reversibility of the *read* function.

Theorem 6.11 For any proof graph \mathcal{G} ,

$$\mathcal{G} \equiv \text{fresh} \circ \text{read} \circ \text{trans} \circ v\text{-merge} \circ \text{back} \circ t\text{-merge}(\mathcal{G}).$$

Proof: by Theorems 6.7, 6.8 and 6.9 we have (abbreviating the translation sequence of *trans*, *v-merge*, *back* and *t-merge* to *translation*):

$$\text{fresh} \circ \text{read}(\mathcal{G}) =_{\alpha} \text{fresh} \circ \text{read} \circ \text{translation} \circ \text{fresh} \circ \text{read}(\mathcal{G}),$$

after which Theorem 6.10 can be applied to produce the desired result. \square

Finally, it will be demonstrated that any proof graph reads back to a (nearly) correct deduction.

Theorem 6.12 For an arbitrary proof graph \mathcal{G} $\text{fresh} \circ \text{read}(\mathcal{G})$ is a valid deduction, with the exception that open assumptions may contain proper variables.

Proof: comparison of the schemes in *Appendix A* with the definition of the *read* function will reveal that the individual inferences in $\text{fresh} \circ \text{read}(\mathcal{G})$ are correct; for the inferences involving substitution, Theorem 4.33 demonstrates that a substitution is correctly represented by a bisimulation modulo identification.

It then remains to show that the restrictions on $\forall I$ - and $\exists E$ -applications are met. Taking the mentioned exception into account, these read that the proper variable of the inference may not occur free in an assumption that is open at (the minor premise of) the inference itself, but closed in $\text{fresh} \circ \text{read}(\mathcal{G})$.

Let the variable a be the proper variable of a $\forall I$ -application, and let a be free in an assumption A^u that is closed by an $\rightarrow I$ -application in $\text{fresh} \circ \text{read}(\mathcal{G})$. All assumption closure, as well as variable binding by $\forall I$ - and $\exists E$ -nodes, is expressed by backpointers in proof graphs, and is preserved under the *fresh* step (see Theorem 4.28). The variable letter a and the assumption marker u are thus fresh in $\text{fresh}(\mathcal{G})$, which implies that each is attributed to only one node.

Let V be the variable node with index a and let V' be the assumption node with index u . All occurrences of the variable a and the assumption A^u have been read from the nodes V and V' respectively. Since a is free in A^u , there is a path Δ from V' to V that is free of candidate binders for V . Let N' be the $\rightarrow I$ -node that closes V' and let N be the $\forall I$ -node that binds V .

By (the interpretation of) Restriction 3.17 all rooted paths in \mathcal{G} to V should cross N . Since N' closes V' there is an edge from N' to V' using the closed assumption port. The path Δ connects V' to V , so there must be a path from N to N' , to prevent a rooted path to N' from connecting to V , through the closed assumption port of N' and the path Δ .

In the deduction read from $fresh(\mathcal{G})$, then, any occurrence of an inference closing assumption class A^u is in the subdeduction of the $\forall I$ -application that binds a . Consequently, A^u cannot be open in the subdeduction of that $\forall I$ -application. \square

7 Conclusions

At the outset the goal was to develop a system of natural deduction, using graphs, that is more elegant than the original and treats variables a little better. As was already indicated, the attributed degree of success can be a matter of taste, but overall it must be concluded that a graph system for natural deduction was indeed implemented.

To help with the further assessment a summary of the distinctive properties of proof graphs will be given, and, where relevant, contrasted with those of natural deduction.

One of the primary goals, developed throughout the second chapter, was to have all variable occurrences that can philosophically be attributed to the same variable, represented by a single node. A convenient consequence of this choice was that a fairly simple representation of substitution could be implemented, in the form of the bisimulation modulo identification. Extending the same principle to assumptions has also helped in simplifying substitution in the rewrite schemes.

The next achievement is the uniform treatment of assumption closure and two forms of variable binding, by quantifiers and by inferences. Almost all of the definitions and results regarding binding are applicable to all three variants.

Concerning the elegance of proof graphs, a less than optimal result are the complicated restrictions. Although the required form of inferences can be expressed in pictures, the restrictions needed for a correct representation of binding are not that beautiful. On the other hand, these should be compared to the explicit restrictions on quantifier inferences, and to the implicit definition of variable binding in deductions.

The use of backpointers, although already an old idea, has also proven useful. In particular it has made bisimulations modulo identification manageable; no real effort was required to deal with the exceptions present in regular substitution regarding variable binding.

Finally, the rewriting part of the graph system is probably a little more elaborate than that of deductions. There are two smaller operations involved, copying and cleaning up, and the merging utility is used as well, while the pictures used to describe the actual rewrite steps contain a lot of information, which can make them hard to interpret.

Overall, it looks like graphs perform a little better than regular deductions, though it should also be mentioned that implementing graphs is in itself a lot more complicated than implementing trees.

Of greater importance than the technical performance is the fact that proof graphs bring these features to the surface. Although most of the implemented

features are well-documented, in proof graphs they are explicit to the system itself. That, and the detailed treatment of sharing, are the main merits of this thesis.

7.1 Further investigations

A few things need to be said on what the next step should be. The first is that since this thesis presents a completed system of graphs and nothing but a completed system of graphs, there are few angles for further research. However, two of the remaining questions are of great importance, and cannot go unmentioned.

The close reader may have noticed that the rewriting of graphs receives relatively little attention. The reason behind this is that due to size and time constraints a major topic was dropped during the writing of this thesis, being the complexity of rewriting. Although in essence similar to the rewrite rules for proofs, the graph rules behave very differently where duplication is concerned. For deductions, the implication contraction is the main cause of complexity, leading to hyperexponential growth, while permutations are linear. In graphs each individual permutation and implication contraction is equally complex: they are linear when not shared, and at most duplicate the graph when shared. The question is then of course how these rules interact on a larger scale; do proof graphs in the end exhibit hyperexponential growth as well, or is the limited form of sharing sufficient to reduce complexity?

The other topic covers different variants of natural deduction. One question is whether proof graphs can be expanded to cover natural deduction for classical logic. Although the first guess could well be ‘of course’, the addition of another assumption discharging rule might prove very non-trivial. Another question could be how proof graphs correspond to Fitch-style proofs, also known as flag deductions. Instead of building a tree, these deductions use line numbers to refer to premises, in essence forming a graph. The translation from this proof system to proof graphs and back could be very straightforward—or very much not so.

Bibliography

- Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 2001.
- Stefan Blom. *Term Graph Rewriting: Syntax and Semantics*. PhD thesis, Department of Mathematics and Informatics, Vrije Universiteit, Amsterdam, 2001.
- Kit Fine. *Reasoning with Arbitrary Objects*. Blackwell, Oxford, 1985.
- Gerhard Gentzen. Untersuchungen über das logische schliessen I, II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in Gentzen [1969], pages 68–131.
- Gerhard Gentzen. *The Collected Papers of Gerhard Gentzen*. North-Holland Publ. Co., Amsterdam, 1969. English translations of Gentzen’s papers, ed. M. E. Szabo.
- Dag Prawitz. *Natural deduction. A Proof-theoretical Study*. Almqvist and Wiksell, Stockholm, 1965.
- Richard Statman. *Structural Complexity of Proofs*. PhD thesis, Department of Philosophy, Stanford University, Stanford, 1974.
- Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1996.

Appendices

Appendix A: Natural deduction schemes

Inference rules

$$A^u \quad \frac{\vdots}{\perp} (\perp E) \quad \frac{\vdots \quad [A]^u \quad \vdots}{A \rightarrow B} (\rightarrow I, u) \quad \frac{\vdots \quad \vdots}{A \rightarrow B \quad B} (\rightarrow E)$$

$$\frac{\vdots \quad \vdots}{A \wedge B} (\wedge I) \quad \frac{\vdots}{A \wedge B} (\wedge EL) \quad \frac{\vdots}{A \wedge B} (\wedge ER)$$

$$\frac{\vdots}{A \vee B} (\vee IL) \quad \frac{\vdots}{A \vee B} (\vee IR) \quad \frac{\vdots \quad [A]^u \quad [B]^v \quad \vdots \quad \vdots}{A \vee B \quad C \quad C} (\vee E, u, v)$$

$$\frac{\vdots}{\forall x.A} (\forall E) \quad \frac{\vdots}{\forall x.A} (\forall I) \quad \text{Restrictions on } \forall I: a = x \text{ or } a \text{ is not free in } A; a \text{ is not free in open assumptions}$$

$$\frac{\vdots}{\exists x.A} (\exists I) \quad \frac{\vdots \quad [A[a/x]]^u \quad \vdots}{\exists x.A \quad C} (\exists E, u) \quad \text{Restrictions on } \exists E: a = x \text{ or } a \text{ is not free in } A; a \text{ is not free in } C \text{ or in open assumptions except } [A[a/x]]^u$$

Contractions

$\wedge L$ -contraction:

$$\frac{\frac{\begin{array}{c} \vdots^{(1)} \\ A \end{array} \quad \begin{array}{c} \vdots \\ B \end{array}}{A \wedge B} (\wedge I)}{A} (\wedge EL) \Rightarrow \begin{array}{c} \vdots^{(1)} \\ A \end{array}$$

$\wedge R$ -contraction:

$$\frac{\begin{array}{c} \vdots \\ A \end{array} \quad \begin{array}{c} \vdots^{(1)} \\ B \end{array}}{A \wedge B} (\wedge I)}{B} (\wedge ER) \Rightarrow \begin{array}{c} \vdots^{(1)} \\ B \end{array}$$

$\vee L$ -contraction:

$$\frac{\frac{\begin{array}{c} \vdots^{(1)} \\ A \end{array}}{A \vee B} (\vee IL) \quad \frac{\begin{array}{c} [A]^u \\ \vdots^{(2)} \\ C \end{array} \quad \begin{array}{c} [B]^v \\ \vdots \\ C \end{array}}{C} (\vee E, u, v)}{C} \Rightarrow \begin{array}{c} \vdots^{(1)} \\ [A] \\ \vdots^{(2)} \\ C \end{array}$$

$\vee R$ -contraction:

$$\frac{\frac{\begin{array}{c} \vdots^{(1)} \\ B \end{array}}{A \vee B} (\vee IR) \quad \frac{\begin{array}{c} [A]^u \\ \vdots \\ C \end{array} \quad \begin{array}{c} [B]^v \\ \vdots^{(2)} \\ C \end{array}}{C} (\vee E, u, v)}{C} \Rightarrow \begin{array}{c} \vdots^{(1)} \\ [B] \\ \vdots^{(2)} \\ C \end{array}$$

\rightarrow -contraction:

$$\frac{\frac{\begin{array}{c} [A]^u \\ \vdots^{(1)} \\ B \end{array}}{A \rightarrow B} (\rightarrow I, u) \quad \begin{array}{c} \vdots^{(2)} \\ A \end{array}}{B} (\rightarrow E) \Rightarrow \begin{array}{c} \vdots^{(2)} \\ [A] \\ \vdots^{(1)} \\ B \end{array}$$

\forall -contraction:

$$\frac{\frac{\begin{array}{c} \vdots^{(1)} \\ A \end{array}}{\forall x. A[x/a]} (\forall I)}{A[t/a]} (\forall E) \Rightarrow \left. \begin{array}{c} \vdots^{(1)} \\ A \end{array} \right\} [t/a]$$

\exists -contraction:

$$\frac{\frac{\begin{array}{c} \vdots^{(1)} \\ A[t/a] \end{array}}{\exists x. A[x/a]} (\exists I) \quad \begin{array}{c} [A]^u \\ \vdots^{(2)} \\ C \end{array}}{C} (\exists E, u) \Rightarrow \left. \begin{array}{c} \vdots^{(1)} \\ [A] \\ \vdots^{(2)} \\ C \end{array} \right\} [t/a]$$

Permutations

\forall -permutation:

$$\frac{\frac{\frac{\vdots(1)}{A \vee B} \quad \frac{\vdots(2)}{C} \quad \frac{\vdots(3)}{C}}{C} \quad \frac{\vdots(4)}{D}}{D} \quad (\forall E, u, v) \quad \vdots(4)}{D} \quad (?E) \Rightarrow \frac{\frac{\frac{\vdots(1)}{A \vee B} \quad \frac{\vdots(2)}{C} \quad \frac{\vdots(4)}{D}}{D} \quad \frac{\vdots(3)}{C} \quad \frac{\vdots(4)}{D}}{D} \quad (?E)}{D} \quad (\forall E, u, v)$$

\exists -permutation:

$$\frac{\frac{\frac{\vdots(1)}{\exists x.A} \quad \frac{\vdots(2)}{C}}{C} \quad \frac{\vdots(3)}{D}}{D} \quad (\exists E) \quad \frac{\vdots(3)}{D} \quad (?E)}{D} \quad (\exists E) \Rightarrow \frac{\frac{\frac{\vdots(1)}{\exists x.A} \quad \frac{\vdots(2)}{C}}{C} \quad \frac{\vdots(3)}{D}}{D} \quad (\exists E)}{D} \quad (?E)$$

Simplifications

$\forall L$ -simplification:

$$\frac{\frac{\frac{\vdots}{A \vee B} \quad \frac{\vdots(1)}{C} \quad \frac{\vdots}{C}}{C} \quad \frac{\vdots}{D}}{D} \quad (\forall E, u, v) \Rightarrow \frac{\vdots(1)}{C} \quad \text{Where } [A]^u \text{ contains zero instances of } A^u$$

$\forall R$ -simplification:











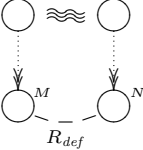
$$\frac{\frac{\frac{\vdots}{A \vee B} \quad \frac{\vdots}{C} \quad \frac{\vdots(1)}{C}}{C} \quad \frac{\vdots}{D}}{D} \quad (\forall E, u, v) \Rightarrow \frac{\vdots(1)}{C} \quad \text{Where } [B]^v \text{ contains zero instances of } B^v$$

\exists -simplification:

$$\frac{\frac{\frac{\vdots}{\exists x.A} \quad \frac{\vdots(1)}{C}}{C} \quad \frac{\vdots}{D}}{D} \quad (\exists E, u) \Rightarrow \frac{\vdots(1)}{C} \quad \text{Where } [A]^u \text{ contains zero instances of } A^u$$

Appendix B: Inference schemes for proof graphs

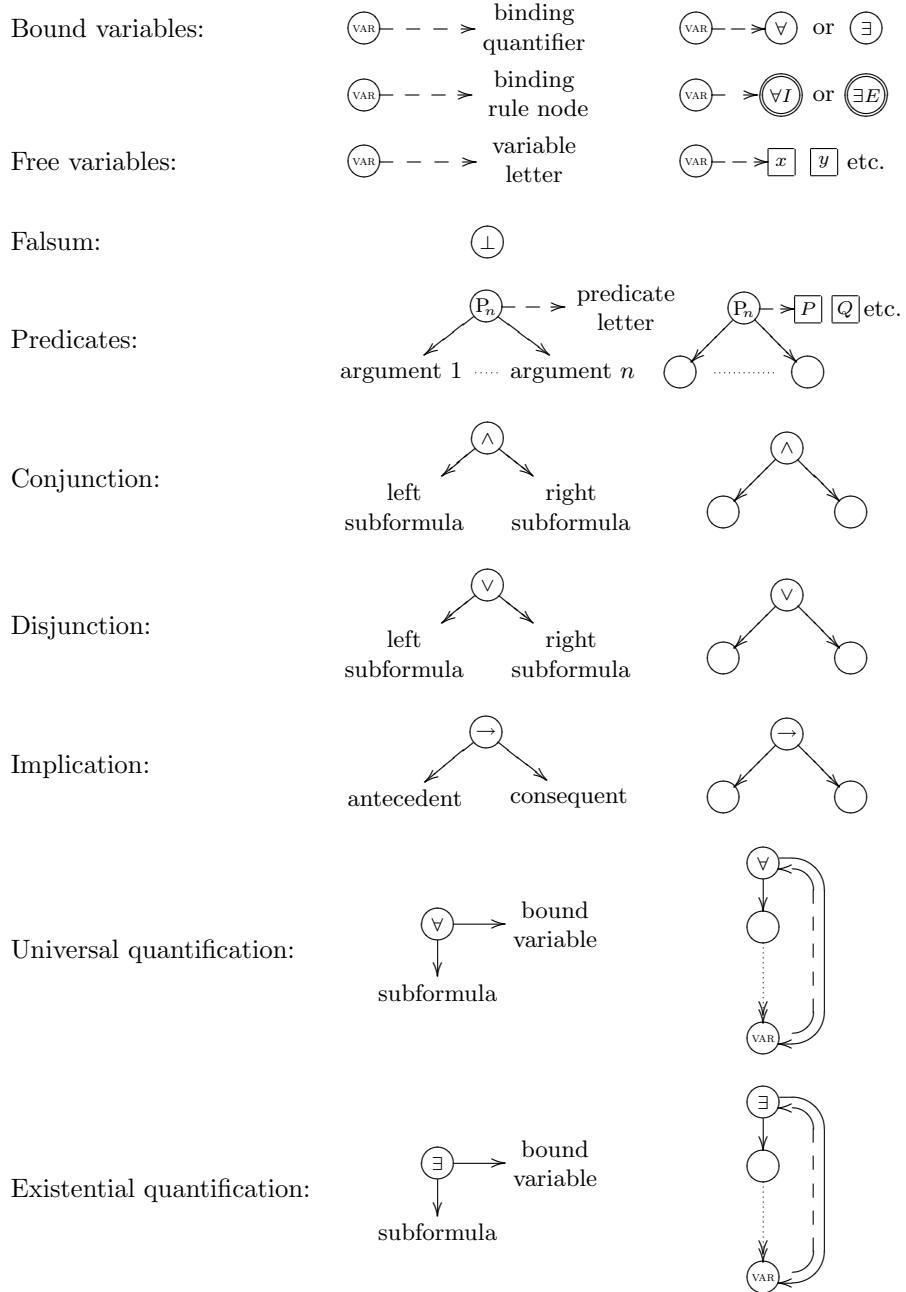
Legend

	rule node
	formula node
	index
	edge
	multiple edges
	indicates scope and predecessors
	no path
	backpointer
	bisimulation
	bisimulation modulo identification
	bisimulation modulo identification of nodes M and N

List of ports

left closed assumption	}	assumption-type ports
closed assumption		
right closed assumption		
major premise	}	premise-type ports
left premise		
premise		
right premise		
left minor premise		
minor premise		
right minor premise		
conclusion		
proper term		
proper variable	}	variable-type ports
bound variable		
left subformula	}	subformula-type ports
subformula		
right subformula		
antecedent		
consequent		
argument 1		
argument 2		
argument 3		
⋮	}	

Formula nodes



Rule nodes

Open assumptions:



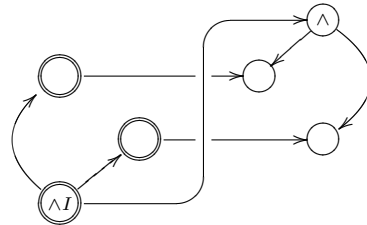
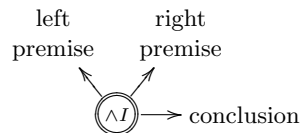
Closed assumptions:



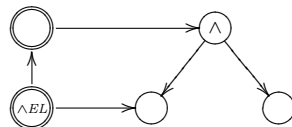
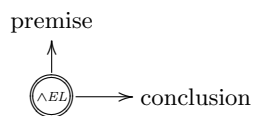
Falsum elimination:



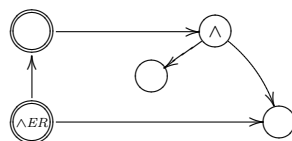
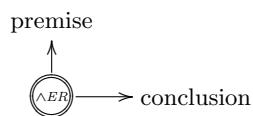
Conjunction introduction:



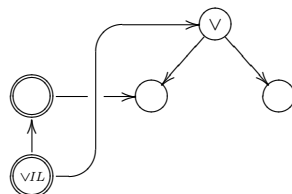
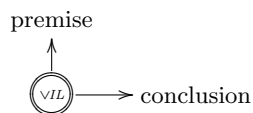
Conjunction elimination (left):



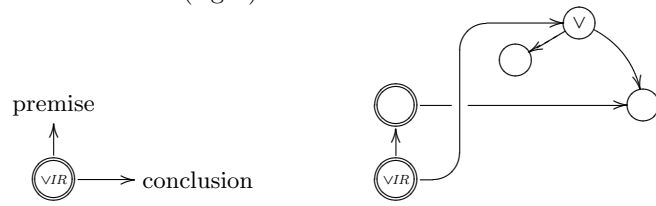
Conjunction elimination (right):



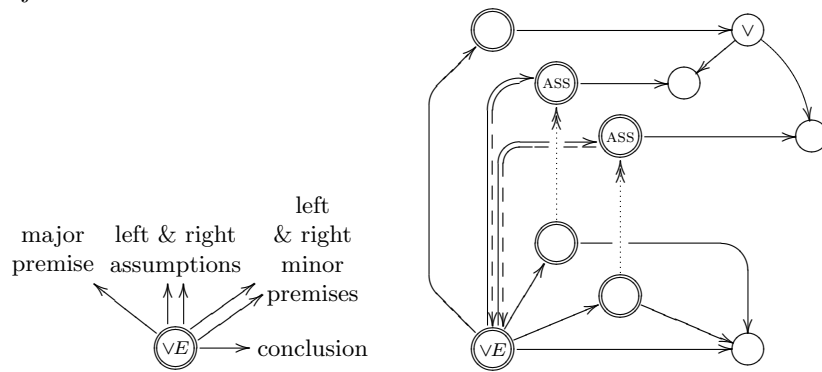
Disjunction introduction (left):



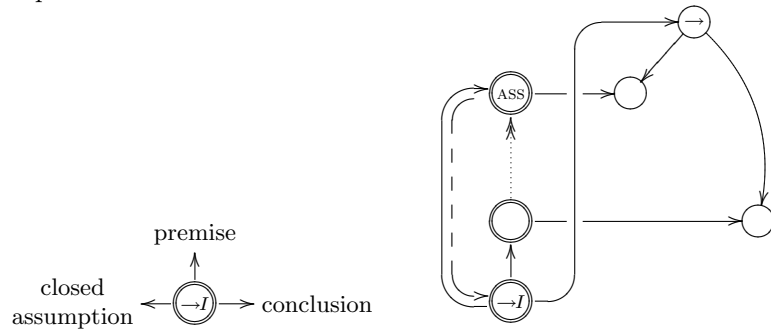
Disjunction introduction (right):



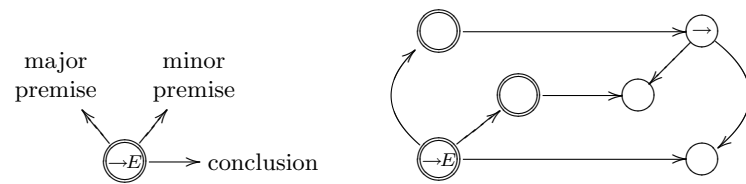
Disjunction elimination:



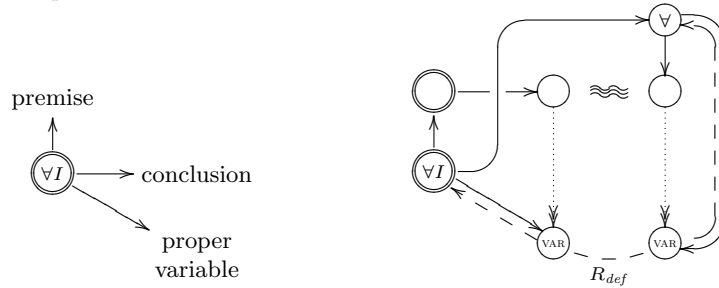
Implication introduction:



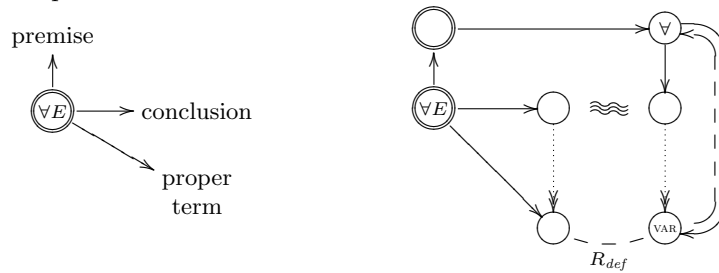
Implication elimination:



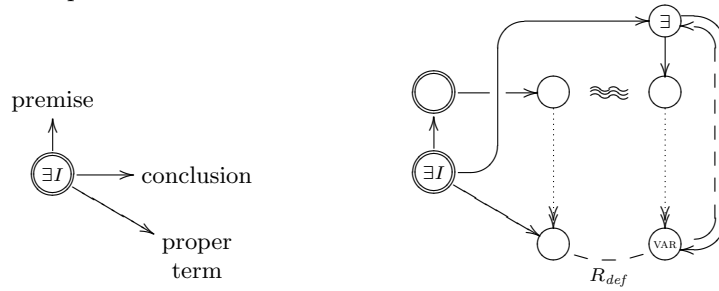
Universal quantifier introduction:



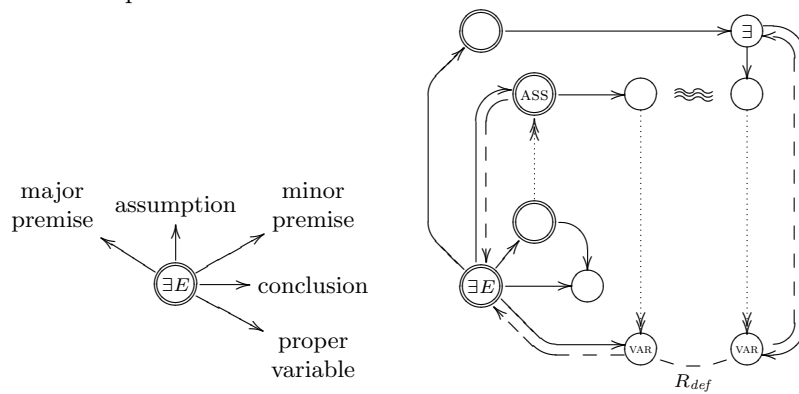
Universal quantifier elimination:



Existential quantifier introduction:



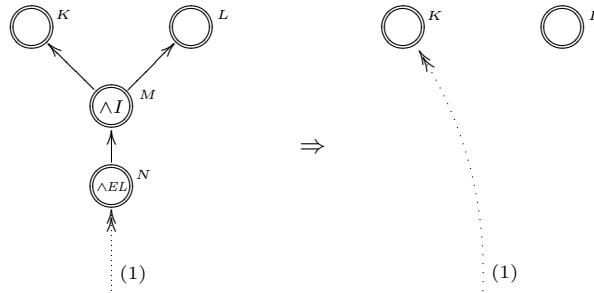
Existential quantifier elimination:



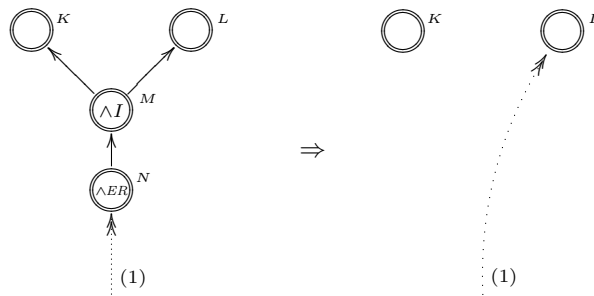
Appendix C: Rewrite schemes for proof graphs

Contractions

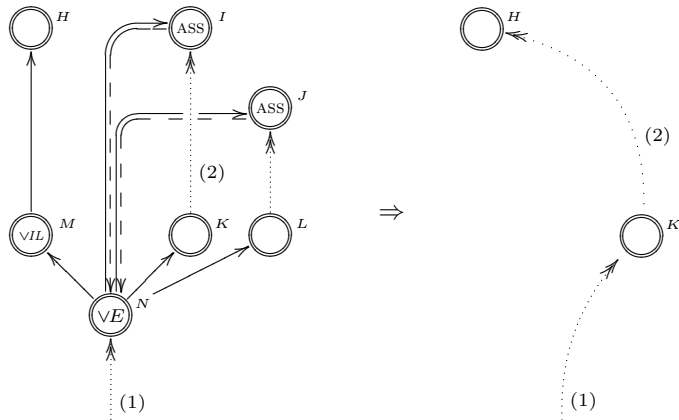
Conjunction contraction (left):



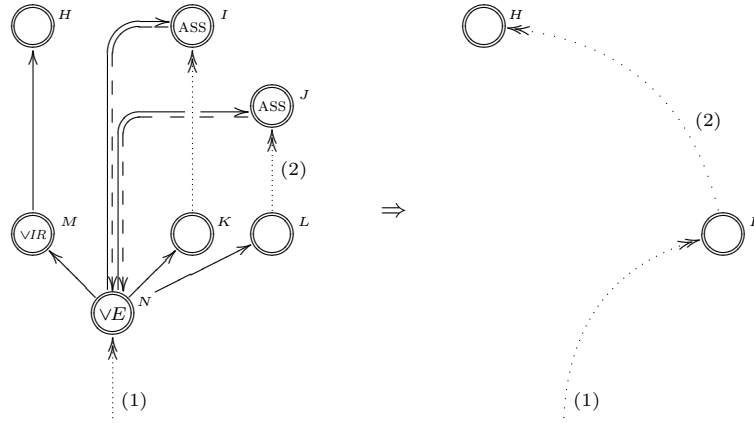
Conjunction contraction (right):



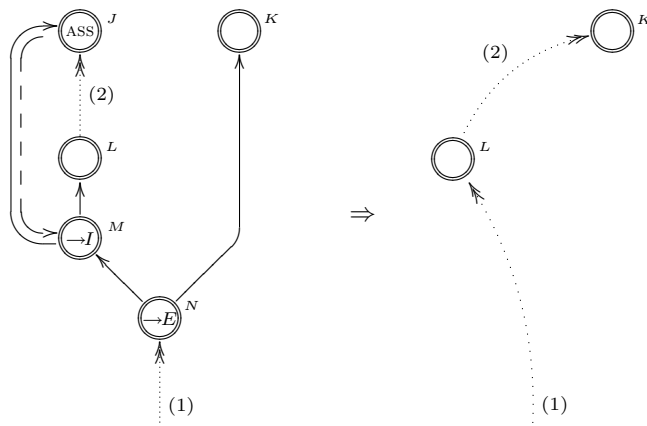
Disjunction contraction (left):



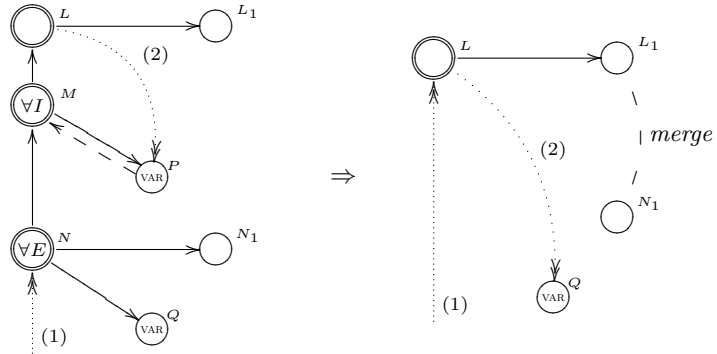
Disjunction contraction (right):



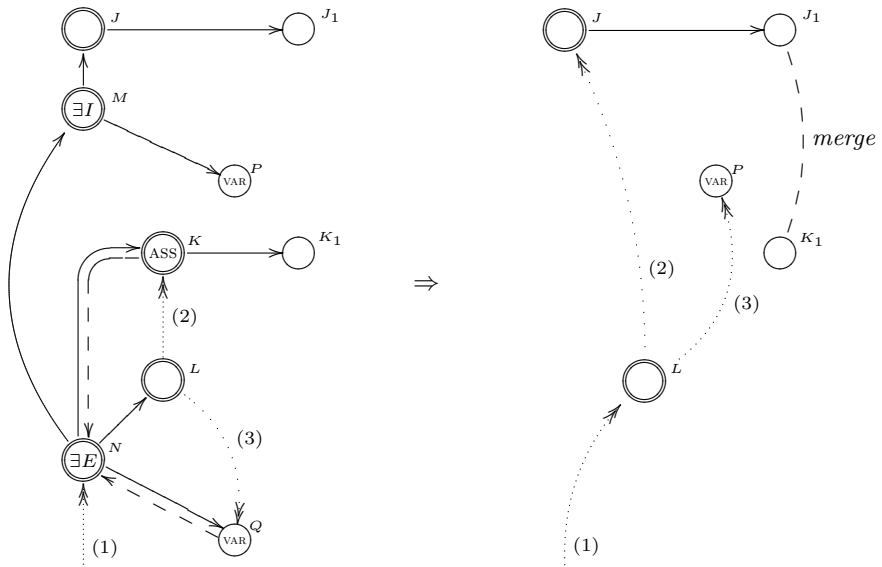
Implication contraction:



Universal quantifier contraction:

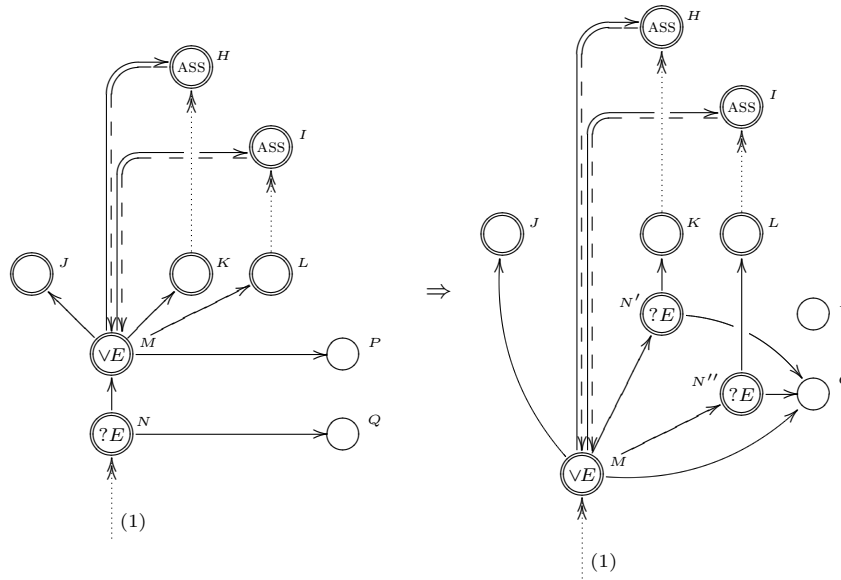


Existential quantifier contraction:

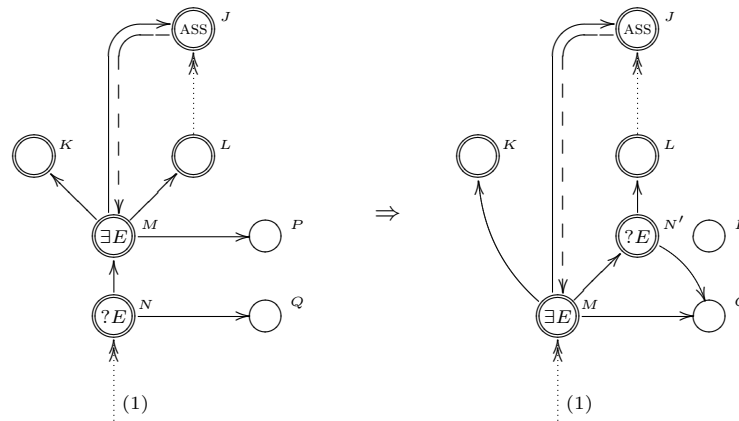


Permutations

Disjunction permutation:

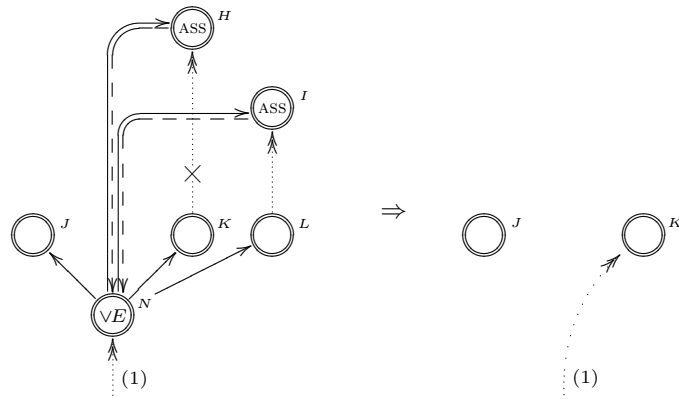


Existential quantifier permutation:

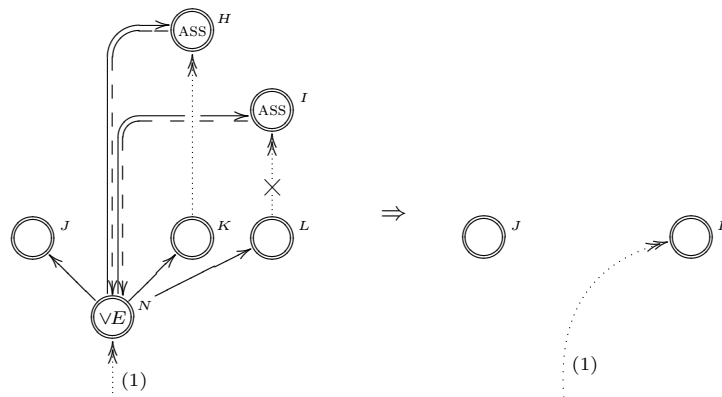


Simplifications

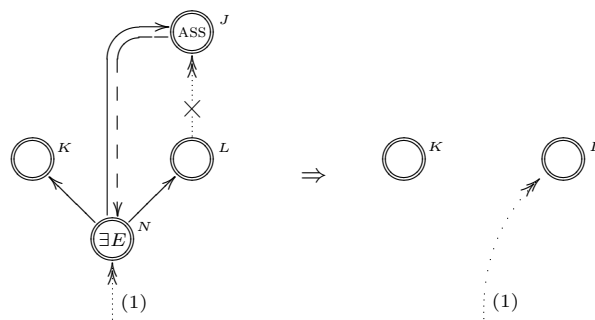
Disjunction simplification (left):



Disjunction simplification (right):



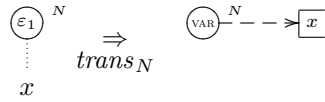
Existential quantifier simplification:



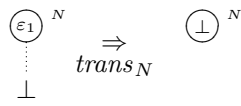
Appendix D: Proof-to-graph translation schemes

Formula translation schemes

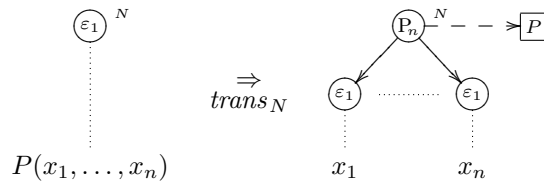
Variables:



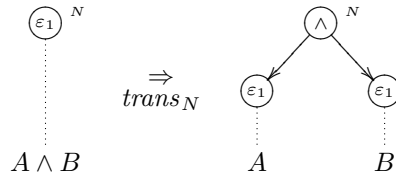
Falsum:



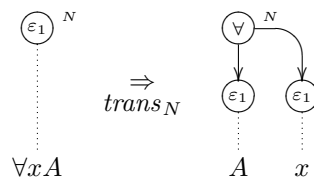
Predicates:



Connectives:

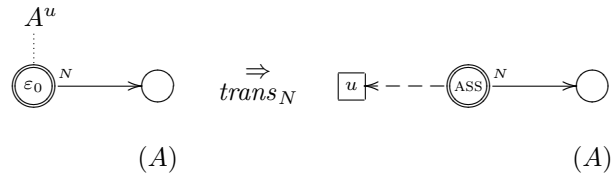


Quantifiers:

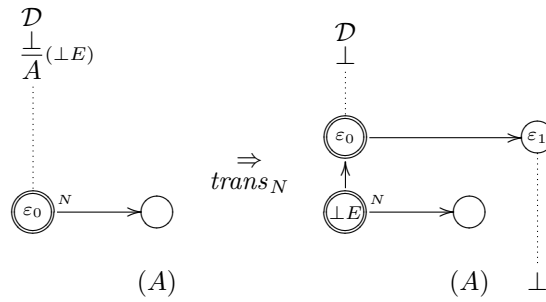


Inference translation schemes

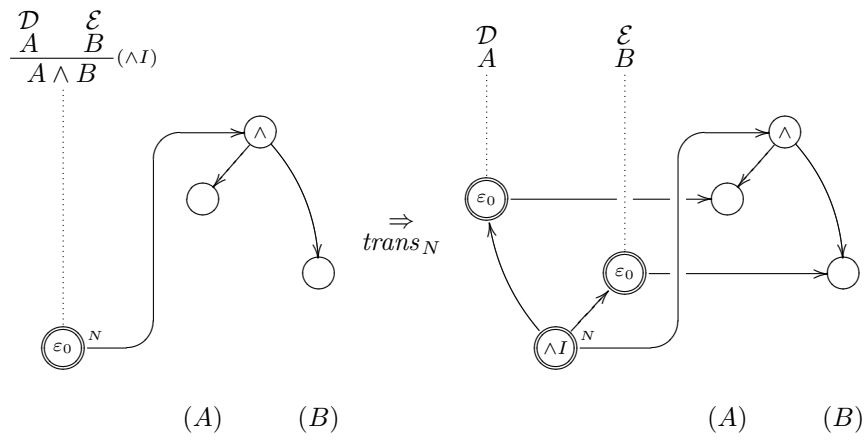
Assumptions:



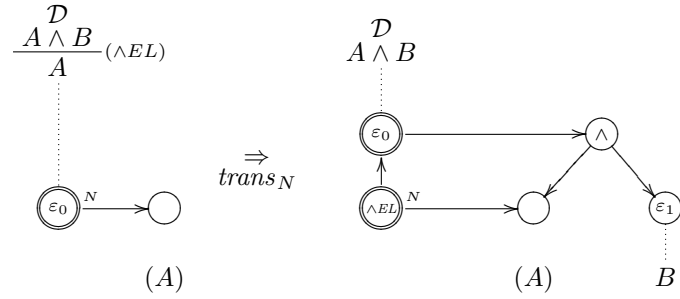
Falsum elimination:



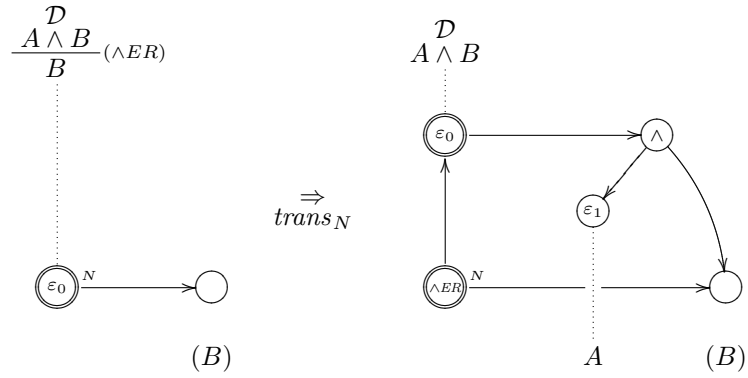
Conjunction introduction:



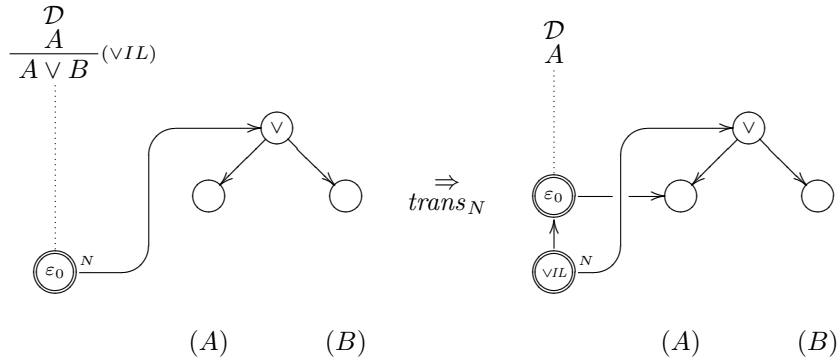
Conjunction elimination (left):



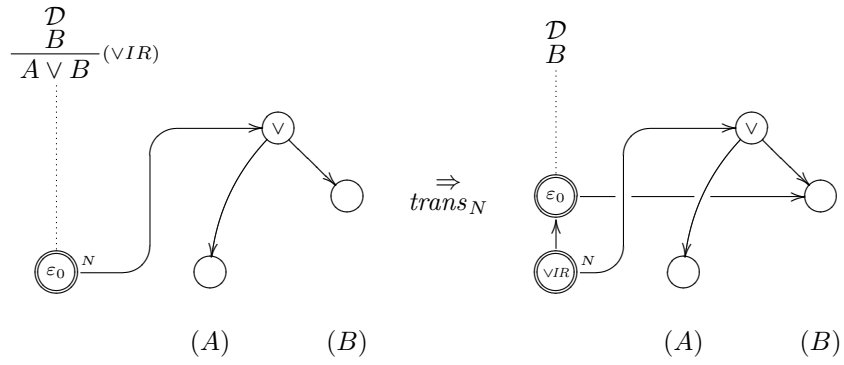
Conjunction elimination (right):



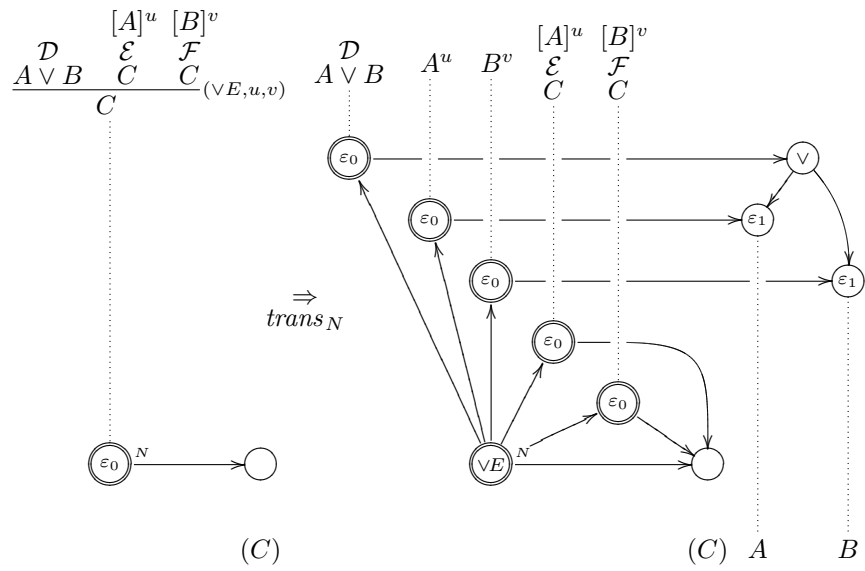
Disjunction introduction (left):



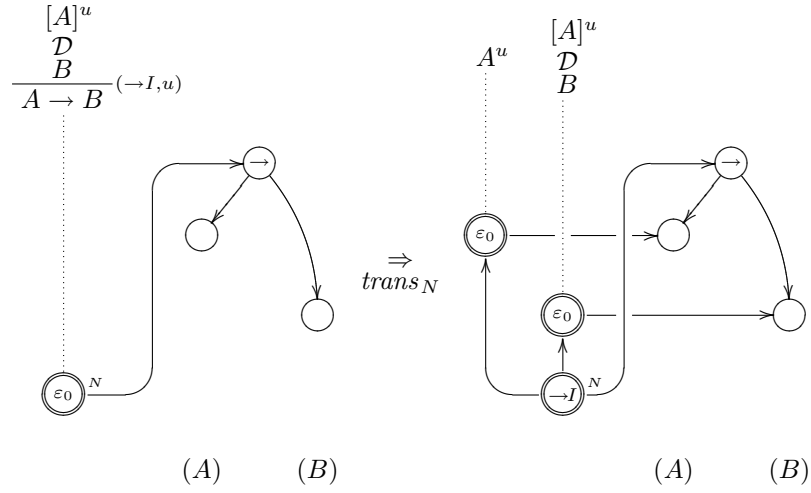
Disjunction introduction (right):



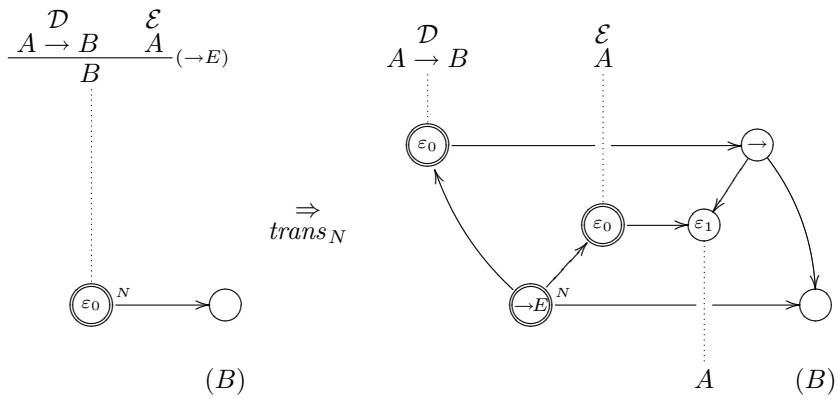
Disjunction elimination:



Implication introduction:



Implication elimination:



Universal quantifier introduction:

