UNIVERSITY OF OSLO
Department of Informatics

Incremental Proof Search in the Splitting Calculus

Thesis presented for the degree of *Candidatus Scientiarum*

Christian Mahesh Hansen

13th December 2004

Abstract

This thesis presents an incremental proof search procedure based on a variable splitting sequent calculus for first-order logic without equality. By means of an index system for formulae, the calculus generates variable sharing derivations in which γ-inferences for different occurrences of the same formula introduce identical variables. Formula indices are utilized to keep track of how variables are split into different branches of a derivation. This allows closing substitutions to instantiate occurrences of the same variable differently under certain conditions. We represent closing substitutions as syntactic constraints and define an incremental method for calculating these constraints alongside derivation expansion.

Acknowledgments

During my first week as a MS student at Department of Informatics I attended a presentation concerning possible thesis topics. One of the presenters, **Arild Waaler**, talked for no more than three minutes about something he called "automated theorem proving", a subject which I at that time had no prior knowledge of. Nevertheless, the way he held his short presentation immediately caught my attention. When I later asked him to be my supervisor, he accepted and invited me to attend his graduate course in logic. It was the subtle and inspiring lectures of Arild and the excellent student workshops led by teacher's assistant **Roger Antonsen** which introduced me to the fascinating world of logic. Roger later became my second supervisor.

The work culminating in the document you are now reading would not have been possible without the inspiration and support provided by Arild and Roger. When I have been overwhelmed by details and unable to see the big picture, Arild has many a time rescued me from "drowning". With Roger I have had long and interesting conversations about logic and other subjects. His consuming dedication to his work is very inspiring and his scrutinizing attention to detail makes him the perfect proof reader. *I sincerely hope to continue working with them both in the future!* I also want to thank Martin Giese for some very helpful comments in the final stage of the writing process.

Researchers and fellow students at the PMA research group at Department of Informatics have created an inspiring environment for learning and discovery, for which I am grateful. I owe a special thank you to Ragnar Normann for introducing me to *theoretical* computer science with his memorable lectures on database theory.

My parents have always encouraged me to ask questions and, more importantly, find *my own* answers. For that I am eternally grateful! My sister, with her no-mountain-too-high-to-climb attitude to life, has always been a great inspiration to me. You are the best sister anyone can have, Anne, and I dedicate this thesis to you. I love you very much!

Christian Mahesh Hansen
Oslo, 13th December 2004

Contents

Chapter 1

Introduction

Designing an efficient proof search procedure is by no means a trivial task, especially not when entering the undecidable world of classical first-order logic. The nondeterministic nature of a logical calculus must be converted into a deterministic algorithm, which searches for a proof of an input formula by systematically analyzing its intrinsic structure. It is desirable to reduce the number of analyzation steps needed in order to find a proof, if it exists. Optimizing a search procedure can be done by fine-tuning the analyzation process, but just as important is improvement of the *proof detection* mechanism, which controls termination of the procedure. Efficient proof detection is the main topic of this thesis.

The logical system serving as basis for our proof search procedure is a free variable sequent calculus for first-order logic without equality. By means of an index system for formulae, the calculus generates *variable sharing* derivations, in which identical variables are introduced when analyzing different occurrences of the same universally quantified formula. These derivations are invariant under order of rule application, meaning that interchanging the inference order in a derivation does not alter the leaf sequents. Variable sharing derivations are closely related to matrix systems, facilitating connection-driven proof search [31].

If the variables in a variable sharing derivation are instantiated *rigidly*, i.e. every occurrence of a variable is instantiated with the same term, the search space cannot be restricted branchwise. This prohibits early termination in some cases of unprovability. It is however the case that some occurrences of a variable are *independent* of others, in the sense that it is sound to instantiate them differently. Formula indices can be utilized to keep track of how variables are split into different branches of a derivation. Each variable occurrence is labelled with a splitting history from which information needed to determine variable independence can be extracted. This technique is called *variable splitting*.

With rigid variable instantiation, the question of whether a variable sharing derivation is closed, and thus is a proof, can be formulated as a *unification problem*. The *closure check* consists of finding a *closing* substitution, i.e. a substitution unifying one connection from each leaf sequent of the derivation. Early proof detection during a proof search requires frequent closure checks. Since the number of possible combinations of connections grows exponentially with derivation expansion, a global closure check is not feasible. We can however utilize the context sharing nature of the inference rules of the calculus to construct an *incremental* closure check. Satisfiable sets of equations for partial connection sets are distributed as syntactic constraints following the intrinsic structure of the derivation, and the constraints are updated incrementally in parallel with derivation expansion. The proof search is terminated as soon as a constraint represents a unifier for a *spanning* set of connections, a set containing exactly one connection from each leaf sequent.

The task of identifying exactly when variable occurrences are independent of each other is called the variable splitting problem. We present a variable splitting calculus in which closing substitutions must unify a set of *primary* and *balancing* equations generated from a spanning connection set. The primary equations correspond the unification problem generated in the non-splitting calculus, and the balancing equations reenforce broken identities between variable occurrences caused by skewness in derivations. In addition, every substitution induces a *dependency relation* on inferences in a derivation according to how variable occurrences are split by the substitution. In order for a substitition to be closing, the induced dependency relation must be *acyclic*.

An incremental proof search procedure based on the splitting calculus must reflect these extended requirements. We present two different approaches to defining incremental constraints; one in which the generation of equations is done incrementally and the cycle check is done globally, and one in which both generation of equations and cycle checking is done incrementally. It is shown that in order to determine whether a closing substitution exists, it is sufficient to cycle check a most general unifier for the primary and secondary equations.

The field of variable splitting is not yet fully explored. Among the unsolved problems is to determine the weakest restrictions possible to put on closing substitutions without compromising consistency. Because of this, calculus correctness is not a central topic in this thesis. Instead we focus on defining incremental closure check procedures and showing their correctness, i.e. whether the closure check captures the closure condition *defined in the calculus*. It will become apparent that an incremental proof search procedure is far more efficient than its counterpart utilizing frequent global closure checks. Complexity analysis of the presented proof search procedures is however beyond the scope of this thesis.

Chapter Guide

In *Chapter 2* I present sequent calculus notation and terminology. Syntax and semantics for ground sequent calculi are presented before the concept of free variables is introduced. Here I also define the *formula index system*, an important part of the syntax in later chapters. A reader familiar with free variable sequent calculi or tableaux systems may skip this chapter entirely, although I recommend reading Section 2.2.2 and 2.2.1. In *Chapter 3* I define an incremental proof search procedure for variable sharing derivations with rigid variable instantiation. I define a constraint language and present both a global and an incremental closure check and show their correctness. In *Chapter 4* I present the variable splitting technique and define an incremental proof search procedure incorporating variable splitting and show its correctness.

Typographic Conventions

When a term or a concept is introduced for the first time it is emphasized *like this* to indicate the place it is defined. Some definitions are enclosed in an environment

1.1 Definition *This is a definition.*

and others occur within ordinary text, indicating the importance of the defined concept.

Scientific Acknowledgment and Contribution

The variable splitting sequent calculus [1, 2, 32] is due to Arild Waaler and Roger Antonsen at Department of Informatics, University of Oslo. The incremental closure framework [17, 18] was originally presented by Martin Giese. My contributions consist of an explication of parts of the incremental closure framework in the context of a free variable sequent calculus with rigid variable instantiation, and an adaptation of incremental proof search to a variable splitting calculus.

Chapter 2

Sequent Calculi: Notation and Terminology

Although there are numerous publications written about the sequent calculus, I feel that it is important to be explicit about the terminology used in this thesis. Proof search is essentially a matter of syntax manipulation, and in order to avoid ambiguity in the discussion of search procedures it is fruitful to clarify the syntactic notions. Fundamental concepts and a ground sequent calculus is presented in Section 2.1. In Section 2.2 I shall introduce the free variable sequent calculus which serves as a basis for our search procedures. The free variable syntax differs slightly from common practice as it incorporates an *index system* [3] for formulae, forcing the generated derivations to be variable sharing.

2.1 Ground Systems

I shall introduce syntax and semantics for closed first-order formulae and sequents. Common notions regarding sequent calculi and proof search procedures are presented, and the ground sequent calculus LK is used as an example to demonstrate concepts and to serve as a basis onto which the free variable calculus is built. The calculus is similar to Gentzen's *Logische Kalküle* [15], from which the name LK is collected.

2.1.1 Syntax

The syntactic objects are built up of symbols from a *first-order language*. All such languages have a common part consisting of a set of *logical connectives* $\{\wedge, \vee, \rightarrow, \neg, \forall, \exists\}$, a set of *punctuation symbols* $\{\text{`('}, \text{`)'}, \text{`,'}\}$ and a countably infinite set of *quantification variables* $\{v_1, v_2, v_3, \ldots\}$. The parts that vary

from language to language and thus define a particular first-order language are countable disjoint sets $\mathcal{F}$ of *function symbols* and $\mathcal{P}$ of *predicate symbols*[1]. With each function and predicate symbol we associate a natural number called the *arity* of the symbol, which indicates the number of arguments the symbol takes. A function symbol with arity 0 is called a *constant*. Function and predicate symbols with arity 1 (2) are called *unary* (*binary*) symbols. $\forall$ and $\exists$ are referred to as *quantifiers*, $\neg$ as a *unary connective*, and $\wedge$, $\vee$ and $\rightarrow$ as *binary connectives*. The *propositional connectives* are $\wedge$, $\vee$, $\rightarrow$ and $\neg$.

We will denote quantification variables by x, y, z, function symbols by f, g, h, constants by a, b, c, d and predicate symbols by P, Q, R. The arities of function and constant symbols will be clear from the context. A first-order language with a set of function symbols $\mathcal{F}$ and a set of predicate symbols $\mathcal{P}$ will be denoted $\mathcal{L}(\mathcal{F}, \mathcal{P})$. Since first-order languages are not the main objects of study I will mostly refrain from defining the particular language used from case to case. For simplicity, I will assume a fixed first-order language $\mathcal{L}$ consisting of infinite sets of function and predicate symbols of all arities, unless otherwise stated.

2.1 Definition *The set of $\mathcal{L}$-terms over a first-order language $\mathcal{L}(\mathcal{F}, \mathcal{P})$, denoted $\mathcal{T}(\mathcal{L})$, is the smallest set such that:*

- *If x is a quantification variable, then x is in $\mathcal{T}(\mathcal{L})$.*
- *If f is an n-ary function symbol in $\mathcal{F}$ and $t_1, \ldots, t_n$ are in $\mathcal{T}(\mathcal{L})$, then $f(t_1, \ldots, t_n)$ is in $\mathcal{T}(\mathcal{L})$.*

A ground term is a term in which no quantification variables occur.

2.2 Example *If a, f and g are function symbols of a language $\mathcal{L}$ and x is a quantification variable, then a, $f(a)$, x, $g(f(a), x)$ and $g(x, a)$ are all $\mathcal{L}$-terms, of which a and $f(a)$ are* ground. *The function symbol f is* unary, *g is* binary *and a is a* constant.

I will mostly refrain from writing punctuation symbols when denoting terms, as long as term parsing is unique from the context. The terms in Example 2.2 will thus be written a, fa, x, $gfax$ and gxa. The symbol $\vec{t}$ is shorthand for a finite term list $t_1, \ldots, t_n$.

2.3 Definition *An atomic basic $\mathcal{L}$-formula over a first-order language $\mathcal{L}(\mathcal{F}, \mathcal{P})$ is any object of the form $P(t_1, \ldots, t_n)$ in which P is an n-ary predicate symbol in $\mathcal{P}$ and $t_1, \ldots, t_n$ are $\mathcal{L}$-terms.*

2.4 Definition *The set of basic $\mathcal{L}$-formulae over a first-order language $\mathcal{L}(\mathcal{F}, \mathcal{P})$ is the smallest set such that:*

[1] Predicate symbols are also referred to as *relation symbols* in the literature.

- *Any atomic basic $\mathcal{L}$-formula is a basic $\mathcal{L}$-formula.*
- *If φ is a basic $\mathcal{L}$-formula, then $\neg\varphi$ is a basic $\mathcal{L}$-formula.*
- *If φ and ψ are basic $\mathcal{L}$-formulae, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\varphi \to \psi)$ are basic $\mathcal{L}$-formulae.*
- *If φ is a basic $\mathcal{L}$-formula and x is a quantification variable, then $\forall x(\varphi)$ and $\exists x(\varphi)$ are basic $\mathcal{L}$-formulae.*

2.5 Example *We extend the language $\mathcal{L}$ from Example 2.2 with the unary predicate symbol P and the binary predicate symbol Q. Then $P(a)$, $(P(x) \to Q(x,y))$, $\neg(P(x) \to Q(x,y))$ and $\forall x(\exists y(\neg(P(x) \to Q(x,y))))$ are basic $\mathcal{L}$-formulae in which $P(a)$, $P(x)$ and $Q(x,y)$ are atomic basic $\mathcal{L}$-formulae.*

The notion 'basic formula' defined here corresponds to the concept of a *formula* commonly used in the literature. We reserve the word 'formula' for a certain extension of the set of basic formulae which will be defined later in this chapter. I will however sometimes refer to basic formulae as just formulae, since there is no risk of mixing the two concepts in the context of the present section. Note that most of the notions defined in the following regarding basic formulae also apply to formulae.

As for terms, we will omit unnecessary punctuation symbols when denoting formulae. The formulae of Example 2.5 will be written Pa, $Px \to Qxy$, $\neg(Px \to Qxy)$ and $\forall x\exists y\neg(Px \to Qxy)$. As one can see, it is no exaggeration to state that this notation increases readability compared to the more pedantic one in the example. As seen in Definition 2.4, I will use φ and ψ as placeholders for formulae throughout this thesis. They are implicitly universally quantified over the set of (basic) formulae. Likewise, $\circ$ spans over the set of binary connectives and Q spans over the set of quantifiers.

Subformulae are defined according to [12]. The *immediate subformula* of $\neg\varphi$ is φ. The immediate subformulae of $\varphi \circ \psi$ are φ and ψ, and the immediate subformula of $\mathsf{Q}x\varphi$ is φ. Atomic formulae have no subformulae. The set of *subformulae* of a formula φ is defined as the smallest set that contains φ and contains with each member, the immediate subformulae of that member. A formula is an *improper subformula* of itself. The *main connective* of a formula of the form $\neg\varphi$ is $\neg$, the main connective of $\varphi \circ \psi$ is the binary connective $\circ$. For a formula $\mathsf{Q}x\varphi$, the main connective is the quantifier Q.

The set of *free variables* occurring in a formula is recursively defined as follows. The set of free variables in a formula of the form $\neg\varphi$ is the set of free variables in φ. The set of free variables in $\varphi \circ \psi$ is the union of the sets of free variables in φ and ψ. The set of free variables in a formula $\mathsf{Q}x\varphi$ is defined as the set of free variables in φ with x removed. The set of free variables in an atomic formula $P(t_1, \ldots, t_n)$ is the set of free variables occurring in the

terms $t_1, \ldots, t_n$. In a formula of the form $\mathsf{Q}x\varphi$, all *free* occurrences of x in φ are *bound* by the quantifier Q. A formula is *closed* if the set of free variables occurring in it is empty.

2.6 Example *The set of free variables in $Pxyzxy$ is $\{x, y, z\}$. The set of free variables in $Px \to \forall xPx$ is the singleton $\{x\}$. Note that the leftmost occurrence of x is* free*, but the rightmost occurrence is* bound *by the universal quantifier. The formula $\forall x \forall y Pxy$ is closed, since the set of free variables occurring in it is empty.*

A *substitution* is a function having the set of quantification variables as its domain and a set of terms as its codomain. Throughout this section, the codomain of all substitutions is the set $\mathcal{T}(\mathcal{L})$ for a first-order language $\mathcal{L}$ given by the context. The *support* of a substitution σ is the set of variables v in its domain such that $\sigma(v) \neq v$. A substitution has *finite support* if its support set is finite. The notation $\{x_1/t_1, \ldots, x_n/t_n\}$ is shorthand for the substitution σ having finite support and mapping the variables $x_1, \ldots, x_n$ to the terms $t_1, \ldots, t_n$ respectively. A substitution σ is *ground* if $\sigma(v)$ is a ground term for all variables v in its support set.

Substitutions are extended to terms and formulae in the following way. $t\sigma$ ($\varphi\sigma$) denotes the result of applying a substitution σ to a term t (formula φ). For constants c, $c\sigma = c$. For terms of the form $f(t_1, \ldots, t_n)$, $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. For atomic formulae we define $P(t_1, \ldots, t_n)\sigma = P(t_1\sigma, \ldots, t_n\sigma)$. Further, we define $(\neg\varphi)\sigma = \neg(\varphi\sigma)$ and $(\varphi \circ \psi)\sigma = (\varphi\sigma) \circ (\psi\sigma)$, in which $\circ$ is a binary connective. For quantifiers Q, we define $(\mathsf{Q}x\varphi)\sigma = (\mathsf{Q}x)(\varphi\sigma_x)$ where σ_x is like σ except that $\sigma_x(x) = x$.

A variable is not allowed to become bound as a result of applying a substitution to a formula. Consider the formula $\forall xPxy$ and the substitution $\rho = \{y/x\}$. If we apply ρ in a naive way, the variable x replacing y becomes bound by the universal quantifier. This problem is solved as follows. Let σ be a substitution $\{x_1/t_1, x_2/t_2, \ldots\}$ in which the variable x occurs in one of the terms t_i and φ a formula of the form $\mathsf{Q}x\psi$. In such cases, we define $\varphi\sigma = (\mathsf{Q}z(\psi[x/z]))\sigma$ in which z is a variable *not* occurring free in ψ and *not* occurring in any term t_i. As a result of applying ρ to the example formula $\forall xPxy$ above, we get for instance $\forall zPzx$. We sometimes denote the result of applying a substitution having finite support $\{x_1/t_1, \ldots, x_n/t_n\}$ to the formula φ as $\varphi[x_1/t_1, \ldots, x_n/t_n]$.

Let σ and τ be substitutions. The *composition* of σ and τ, denoted $\sigma\tau$, is defined as the substitution such that $x(\sigma\tau) = (x\sigma)\tau$ for each quantification variable x. If there is some substitution ρ such that $\tau = \sigma\rho$, then σ is *more general* than τ. A substitution σ is *idempotent* if $\sigma = \sigma\sigma$.

2.7 Example *Let $\sigma = \{u/fv\}$, $\tau = \{u/fa, v/a\}$ and $\rho = \{v/a\}$ be substitutions. Then, $\sigma\rho = \tau$, and thus σ is more general than τ. The substitution*

$\sigma = \{u/fv, v/w\}$ *is* not *idempotent, since* $\sigma\sigma = \{u/fw, v/w\}$*, but* $\sigma\sigma$ *is idempotent, since* $(\sigma\sigma)(\sigma\sigma) = \sigma\sigma$*.*

2.8 Definition *A* sequent *is an object of the form* $\Gamma \vdash \Delta$*, where* Γ *and* Δ *are finite* multisets *of closed basic formulae. The formula set* Γ *(*Δ*) is called the* antecedent *(*succedent*) of the sequent. If* $\Gamma' \subseteq \Gamma$ *and* $\Delta' \subseteq \Delta$*, then* $\Gamma' \vdash \Delta'$ *is a* subsequent *of* $\Gamma \vdash \Delta$*.*

The non-standard requirement that a sequent contains only closed formulae simplifies the definition of sequent semantics in the next section. The symbol '$\vdash$' is called the *sequent symbol*. Since the antecedent and succedent are defined as multisets, a formula can occur more than once in a sequent. Set operations are extended to multisets in the usual way. If Γ is a multiset of closed basic formulae and φ is a closed basic formula, then Γ, φ denotes $\Gamma \cup \{\varphi\}$.

2.9 Example *Both*

$$\forall x(Px \to Qx), Pa \vdash Qa$$

and

$$\forall xPx, \forall xPx \vdash \forall xPx$$

are sequents. In the latter, the formula $\forall xPx$ *has three occurrences, two in the antecedent and one in the succedent.*

2.1.2 Semantics

In order to define validity for sequents, we need some basic semantical definitions. Truth values for closed formulae are defined using the concept of an *extended language*, taken from [1]. Since consistency of sequent calculi is not the primary issue in this thesis, consider this section a brief lineup of the essentials. I refer to other sources for a more detailed discussion (e.g. [12, 21]).

A *model* M for a first-order language $\mathcal{L}(\mathcal{F}, \mathcal{P})$ consists of a non-empty set $|M|$, called a *domain*, and an *interpretation function* $(\cdot)^M$. For all n-ary function symbols f in $\mathcal{F}$ and predicate symbols P in $\mathcal{P}$, we require that f^M is a function from $|M|^n$ to $|M|$ and P^M is a relation over $|M|^n$. $\mathcal{L}$-terms are interpreted recursively, i.e. $f(t_1, \ldots, t_n)^M = f^M({t_1}^M, \ldots, {t_n}^M)$.

2.10 Example *Let* M *be a model having domain* $\{0, 1\}$ *and interpreting the function symbols* a*,* b *and* f *such that* $a^M = 0$*,* $b^M = 1$*,* $f^M(0,0) = f^M(1,1) = 1$ *and* $f^M(0,1) = f^M(1,0) = 0$*. Then* M *interprets the term* $f(fab, fbb)$ *as 0.*

The *extended language* $\mathcal{L}(M)$ is like $\mathcal{L}$, except that we have added a constant symbol $\bar{a}$ for each element a in $|M|$. We require that all models for $\mathcal{L}(M)$ interpret $\bar{a}$ as a. When evaluating closed basic $\mathcal{L}$-formulae in a model M we will use the extended language $\mathcal{L}(M)$ and assume M to be a model for $\mathcal{L}(M)$ by interpreting $\bar{a}$ as a. A closed basic $\mathcal{L}$-formula φ is *true* in M or, equivalently, M *satisfies* φ, written $M \models \varphi$, as defined in the following.

- For atomic formulae: $M \models P(t_1, \ldots, t_n)$ if $\langle {t_1}^M, \ldots, {t_n}^M \rangle \in P^M$.
- $M \models \neg\varphi$ if it is *not* the case that $M \models \varphi$.
- $M \models (\varphi \wedge \psi)$ if $M \models \varphi$ and $M \models \psi$.
- $M \models (\varphi \vee \psi)$ if $M \models \varphi$ or $M \models \psi$.
- $M \models (\varphi \to \psi)$ if $M \models \varphi$ does not hold or $M \models \psi$ holds.
- $M \models \forall x\varphi$ if $M \models \varphi[x/\bar{a}]$ for all a in $|M|$.
- $M \models \exists x\varphi$ if $M \models \varphi[x/\bar{a}]$ for one a in $|M|$.
- For sets S of closed basic $\mathcal{L}$-formulae, $M \models S$ if $M \models \varphi$ for all φ in S.

A closed basic $\mathcal{L}$-formula φ or a set of closed basic $\mathcal{L}$-formulae is *satisfiable* if there exists a model satisfying it. M *falsifies* φ (equiv. φ is *false* in M) if it is *not* the case that $M \models \varphi$.

2.11 Definition *A sequent $\Gamma \vdash \Delta$ is* valid *if all models satisfying Γ also satisfy a formula in Δ. A* countermodel *to a sequent $\Gamma \vdash \Delta$ is a model satisfying* all *formulae in Γ and falsifying* all *formulae in Δ. A sequent having a countermodel is* falsifiable.

It follows that a sequent is valid if and only if it has no countermodel.

2.12 Example *The sequent*

$$\forall x Px, Pa \vdash Pa$$

is valid, since all models satisfying the antecedent also satisfies Pa in the succedent. The sequent

$$\forall x(Px \vee Qx) \vdash \forall x Px$$

is not valid. A countermodel is for instance a model satisfying Qx and falsifying Px for all elements x in its domain.

Any sequent $\Gamma \vdash \Delta$ can be represented as a closed basic formula φ such that φ is true in all models if and only if $\Gamma \vdash \Delta$ is valid. Let $\Gamma = \{\varphi_1, \ldots \varphi_n\}$ and $\Delta = \{\psi_1, \ldots, \psi_m\}$. Then, $\Gamma \vdash \Delta$ corresponds to the formula $(\varphi_1 \wedge \ldots \wedge \varphi_n) \to (\psi_1 \vee \ldots \vee \psi_m)$. Hence, the commas in the antecedent of a sequent are conjunctive and the commas in the succedent are disjunctive.

2.1.3 Sequent Calculi

A *rule* is a binary relation either on S or from $S \times S$ to S, where S is the set of all sequents. A rule of the former (latter) type relates a *premiss* (pair of *premisses*) to a *conclusion* and is called a *one-premiss* (*two-premiss*) rule. The members of a rule θ are denoted

$$\frac{\text{premiss}}{\text{conclusion}} \qquad \text{or} \qquad \frac{\text{premiss1} \quad \text{premiss2}}{\text{conclusion}}$$

and are called *θ-inferences*. A *schema* for a rule θ is an object containing placeholders such that each θ-inference is the result of replacing the placeholders with formulae. Schemas have premisses and conclusion and are denoted in the same way as inferences. We require that all premisses and conclusions of schemas contain the symbols Γ and Δ, and we refer to the formulae replacing them as *extra formulae* or *context*. Hence, rules defined by schemas are *context sharing*. Conclusions and premisses of schemas can contain formula placeholders with logical connectives and quantification variables. The formula obtained by replacing placeholders with formulae is called a *principal formula* if it occurs in the conclusion, and an *active formula* if it occurs in a premiss.

2.13 Example *The object*

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}\ \mathsf{R}\wedge$$

is a schema for the two-premiss rule $\mathsf{R}\wedge$*, and*

$$\frac{\exists x Px \vdash \forall x Px, \exists x Rx \quad \exists x Px \vdash \forall x Qx, \exists x Rx}{\exists x Px \vdash \forall x Px \wedge \forall x Qx, \exists x Rx}\ \mathsf{R}\wedge$$

is a $\mathsf{R}\wedge$*-inference having principal formula* $\forall x Px \wedge \forall x Qx$*, active formulae* $\forall x Px$ *and* $\forall x Qx$*, and context* $\exists x Px, \exists x Rx$*.*

Throughout this thesis we only consider rules whose inferences are instantiations of a given rule schema. Thus, all rules under consideration are context sharing. In the following I will refer to a rule schema as just a rule. Also, the concepts of principal formula, active formula and extra formula used for inferences transfer to rule schemas in the obvious way.

$$\frac{\Gamma, \varphi, \varphi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\ \mathsf{LC} \qquad \frac{\Gamma \vdash \varphi, \varphi, \Delta}{\Gamma \vdash \varphi, \Delta}\ \mathsf{RC}$$

Figure 2.1: The contraction rules.

The rules in Figure 2.1, called left and right *contraction* (denoted LC and RC), play a special role. Although they are not included as rules of their own in any of the calculi we are presenting, their *implicit* presence[2] is needed in the rules handling universally quantified formulae to ensure that our calculi are *complete*[3].

All rules under consideration in this thesis have the property that a complex formula in the conclusion is broken into its less complex subparts in the premisses (as seen in Example 2.13). Rules of this kind are *synthetic* when viewed from above as generating conclusions from premisses, and *analytic* when viewed from below as generating premisses from conclusions. Both point of views are of proof theoretic interest, but the latter is more suitable for automated proof search. Hence, we will focus on the analytic point of view.

Derivations are finitely branching trees regulated by the rules of the calculus at hand. The nodes of a derivation are labelled by sequents. The sequent labelling the root node is called the *root sequent* and the sequents labelling the leaf nodes are called *leaf sequents*.

2.14 Definition *For a fixed set $\mathcal{R}$ of rules we define inductively the* set of derivations *generated by the rules as the least set satisfying the following conditions.*

- *A sequent is a derivation.*
- *If π is a derivation with a leaf sequent l, θ is a one-premiss rule in $\mathcal{R}$ and l is the conclusion of a θ-inference with premiss l', then the result of extending π with l' above l is a derivation:*

$$\begin{array}{c} l \\ \vdots \\ \pi \end{array} \quad \rightsquigarrow \quad \dfrac{\dfrac{l'}{l}}{\vdots}\,\pi$$

- *If π is a derivation with a leaf sequent l, θ is a two-premiss rule in $\mathcal{R}$ and l is the conclusion of a θ-inference with premisses l' and l'', then the result of extending π with l' and l'' above l is a derivation:*

$$\begin{array}{c} l \\ \vdots \\ \pi \end{array} \quad \rightsquigarrow \quad \dfrac{\dfrac{l' \qquad l''}{l}}{\vdots}\,\pi$$

[2] Se page 14.
[3] See Definition 2.15.

In the above definition the process of extending π with the premisses of θ is called a *θ-expansion* or just an *expansion* (of π). If φ is the principal formula of θ, we equivalently say that we *expand* φ. The leaf sequent l of π containing the expanded formula is called the *expanded leaf sequent*, and the branch in π having l as leaf sequent is called the *expanded branch*. Expansion by two-premiss inferences splits the expanded branch into two new branches. Two-premiss rules (inferences) are thus sometimes referred to as *branching* rules (inferences). One-premiss rules (inferences) are referred to as *non-branching* rules (inferences).

A *sequent calculus* $\mathcal{K}$ consists of a set of rules $\mathcal{R}$ and a *closure condition*. Derivations regulated by $\mathcal{R}$ are called $\mathcal{K}$-derivations. The *closure condition* of $\mathcal{K}$ is a condition $\mathcal{K}$-derivations must meet in order to be *closed*. A *closed* $\mathcal{K}$-derivation is called a *$\mathcal{K}$-proof* (of its root sequent). A sequent $\Gamma \vdash \Delta$ is *$\mathcal{K}$-provable* if it exists a $\mathcal{K}$-proof having $\Gamma \vdash \Delta$ as its root sequent.

2.15 Definition *A sequent calculus $\mathcal{K}$ is* sound *if every $\mathcal{K}$-provable sequent is valid. $\mathcal{K}$ is* complete *if every valid sequent is $\mathcal{K}$-provable.*

One can easily construct calculi that are sound but not complete and vice versa. An example of the latter is the calculus stating that any sequent is provable. This calculus is obviously complete, since all valid sequents are provable. But since it also proves sequents which are not valid, it is not sound.

2.1.4 Proof Search Procedures

The inference rules of a sequent calculus may provide several expansion options at each expansion step in a proof search process. This nondeterminism is not desirable in the context of automated proof search. We need to construct a deterministic procedure which controls derivation expansion and checks whether the current derivation has reached a closable state.

A *selection function* utilizes the rules of a sequent calculus to return for each derivation at most one successor derivation. A selection function is *fair* if it expands every option in finite time. A *proof search procedure* takes a sequent s as input and repeatedly applies some fixed selection function to its own output using s as the initial derivation. A proof search procedure is *complete* if it terminates for every valid input sequent. The sequent calculi in this thesis are known to be *proof confluent*, meaning that no matter how we choose the expansion steps in a proof search starting with a provable sequent, it is always possible to reach a proof. A proof search procedure equipped with a fair selection function based on a complete proof confluent sequent calculus is complete. We require that a proof search procedure terminates as soon as the output of the associated selection function is a proof of s. Hence, it

must incorporate a proof detection algorithm, referred to as a *closure check*. In the following chapters we assume that the selection function of the proof search procedure at hand is fair, and focus on how to design efficient closure checks.

2.1.5 The Sequent Calculus LK

The rules of the sequent calculus LK are listed in Figure 2.2. They are divided into four types, following Smullyan's *uniform notation* [29]: The *α-rules* are the non-branching rules in which the main connective of the principal formula is propositional. The branching rules are called *β-rules*. The *δ-rules* handle succedent (antecedent) occurrences of universally (existentially) quantified formulae, and the *γ-rules* handle antecedent (succedent) occurrences of universally (existentially) quantified formulae. From now on, θ denotes a rule (or an inference) of type α, β, δ or γ. The principal formula of a rule or inference of type θ is called a θ-formula.

The δ and γ-rules, in which the principal formula is of the form $\mathsf{Q}x\varphi$, require some extra explanation. The active formula of a δ-inference is generated by substituting the constant symbol a for x in the quantified formula φ. The symbol a is called an *eigenparameter*, and we require that a does not occur in the conclusion of the δ-inference, i.e. neither in the quantified formula φ nor in any for the formulae in $\Gamma \cup \Delta$. This requirement is called the *eigenparameter condition* for LK. In γ-inferences, we substitute an arbitrary *closed* term t of the first-order language at hand for x in the active formula. The premiss in addition contains a copy of the principal γ-formula. This feature of the γ-rules is called *implicit contraction*. It is obvious that all LK-rules preserve the sequent property of containing only *closed* formulae.

2.16 Definition *An* axiom *is a sequent of the form*

$$\Gamma, P(t_1, \ldots, t_n) \vdash P(t_1, \ldots, t_n), \Delta$$

2.17 Example *The sequent*

$$\forall x Px, Pa \vdash Pb, \exists x Qx$$

is not *an axiom, since Pa and Pb do not have the same term as argument, but the sequent*

$$\forall x Px, Pa \vdash Pa$$

is an axiom.

2.18 Definition (Closure Condition of LK) *A* closed branch *in an* LK*-derivation is a branch having an axiom as leaf sequent. A branch is called* open *if it is not closed. An* LK*-derivation is closed if all its branches are closed.*

α-rules

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta}\ \mathsf{L}\wedge \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}\ \mathsf{R}\vee$$

$$\frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \to \psi, \Delta}\ \mathsf{R}\to \qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta}\ \mathsf{L}\neg \qquad \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}\ \mathsf{R}\neg$$

β-rules

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}\ \mathsf{R}\wedge \qquad \frac{\Gamma, \varphi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}\ \mathsf{L}\vee$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \to \psi \vdash \Delta}\ \mathsf{L}\to$$

δ-rules

$$\frac{\Gamma \vdash \varphi[x/a], \Delta}{\Gamma \vdash \forall x\varphi, \Delta}\ \mathsf{R}\forall \qquad \frac{\Gamma, \varphi[x/a] \vdash \Delta}{\Gamma, \exists x\varphi \vdash \Delta}\ \mathsf{L}\exists$$

γ-rules

$$\frac{\Gamma, \forall x\varphi, \varphi[x/t] \vdash \Delta}{\Gamma, \forall x\varphi \vdash \Delta}\ \mathsf{L}\forall \qquad \frac{\Gamma \vdash \exists x\varphi, \varphi[x/t], \Delta}{\Gamma \vdash \exists x\varphi, \Delta}\ \mathsf{R}\exists$$

Figure 2.2: The rules of the sequent calculus LK. We require that the symbol a in the δ-rules is an *eigenparameter* not occurring in any formula in the conclusion. For the γ-rules, t can be any *closed* term of the first-order language at hand.

2.19 Example *The* LK*-derivation*

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall x(Px \wedge Qx), Pa, Qa \vdash \forall xPx, Pa}{\forall x(Px \wedge Qx), Pa \wedge Qa \vdash \forall xPx, Pa}\,\mathsf{L}\wedge}{\forall x(Px \wedge Qx) \vdash \forall xPx, Pa}\,\mathsf{L}\forall}{\forall x(Px \wedge Qx) \vdash \forall xPx}\,\mathsf{R}\forall \qquad \dfrac{\dfrac{\dfrac{\forall x(Px \wedge Qx), Pa, Qa \vdash \forall xQx, Qa}{\forall x(Px \wedge Qx), Pa \wedge Qa \vdash \forall xQx, Qa}\,\mathsf{L}\wedge}{\forall x(Px \wedge Qx) \vdash \forall xQx, Qa}\,\mathsf{L}\forall}{\forall x(Px \wedge Qx) \vdash \forall xQx}\,\mathsf{R}\forall}{\forall x(Px \wedge Qx) \vdash \forall xPx \wedge \forall xQx}\,\mathsf{R}\wedge$$

is closed *since both leaves are* axioms*. Thus, it is an* LK-proof *of the sequent* $\forall x(Px \wedge Qx) \vdash \forall xPx \wedge \forall xQx$*. Note that the eigenparameter* a *is introduced in* both *branches. This is possible since* a *does not occur in any of the conclusions of the* R$\forall$*-inferences. The* LK*-derivation*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi, Pbc, Poa \vdash \psi, Pdc, Pba}{\varphi, Pbc, Poa \vdash \psi, \forall yPyc, Pba}\,\delta_d}{\varphi, Pbc, Poa \vdash \psi, Pba}\,\gamma_c}{\varphi, \exists yPby, Poa \vdash \psi, Pba}\,\delta_c}{\varphi, Poa \vdash \psi, Pba}\,\gamma_b}{\varphi, Poa \vdash \psi, \forall yPya}\,\delta_b}{\varphi, Poa \vdash \psi}\,\gamma_a}{\varphi, \exists yPoy \vdash \psi}\,\delta_a}{\underbrace{\forall x\exists yPxy}_{\varphi} \vdash \underbrace{\exists x\forall yPyx}_{\psi}}\,\gamma_o$$

is not closable. Because of the eigenparameter condition of LK*, it is impossible to construct an axiom in its leaf sequent, no matter how we expand the derivation. The formula* $\forall x\exists yPxy$ *(*$\exists x\forall yPyx$*) in the root sequent is underbraced by* φ *(*ψ*), which means that the formula placeholder will be used to denote the formula throughout the derivation. The inferences are labelled by rule types subscripted by the terms introduced. These notations will be used commonly throughout the thesis.*

If we study the LK-rules in the context of the truth definition given in Section 2.1.2, we see that the rules are designed in order to systematically satisfy (falsify) formulae in the antecedent (succedent) of a sequent. Thus a proof search in LK can bee seen as a systematic search for a countermodel for the root sequent. When an axiom is encountered on a branch, the branch is closed since it is of no use for constructing a countermodel. If all branches are closed, it is impossible to construct a countermodel for the root sequent.

2.20 Proposition *The sequent calculus* LK *is sound and complete.*

A proof can be found in any standard textbook on sequent calculi [13, 19, 20, 24]. The proof of soundness goes by induction on sequent depth[4] in LK-proofs. As an intermediate result, it is shown that every LK-rule preserves validity downwards, i.e. that the conclusion is valid under the assumption that the premisses are valid. The completeness proof is done as a reductio argument from countermodel existence, i.e. that every unprovable sequent has a countermodel. The countermodel existence proof is done constructively from an open branch in a limit object (a possibly infinite derivation object) generated by a proof search procedure equipped with a fair selection function for LK. A model is constructed such that it satisfies (falsifies) every atomic formula occurring in an antecedent (succedent) on the open branch. One shows by structural induction on formulae in the open branch that the constructed model satisfies (falsifies) every antecedent (succedent) formula on the branch, and thus is a countermodel for the root sequent.

2.2 Systems with Variables

Since the γ-rules of LK introduce arbitrary closed terms, a selection function for LK will have to deal with the problem of term selection in γ-expansions. It is desirable to reduce the number of expansion steps needed to find a proof, a measure which clearly is influenced by γ-term selection.

$$\dfrac{\dfrac{\dfrac{\dfrac{(Pa \wedge Pb), \boxed{Pc} \vdash \exists x Px, Pa, Pb, \boxed{Pc}}{(Pa \wedge Pb), Pc \vdash \exists x Px, Pa, Pb}\,\gamma_c}{(Pa \wedge Pb), Pc \vdash \exists x Px, Pa}\,\gamma_b}{(Pa \wedge Pb), Pc \vdash \exists x Px}\,\gamma_a}{(Pa \wedge Pb) \wedge Pc \vdash \exists x Px}\,\mathsf{L}\wedge \qquad \dfrac{\dfrac{(Pa \wedge Pb), \boxed{Pc} \vdash \exists x Px, \boxed{Pc}}{(Pa \wedge Pb), Pc \vdash \exists x Px}\,\gamma_c}{(Pa \wedge Pb) \wedge Pc \vdash \exists x Px}\,\mathsf{L}\wedge$$

Figure 2.3: Two LK-proofs of the sequent $(Pa \wedge Pb) \wedge Pc \vdash \exists x Px$. In the leftmost proof, the number of rule applications is doubled compared to the rightmost proof because of unfavorable selection of instantiation terms.

A free variable sequent calculus provides a solution to the problem of selecting γ-terms by letting γ-inferences introduce free variables instead of arbitrary closed terms. In order to close a branch in a free variable derivation, the free variables have to be instantiated with terms in such a way that the arguments of two atomic formulae Pt, Ps occurring in different sides of the leaf sequent become *identical*, or in other words one has to solve the equation $t = s$ in the space of terms [12]. As atomic formulae often have more than one argument and derivations have more than one branch, closing a free variable derivation corresponds to simultaneously solving a set of

[4]The number of inferences to the farthest away leaf sequent above a sequent in a derivation, see Definition 3.38 on page 47.

equations. In this way, the problem of selecting γ-terms in LK is reduced to a unification problem in a free variable calculus.

$$\dfrac{\dfrac{(Pa \wedge Pb), \boxed{Pc} \vdash \exists x Px, \boxed{Pu}}{(Pa \wedge Pb), Pc \vdash \exists x Px}\,{\scriptstyle u/c}\;\gamma_u}{(Pa \wedge Pb) \wedge Pc \vdash \exists x Px}\;\mathsf{L}\wedge$$

Figure 2.4: A free variable derivation for the sequent $(Pa \wedge Pb) \wedge Pc \vdash \exists x Px$. The γ-inference introduce the free variable u and the derivation is closed by any substitution instantiating u with c.

Introduction of free variables by γ-inferences leads to difficulties in selecting the terms introduced by δ-inferences. No matter how the free variables in the conclusion are instantiated the introduced δ-term must be *new*, i.e. it cannot occur in the conclusion. An excellent overview of different approaches to define such δ-terms can be found in [1]. The δ-terms of our free variable calculi are of the form $f(x_1, \ldots, x_n)$, in which $x_1, \ldots, x_n$ are the free variables occurring in the δ-formula in the conclusion. How the function symbol f is selected will become clear later in this chapter.

$$\dfrac{Pufu \vdash Pab}{\exists x Pux \vdash Pab}\;\delta_{fu}$$

Figure 2.5: Example of a δ-inference introducing the term fu.

In a *variable pure* derivation, each γ-inference introduces a free variable not already occurring in the derivation. A *reuse* of free variables can be achieved by letting different occurrences of the same γ-formula introduce the same variable, producing *variable sharing* derivations. Variable sharing derivations are *invariant under order of rule application* [1, 2, 31], meaning that their leaf sequents are the same no matter which order we apply the rules in (for intuitions, see Figure 2.6). This ensures a tight relation to matrix systems, allowing the design of connection-driven proof search procedures [31].

We do not discuss connection-driven proof search in this thesis, but the free variable calculus introduced is constructed bearing such selection functions in mind [32]. Hence we need to provide invariance under order of rule application by incorporating variable sharing in our calculi. To do this, we use an index system for formulae inspired by the one used by Wallen in [33]. When a formula is copied by means of implicit contraction its index is changed, while indices of formulae copied as part of context are left untouched, as

$$\dfrac{\dfrac{Pu \vdash Pa}{\forall xPx \vdash Pa} \quad \dfrac{Pv \vdash Pb}{\forall xPx \vdash Pb}}{\forall xPx \vdash Pa \wedge Pb}\ \textbf{(1)} \qquad \dfrac{\dfrac{Pu \vdash Pa}{\forall xPx \vdash Pa} \quad \dfrac{Pu \vdash Pb}{\forall xPx \vdash Pb}}{\forall xPx \vdash Pa \wedge Pb}\ \textbf{(2a)} \qquad \dfrac{\dfrac{Pu \vdash Pa \quad Pu \vdash Pb}{Pu \vdash Pa \wedge Pb}}{\forall xPx \vdash Pa \wedge Pb}\ \textbf{(2b)}$$

Figure 2.6: Variable pure **(1)** and variable sharing **(2)** derivations for the sequent $\forall xPx \vdash Pa \wedge Pb$. **(2a)** and **(2b)** are permutation variants having identical leaf sequents.

illustrated in Figure 2.7. By labelling a free variable with the index of the principal formula in the γ-inference introducing it, variables introduced by expansion of different occurrences of the same γ-formula are forced to be identical.

$$\dfrac{\forall xPx^{a'}, Pa^{c} \vdash \forall xPx^{b}}{\forall xPx^{a} \vdash \forall xPx^{b}}\ \mathsf{L}\forall$$

Figure 2.7: Let the superscripts be indices of some indexing system. In the premiss of the $\mathsf{L}\forall$-inference, the formula occurrence $\forall xPx^{b}$ is copied as part of the context and the occurrence $\forall xPx^{a'}$ by implicit contraction. This is reflected by their indices when compared to those of the corresponding occurrences in the conclusion.

To distinguish free variables and terms introduced by δ-inferences from the quantification variables and terms of a first-order language, we will refer to them as *instantiation variables*[5] [1, 32] and *Skolem terms*, respectively. A derivation containing instantiation variables does not carry logical force, meaning that the instantiation variables have to be instantiated in order to interpret the formulae occurring in it. To emphasize this fact, derivations containing instantiation variables are from now on called *skeletons* [31]. In the rest of this chapter we extend the ground sequent calculus syntax to include notions needed in order to construct the free variable sequent calculus LK^{v}. The $^{\mathsf{v}}$ indicates that it is a free variable version of the sequent calculus LK.

2.2.1 Syntax

The definition of a first-order language from Section 2.1.1 is extended such that all first-order languages additionally contain a countably infinite set $\mathcal{V}$ of *instantiation variables* of the form u_m^{κ} and a non-empty set $\mathcal{S}$ of *Skolem functions* of the form f_m, in which m is a natural number and κ is a sequence of natural numbers. The sets $\mathcal{V}$ and $\mathcal{S}$ are disjoint. The set $\mathcal{S}$ contains

[5] The terms *meta variables* and *parameters* are common in the literature.

an infinite number of Skolem functions of different arities and is disjoint from the sets of predicate and function symbols of the first-order language. Skolem functions with arity 0, denoted a_m, are called *Skolem constants*. Instantiation variables are denoted u, v, w. Skolem functions are denoted f, g, h and Skolem constants a, b, c, as long as it is clear from the context whether a symbol is a Skolem function or a function symbol.

2.21 Definition *The set of instantiation terms for a first-order language $\mathcal{L}$, denoted $\mathcal{T}_I(\mathcal{L})$, is the smallest set such that:*

- *Every instantiation variable u is in $\mathcal{T}_I(\mathcal{L})$.*
- *If f is an n-ary function symbol in $\mathcal{L}$ and $t_1, \ldots, t_n$ are in $\mathcal{T}_I(\mathcal{L})$, then $f(t_1, \ldots, t_n)$ is in $\mathcal{T}_I(\mathcal{L})$.*
- *If f_m is an n-ary Skolem function and $t_1, \ldots, t_n$ are in $\mathcal{T}_I(\mathcal{L})$, then $f_m(t_1, \ldots, t_n)$ is in $\mathcal{T}_I(\mathcal{L})$.*

An instantiation term is ground if it contains no instantiation variables.

Note that an instantiation term contains no quantification variables.

2.22 Definition *The set of formulae over a first-order language $\mathcal{L}$ is defined from the set of basic $\mathcal{L}$-formulae by substitution:*

- *A basic $\mathcal{L}$-formula is a formula.*
- *If a quantification variable x occurs free in a formula φ and t is an instantiation term, then $\varphi[x/t]$ is a formula.*

A formula is closed if all quantification variables occurring in it are bound.

It is immediate that the set of formulae extends the set of basic formulae defined in Section 2.1.1. Instantiation variables are never bound by quantifiers. Formulae with instantiation terms are generated by the rules of the calculus and do not exist outside such a context. Their purpose is to provide a syntax for free variables and run-time skolemization[6].

A *substitution for instantiation terms* is a substitution having the set of instantiation variables as domain and the set of instantiation terms (over a given first-order language) as its codomain. All notions regarding substitutions defined in Section 2.1.1 apply to substitutions for instantiation terms. The set of *all* substitutions for instantiation terms is denoted $\mathcal{I}$. A substitution σ in $\mathcal{I}$ is *ground* if $\sigma(u)$ is a ground instantiation term for each

[6]The introduction of Skolem terms by δ-inferences is a skolemization process.

instantiation variable u in its support set. In the following, we refer to substitutions for instantiation terms just as substitutions when there is no risk of misunderstanding. Note that substitutions are defined for *all* instantiation variables in $\mathcal{V}$. This will simplify some definitions in later chapters.

2.23 Definition (Unifier) *If s and t are instantiation terms, σ is a substitution in $\mathcal{I}$ and $s\sigma = t\sigma$, then σ is a* unifier *for s and t. If φ and ψ are formulae and $\varphi\sigma = \psi\sigma$, then σ is a* unifier *for φ and ψ.*

2.24 Definition *An* equation *is a tuple, written $t_1 \approx t_2$, in which t_1 and t_2 are instantiation terms. A substitution σ* solves *an equation $t_1 \approx t_2$ if it is a unifier for t_1 and t_2. An equation is* solvable *if there is some substitution solving it. An* equation set *is a set of equations. Let S be an equation set and σ a substitution. We say that σ* satisfies *S, written $\sigma \models S$, if σ solves all equations in S. Further, S is* satisfiable *if there is some substitution satisfying it.*

2.25 Example *The equation set $\{fu \approx fv,\ v \approx gaw,\ fw \approx fb\}$ is satisfiable, since the equations are solved simultaneously by the substitution $\{u/gab, v/gab, w/b\}$.*

2.26 Definition *An* indexed formula *is an object of the form φ^κ in which φ is a formula and κ is a sequence of natural numbers called a* copy history*. All subformulae of φ (this* includes *φ) are assigned* distinct *natural numbers, called* formula numbers*. The* index *of an indexed formula φ^κ is the pair $\overset{\kappa}{m}$ consisting of the copy history κ of the indexed formula and the formula number m of φ. An indexed formula φ^κ is* closed *if the formula φ is closed.*

We write copy histories as strings whenever the parsing is clear from the context; $\langle t_1, \ldots, t_n \rangle$ is written $t_1 \ldots t_n$. Concatenation of copy histories are done by the '.'-operator. If $\kappa = k_1 \ldots k_n$ and $\tau = t_1 \ldots t_m$ are copy histories and p is a natural number, then $\kappa.p = k_1 \ldots k_n p$ and $\kappa.\tau = k_1 \ldots k_n t_1 \ldots t_m$. We also define the operator $'$ for copy histories as $\kappa' := k_1 \ldots k_{n-1}.(t_n + 1)$, i.e. a sequence identical to κ except that the last element is increased by one.

2.27 Example *The following are indexed formulae:*

$$\overset{1}{\forall}x\overset{2}{\exists}y(\overset{4}{P}x \overset{3}{\rightarrow} \overset{5}{Q}y)^{1} \qquad \textit{and} \qquad \overset{10}{\forall}x(\overset{13}{Q}y \overset{15}{\rightarrow} \overset{7}{P}x)^{1.1.1.2.1}$$

The former indexed formula is closed, the latter is not. As a convention, we write the formula numbers below their respective subformulae and the copy history superscripted to the right of the formula. The object

$$\overset{1}{\exists}z(\overset{2}{P}x \overset{1}{\wedge} \overset{3}{Q}x)^{2}$$

is not *an indexed formula, since two subformulae have identical formula numbers.*

Note that in order to be syntactically equal, two indexed formulae must have identical copy histories and their underlying formulae must be assigned formula numbers in the same way. Thus, neither

$$\forall_{1\ 2}^{\ } x P x^{1} \quad \text{and} \quad \forall_{1\ 2}^{\ } x P x^{2}$$

nor

$$\forall_{1\ 2}^{\ } x P x^{1} \quad \text{and} \quad \forall_{3\ 4}^{\ } x P x^{1}$$

are syntactically equal.

2.28 Definition *An* indexed sequent *is an object* $\Gamma \vdash \Delta$ *in which* Γ *and* Δ *are disjoint* sets *of closed indexed formulae.*

Recall from Definition 2.8 that antecedents and succedents of sequents in ground calculi are *multisets* of formulae. Although the antecedent and succedent of an indexed sequent are defined as *sets*, a formula can still occur more than once, provided that different occurrences are indexed differently. Hence, in an indexed sequent different formula occurrences are distinguished syntactically. We often refer to indexed sequents as just sequents and indexed formulae as just formulae, assuming that they are indexed according to the definitions given in this section.

2.29 Example *The following is an example of an indexed sequent:*

$$\forall_{1\ 2}^{\ } x \exists y (P_{4}^{\ } x \underset{3}{\rightarrow} Q_{5}^{\ } y)^{1}, \forall_{6\ 7}^{\ } x P x^{1} \vdash \forall_{8\ 9}^{\ } x Q x^{1}, \forall_{10\ 11}^{\ } x Q x^{1}$$

The indices of the indexed formulae in the sequent above are (from left to right) 1_{1}, 1_{6}, 1_{8} *and* 1_{10}. *The object*

$$\forall_{1\ 2}^{\ } x P x^{1} \vdash \forall_{1\ 2}^{\ } x P x^{1}$$

is not *an indexed sequent, since the antecedent and succedent are not disjoint. We will refrain from writing formula numbers and copy histories of indexed formulae when they are clear from the context.*

2.2.2 The Sequent Calculus LK^{v}

The rules of the free variable sequent calculus LK^{v} define relations on indexed sequents. Therefore, the placeholders in an LK^{v}-rule can only be instantiated in such a way that the conclusion and premisses of the resulting inference are indexed sequents. The α- and β-rules are just like the α- and β-rules in LK, except that principal and active formulae in a rule have identical copy histories. When generating premisses from conclusions, α- and β-rules transfer the copy history of the principal formula to the active formulae in

$$\frac{Pa^{1}_{2}, Pb^{1}_{3} \vdash Pa^{1}_{4}}{Pa^{1}_{2} \wedge_{1} Pb^{1}_{3} \vdash Pa^{1}_{4}}\,\mathsf{L}\wedge \qquad \frac{\vdash Pa^{1}_{2}, Pa^{1}_{4} \quad Pb^{1}_{3} \vdash Pa^{1}_{4}}{Pa^{1}_{2} \rightarrow_{1} Pb^{1}_{3} \vdash Pa^{1}_{4}}\,\mathsf{L}\rightarrow$$

Figure 2.8: Example of an α- and a β-inference of the LK^{v}-system. The copy history of the principal formula is transferred to the active formulae. Extra formulae are copied unaltered.

the premisses. Extra formulae are copied into the premisses without being altered. Figure 2.8 shows an α- and a β-inference.

The δ- and γ-rules of LK^{v} are listed in Figure 2.9. A δ-inference having principal formula $\mathsf{Q}x\varphi^{\kappa}$ introduces the Skolem term $f_m\vec{u}$, in which the number m is the formula number of the principal formula and $\vec{u}$ are the instantiation variables occurring in φ. If φ contains no instantiation variables, the Skolem constant a_m is introduced. As for the α- and β-rules, the copy history of the principal formula is attached to the active formula. Thus, δ-formulae having the same formula number introduce identical Skolem functions when expanded.

δ-rules

$$\frac{\Gamma \vdash \varphi[x/f_m\vec{u}]^{\kappa}, \Delta}{\Gamma \vdash \forall x\varphi^{\kappa}, \Delta}\,\mathsf{R}\forall \qquad \frac{\Gamma, \varphi[x/f_m\vec{u}]^{\kappa} \vdash \Delta}{\Gamma, \exists x\varphi^{\kappa} \vdash \Delta}\,\mathsf{L}\exists$$

γ-rules

$$\frac{\Gamma, \forall x\varphi^{\kappa'}, \varphi[x/u^{\kappa}_{m}]^{\kappa.1} \vdash \Delta}{\Gamma, \forall x\varphi^{\kappa} \vdash \Delta}\,\mathsf{L}\forall \qquad \frac{\Gamma \vdash \exists x\varphi^{\kappa'}, \varphi[x/u^{\kappa}_{m}]^{\kappa.1}, \Delta}{\Gamma \vdash \exists x\varphi^{\kappa}, \Delta}\,\mathsf{R}\exists$$

Figure 2.9: The δ- and γ-rules of LK^{v}. The number m is the formula number of the principal formula, and $\kappa.1$ denotes the concatenation of κ and 1. The operator $'$ is defined on page 22.

The γ-rules introduce instantiation variables of the form u^{κ}_{m} in which m and κ are the formula number and the copy history of the principal formula. The last element of the copy history of the contraction copy of the principal formula is increased by one, thus distinguishing it from the principal formula. The copy history of the other active formula is extended with the number 1. As a result, γ-inferences whose principal formulae have identical indices introduce identical instantiation variables.

2.30 Definition (Set of LK^{v}-skeletons) *The* set of LK^{v}-skeletons *is defined inductively with relation to the* LK^{v}*-rules as follows.*

- *If $\Gamma \vdash \Delta$ is an indexed sequent in which all copy histories are equal to 1 and no two subformulae of indexed formulae in $\Gamma \cup \Delta$ have identical formula numbers, then $\Gamma \vdash \Delta$ is an LK^{v}-skeleton.*
- *The induction steps are defined like in Definition 2.14.*

If a formula φ is part of the context in an inference in a skeleton, then all occurrences of φ in that inference are copies of the same *source*. The source is either a formula in the root sequent, or an active formula in an inference farther below on the same branch. In a branching inference, the context is copied into both branches. As a result, formula occurrences with identical source may occur in different branches of a skeleton.

2.31 Definition (Source Identical) *Occurrences of formulae in an LK^{v}-skeleton π are source identical according to the following conditions.*

- *If φ is an extra formula in an inference θ in π, then all occurrences of φ in θ are source identical.*
- *If two source identical occurrences of a formula are principal in separate inferences in π, then equal active formulae in the respective inferences are source identical.*

2.32 Example *In the LK^{v}-skeleton*

$$\dfrac{\dfrac{\mathsf{s}_4 : \quad \forall x P x^2, P u^{1.1} \vdash P a^1}{\mathsf{s}_2 : \quad \forall x P x^1 \vdash P a^1} \qquad \dfrac{\mathsf{s}_5 : \quad \forall x P x^2, P u^{1.1} \vdash P b^1}{\mathsf{s}_3 : \quad \forall x P x^1 \vdash P b^1}}{\mathsf{s}_1 : \quad \forall \underset{1}{x} P \underset{2}{x}^1 \vdash P \underset{4}{a} \wedge \underset{3}{} P \underset{5}{b}^1}$$

u denotes the instantiation variable u^1_1. The occurrences of $\forall x P x^1$ in the sequents s_1, s_2 and s_3 are source identical, the occurrences of $\forall x P x^2$ in s_4 and s_5 are source identical, the occurrences of $P u^{1.1}$ in s_4 and s_5 are source identical, the occurrences of $P a^1$ in s_2 and s_4 are source identical, and the occurrences of $P b^1$ in s_3 and s_5 are source identical. No other formula occurrences in the skeleton are source identical.

The following proposition is needed in later chapters.

2.33 Proposition *Indexed formulae occurring in an LK^{v}-skeleton have identical indices if and only if they are source identical.*

The proof is simple and will be found in [3]. As a consequence of this proposition, different occurrences of the same γ-formula in a skeleton have identical indices, and thus introduce identical instantiation variables when expanded.

2.34 Definition *Let π be a skeleton and l some leaf sequent in π. A* connection *is a subsequent of l of the form*

$$P\vec{t} \vdash P\vec{s}$$

in which $P\vec{t}$ and $P\vec{s}$ are atomic formulae with identical predicate symbols. A connection set *is a set of connections. A connection set is* spanning *for a skeleton if it contains exactly one connection from each of the leaf sequents of the skeleton.*

2.35 Definition *For each connection $c = P(t_1, \ldots, t_n) \vdash P(s_1, \ldots, s_n)$ we define a set of* primary equations*, denoted $\mathsf{Prim}(c)$, as follows.*

$$\mathsf{Prim}(c) := \{t_i \approx s_i \,|\, 1 \leq i \leq n\}$$

For a connection set C the set of primary equations is defined as

$$\mathsf{Prim}(C) := \bigcup_{c \in C} \mathsf{Prim}(c)$$

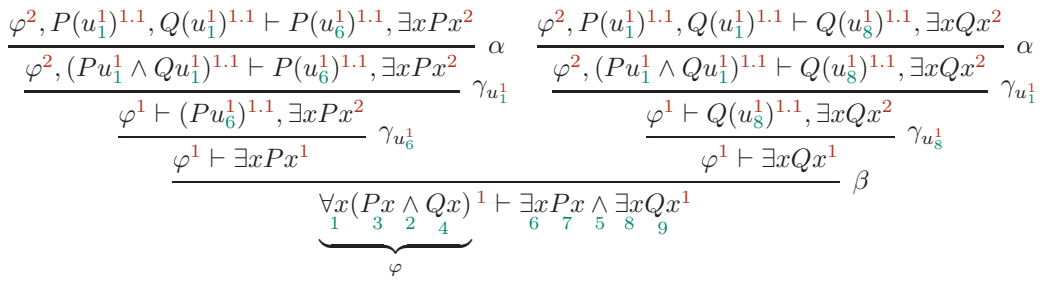
2.36 Definition *A substitution is* closing *for an LK^{v}-skeleton π if it satisfies the set of primary equations generated for some spanning set of connections for π. A skeleton π is* closable *if there is some closing substitution for it. This is the* closure condition *of LK^{v}.*

2.37 Definition (LK^{v}-proof) *A* proof *of a sequent $\Gamma \vdash \Delta$ in the calculus LK^{v} is a tuple $\langle \pi, C, \sigma \rangle$ such that π is a skeleton with $\Gamma \vdash \Delta$ as its root sequent, C is a spanning set of connections for π and σ is a substitution such that $\sigma \models \mathsf{Prim}(C)$.*

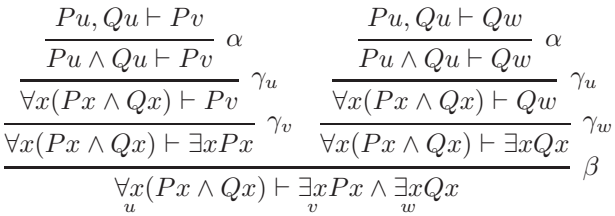
2.38 Example *Figure 2.10 shows two versions of an LK^{v}-skeleton for the sequent $\forall x(Px \wedge Qx) \vdash \exists x Px \wedge \exists x Qx$. In skeleton **(a)** full syntax is used in order to illustrate the copy history manipulation of the γ-inferences. Skeleton **(b)** is written in simplified syntax, leaving out unimportant details. Throughout the rest of the thesis we use mostly the simplified syntax in order to increase readability. The skeleton is closable, since the substitution $\{u/v, w/v\}$ ($\{u_1^1/u_6^1, u_8^1/u_6^1\}$ in full syntax) solves the primary equations for the spanning connection set $\{Pu \vdash Pv, Qu \vdash Qw\}$.*

Figure 2.11 displays two syntax versions of an LK^{v}-skeleton corresponding to the last LK-derivation in Example 2.19. The skeleton is not closable, since any spanning connection set results in an unsatisfiable set of primary equations.

2.39 Proposition *The calculus LK^{v} is sound and complete.*



(a)



(b)

Figure 2.10: Two versions of the same LK^{v}-skeleton. In **(a)** full syntax is used for implicit copies and indices, as opposed to the simplified syntax used in **(b)**. The inference γ_u illustrates how the indices impose the variable sharing property. We mostly use the simplified syntax throughout the rest of this thesis.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi^3, P(u_1^2, f_2u_1^2)^{2.1}, P(u_1^1, f_2u_1^1)^{1.1} \vdash \psi^3, P(f_5u_4^2, u_4^2)^{2.1}, P(f_5u_4^1, u_4^1)^{1.1}}{\varphi^3, P(u_1^2, f_2u_1^2)^{2.1}, P(u_1^1, f_2u_1^1)^{1.1} \vdash \psi^3, (\forall y P y u_4^2)^{2.1}, P(f_5u_4^1, u_4^1)^{1.1}}\,\delta_{f_5u_4^2}}{\varphi^3, P(u_1^2, f_2u_1^2)^{2.1}, P(u_1^1, f_2u_1^1)^{1.1} \vdash \psi^2, P(f_5u_4^1, u_4^1)^{1.1}}\,\gamma_{u_4^2}}{\varphi^3, (\exists y P u_1^2 y)^{2.1}, P(u_1^1, f_2u_1^1)^{1.1} \vdash \psi^2, P(f_5u_4^1, u_4^1)^{1.1}}\,\delta_{f_2u_1^2}}{\varphi^2, P(u_1^1, f_2u_1^1)^{1.1} \vdash \psi^2, P(f_5u_4^1, u_4^1)^{1.1}}\,\gamma_{u_1^2}}{\varphi^2, P(u_1^1, f_2u_1^1)^{1.1} \vdash \psi^2, (\forall y P y u_4^1)^{1.1}}\,\delta_{f_5u_4^1}}{\varphi^2, P(u_1^1, f_2u_1^1)^{1.1} \vdash \psi^1}\,\gamma_{u_4^1}}{\varphi^2, (\exists y P u_1^1 y)^{1.1} \vdash \psi^1}\,\delta_{f_2u_1^1}}{\underbrace{\forall x \exists y P x y}_{\varphi}{}^{\overset{1}{}\ \overset{2}{}\ \overset{3}{}} \vdash \underbrace{\exists x \forall y P y x}_{\psi}{}^{\overset{4}{}\ \overset{5}{}\ \overset{6}{}}}\,\gamma_{u_1^1}$$

(a)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi, Pwfw, Pufu \vdash \psi, Pgzz, Pgvv}{\varphi, Pwfw, Pufu \vdash \psi, \forall y Pyz, Pgvv}\,\delta_{gz}}{\varphi, Pwfw, Pufu \vdash \psi, Pgvv}\,\gamma_z}{\varphi, \exists y Pwy, Pufu \vdash \psi, Pgvv}\,\delta_{fw}}{\varphi, Pufu \vdash \psi, Pgvv}\,\gamma_w}{\varphi, Pufu \vdash \psi, \forall y Pyv}\,\delta_{gv}}{\varphi, Pufu \vdash \psi}\,\gamma_v}{\varphi, \exists y Puy \vdash \psi}\,\delta_{fu}}{\underbrace{\forall x \exists y P x y}_{\varphi} \vdash \underbrace{\exists x \forall y P y x}_{\psi}}\,\gamma_u$$

(b)

Figure 2.11: Two versions of a non-closable LK^{v}-skeleton; full syntax in **(a)** and simplified syntax in **(b)**. The symbols in **(b)** correspond to the ones in **(a)** as follows: $u = u_1^1$, $v = u_4^1$, $w = u_1^2$, $z = u_4^2$, $f = f_2$ and $g = f_5$. The γ-inferences illustrate how contraction copies are made distinct due to the index system.

The standard soundness proof for free variable tableaux [12, 21] is applicable the sequent calculus LK^{v} with some minor changes. One has to define semantics for formulae with instantiation terms and validity of sequents containing such formulae. The proof goes by contraposition; one shows that any skeleton having a falsifiable sequents as root has at least one open branch. As an intermediate result, one has to show that all LK^{v}-rules preserve countermodels upwards. This property of LK^{v}-rules corresponds to preservation of $\forall$-satisfiability for tableaux-rules.

The completeness proof is similar to the one found in [31] and is done as a reductio argument based on countermodel existence, i.e. that any sequent unprovable in LK^{v} has a countermodel. In order to show countermodel existence, a limit object is created for an unprovable sequent by means of a proof search procedure equipped with a fair selection function. Since the selection function is fair, every α, β and δ-formula and infinitely many contraction copies of each γ-formula are expanded on every branch in the limit object. The next step is to *ground* the limit object by a ground substitution. The countermodel existence proof can then be done constructively in the same way as in the completeness proof for LK. Care must be taken to ensure that the grounding substitution meets the satisfiability condition for universally quantified formulae; this property is, however, easily obtained.

Chapter 3

Incremental Proof Search

A naive proof search procedure for the calculus LK^{v} is one that alternates between skeleton expansion and closure check until the skeleton is closable. The closure check — in itself an NP-complete operation[1] — is performed after each expansion step, and the overall efficiency of such a proof search procedure is appalling. One way to address this inefficiency problem is to expand the skeleton within some limit, e.g. the maximum number of γ-inferences allowed on a branch, and check for closability when the limit is reached. If the skeleton is not closable, the proof search is continued with an increased limit. With such an approach we would still have to cope with an intractable closure check. In addition we might have to perform unnecessary expansion steps in cases where the skeleton becomes closable before the limit is reached.

An *incremental proof search procedure*, on the other hand, associates with each sequent in a skeleton a *syntactic constraint*. The constraint for a sequent s represents *all* closing substitutions for the part of the skeleton having s as root sequent. Each time a skeleton is expanded, the relevant constraints are updated in order to capture new closing substitutions due to the expansion. Closing substitutions are defined in terms of satisfiability of spanning sets of connections, and new closing substitutions may arise when new connections are introduced in leaf sequents. Since branching LK^{v}-rules are context sharing, all existing connections are copied into *both* branches. A closing substitution for a skeleton will remain closing regardless of skeleton expansion.

The above properties are exploited in order to update constraints in an *incremental* way. Constraints for sequents are defined inductively following skeleton expansion and new connections resulting from each expansion step. Updates for non-leaf constraints are defined recursively, following the intrinsic structure of the skeleton, in such a way that unsatisfiable constraints are

[1] A proof is found in [18] on page 21.

discarded. Thus, the closure check is reduced to testing whether the constraint associated with the root sequent of a skeleton is satisfiable, which is a constant time operation. An incremental proof search procedure checks for closability after each expansion step, in fact with relation to each new connection resulting from an expansion, and therefore terminates as soon as a proof is found. The incremental closure framework was originally introduced by Giese in [17, 18]. We will in this chapter explicate his work and adapt it to the free variable sequent calculus LK^{v}. Before we dig into the details, we introduce some basic notions.

3.1 Preliminaries

3.1.1 Expansion Sequences

During a proof search for a sequent, the selection function of the chosen proof search procedure is repeatedly applied to its own output using the input sequent as the initial value. This generates a sequence of skeletons, starting with the input sequent.

3.1 Definition *An* LK^{v} expansion sequence *is a finite or infinite sequence* $\pi_0, \pi_1, \pi_2, \ldots$ *having the following properties.*

- *Each* π_k *is an* LK^{v} *skeleton.*
- *The* initial skeleton π_0 *contains exactly one sequent.*
- *Each* π_k, $k > 0$, *is obtained from* π_{k-1} *by* one *expansion step.*

In the following, π_k ranges over skeletons in expansion sequences unless otherwise is clearly stated. Figure 3.1 illustrates the general form of the skeletons in an expansion sequence. The symbol $\rightsquigarrow$ denotes skeleton expansion.

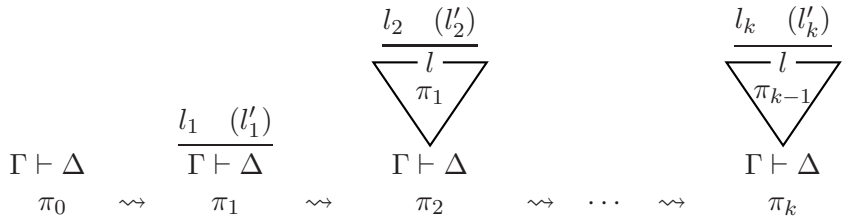


Figure 3.1: An LK^{v} expansion sequence. The initial skeleton π_0 contains a single sequent $\Gamma \vdash \Delta$. The skeleton π_k is obtained from π_{k-1} by expanding some leaf sequent l in π_{k-1}. If the expanding inference is non-branching (branching), the expansion step produces one (two) new leaf sequents l_k (l_k and l'_k) in π_k. The sequent l is leaf in π_{k-1}, but not in π_k.

3.2 Definition *Let s be a sequent in π_k. We define* $\mathsf{Conn}(s)$ *to be the* set of connections in s, *and* $\mathsf{Lvs}_k(s)$ *to be the* set of leaf sequents above s in π_k. *If s is a leaf sequent, then* $\mathsf{Lvs}_k(s) := \{s\}$. *The* set of leaf sequents in π_k, *denoted* $\mathsf{Lvs}(\pi_k)$, *is defined as*

$$\mathsf{Lvs}(\pi_k) := \mathsf{Lvs}_k(r),$$

where r is the root sequent of π_k.

3.1.2 New Connections

When a skeleton π_k is expanded, new leaf sequents are introduced in the subsequent skeleton π_{k+1}. The number of new leaf sequents is determined by the expanding inference, as illustrated in Figure 3.1.

3.3 Definition *For each skeleton π_k we define a* set of new leaf sequents, *denoted* $\mathsf{NewLvs}(\pi_k)$, *as follows.*

- *For an initial skeleton π_0 containing a single sequent r we define*

$$\mathsf{NewLvs}(\pi_0) := \{r\}.$$

- *For $k > 0$ we define*

$$\mathsf{NewLvs}(\pi_k) := \mathsf{Lvs}(\pi_k) \setminus \mathsf{Lvs}(\pi_{k-1}).$$

The leaf sequents in $\mathsf{NewLvs}(\pi_k)$ are referred to as *new to π_k*.

3.4 Definition *Let π_{k+1} be obtained from π_k by expansion of a formula in some leaf sequent l of π_k. Then, l is called the* expanded leaf sequent *of π_k.*

$$\frac{\Gamma, Pa \vdash \Delta \qquad \Gamma, Pb \vdash \Delta}{\Gamma, Pa \vee Pb \vdash \Delta}$$

Figure 3.2: A branching inference in which both active formulae (Pa and Pb) are atomic. Depending on the formulae in Δ, there might be connections in the premisses which *do not* occur in the conclusion of the inference.

Whenever a premiss l of an expanding inference has an *atomic* active formula, there might be connections in l which do not occur in the conclusion of the inference. These connections are *new* to the expanded skeleton. Since a sequent contains only a finite number of formulae, the set of new connections resulting from an expansion is finite. Due to the indexing system introduced

in the previous chapter, *all new connections resulting from an expansion step are distinct*. In a new connection $\varphi \vdash \psi$, either φ or ψ is an active formula in the expanding inference. Otherwise, the connection would also be present in the conclusion of the inference and thus *not* new. Since all subformulae of the principal formula in a branching inference have distinct formula numbers, the active formula in the left and right premiss have distinct indices, as illustrated in Figure 3.3. Thus, it is possible to define a set containing all new connections for a skeleton such that for each new connection there is a unique leaf sequent.

$$\frac{\underset{1}{Pa}^{\kappa}, Qb \vdash Pu, Qv \qquad \underset{2}{Pa}^{\kappa}, Qb \vdash Pu, Qv}{\underset{1}{Pa} \vee \underset{2}{Pa}^{\kappa}, Qb \vdash Pu, Qv}\ \beta$$

Figure 3.3: A branching LK^{v}-inference. Since the formulae are indexed, it is possible to distinguish the new connection in the left premiss from the one in the right. Without the indexing system, these connections would have been identical.

3.5 Definition *For each leaf sequent l in π_k we define a set of new connections, denoted $\mathsf{NewConn}_k(l)$, as follows. For an initial skeleton π_0 containing a single sequent r, we define*

$$\mathsf{NewConn}_0(r) := \mathsf{Conn}(r).$$

For each skeleton π_k, $k > 0$, let l' be the expanded leaf sequent of π_{k-1}.

- *If l is* new *to π_k, then*

$$\mathsf{NewConn}_k(l) := \mathsf{Conn}(l) \setminus \mathsf{Conn}(l').$$

- *Otherwise,*

$$\mathsf{NewConn}_k(l) := \emptyset.$$

For each connection c in $\mathsf{NewConn}_k(l)$ we define $\mathsf{Leaf}(c) := l$. The set of new connections for π_k, denoted $\mathsf{NewConn}(\pi_k)$, is defined as follows.

$$\mathsf{NewConn}(\pi_k) := \bigcup_{l \in \mathsf{NewLvs}(\pi_k)} \mathsf{NewConn}_k(l)$$

3.6 Example *Let l be the conclusion of the inference in Figure 3.3, and let l' and l'' be the left and right premiss, respectively. Assume that the inference is expanding the skeleton π_k into π_{k+1}. Then,*

$$\mathsf{NewConn}_{k+1}(l') = \{\underset{1}{Pa}^{\kappa} \vdash Pu\}$$

and

$$\mathsf{NewConn}_{k+1}(l'') = \{P\underset{2}{a}^{\kappa} \vdash Pu\}.$$

The set of new connections for the skeleton is

$$\mathsf{NewConn}(\pi_{k+1}) = \{P\underset{1}{a}^{\kappa} \vdash Pu,\ P\underset{2}{a}^{\kappa} \vdash Pu\}.$$

We will need the following definition in later proofs.

3.7 Definition *Let π_k be a skeleton in an LK^{v} expansion sequence, and let $C \subseteq \mathsf{NewConn}(\pi_k)$. For each leaf sequent l in π_k we define*

$$C|_l := \{c \in C \mid \mathsf{Leaf}(c) = l\}$$

to be the restriction *of C to l.*

3.8 Example *Let π_{k+1} and l' be as in Example 3.6. Then,*

$$\mathsf{NewConn}(\pi_{k+1})|_{l'} = \{P\underset{1}{a}^{\kappa} \vdash Pu\}$$

The following lemma is a direct consequence of Definition 3.7.

3.9 Lemma *Let π_k be a skeleton in an LK^{v} expansion sequence, and let l be a leaf sequent in π_k. Then, $\mathsf{NewConn}(\pi_k)|_l = \mathsf{NewConn}_k(l)$.*

3.1.3 Equation Sets and Unification

A substitution is closing for a skeleton π if it satisfies the set of primary equations generated from a spanning connection set for π. Thus, in order to determine whether the current skeleton state is closable, a proof search procedure will have to test satisfiability of equation sets. This is essentially a unification problem; does a substitution exists which unifies the left and right-hand side of each equation in the equation set. Unification is decidable in linear time [26, 28], but known linear time algorithms have poor performance in some cases. Martelli and Montanari present in [27] an algorithm which performs well in practice. The algorithm terminates for all input equation sets S. If it terminates with failure, then S is not satisfiable. Otherwise, it terminates with a unifier for S.

3.10 Definition (Solve-function) *Let $\mathfrak{S}$ be the set of all equation sets. We define the function $\mathsf{Solve} : \mathfrak{S} \to \mathfrak{S} \cup \{\bot\}$ as follows. Let S be an equation set. Use some efficient unification algorithm to check whether there exists a unifier for S.*

- *If a unifier for S exists, then*

$$\mathsf{Solve}(S) := S$$

- *Otherwise,*

$$\mathsf{Solve}(S) := \bot\ .$$

As the Solve-function indicates, we are merely interested in the *existence* of a unifier, not the unifier itself. In order to detect a proof, it is sufficient to know that a closing substitution exists; we do not have to apply the closing substitution to the skeleton. However, in the next chapter we present the calculus $\mathsf{LK}^{\mathsf{vs}}$ in which closing substitutions must have a certain property in addition to satisfying the set of equations generated from a spanning connection set. Since the set of unifiers for a satisfiable equation set is infinite, it is impossible to check whether the set of unifiers contains a *closing* unifier (i.e. which has the additional property) by testing all unifiers. It turns out that it is sufficient to check whether a *most general unifier* for an equation set has the additional closing property.

3.11 Definition *A substitution σ is a* most general unifier *(mgu) for an equation set S if it satisfies S and is more general than any other substitution which satisfies S.*

Most general unifiers are unique up to renaming of variables [12].

3.12 Definition *Equation sets are* equivalent *if they are satisfied by exactly the same substitutions.*

3.13 Definition *A set of equations S is* in solved form *if it satisfies the following conditions.*

- *S is of the form $\{u_1 \approx t_1, \ldots, u_n \approx t_n\}$, in which u_i is an instantiation variable and t_i is an instantiation term for $1 \leq i \leq n$.*
- *All the variables $u_1, \ldots, u_n$ are distinct, and none of them occur in any of the terms $t_1, \ldots, t_n$.*

3.14 Example *The equations sets*

$$\{u \approx v, w \approx gv\} \quad \textit{and} \quad \{\}$$

are in solved form. The equation sets

$$\{u \approx v, w \approx gu\} \quad \textit{and} \quad \{u \approx v, u \approx w\}$$

are not *in solved form. In the former set the variable u occurs in the left-hand side of the first equation* and *the right-hand side of the last equation. In the latter set the variable u occurs in the left-hand side of more than one equations.*

An equation set $S = \{u_1 \approx t_1, \ldots, u_n \approx t_n\}$ in solved form is a witness of its own satisfiability. Since all the left-hand side variables are distinct, S can be viewed as an equational representation of the substitution $\sigma = \{u_1/t_1, \ldots, u_n/t_n\}$. Every equation in S is solved by σ, since none of the variables $u_1, \ldots, u_n$ occur in any of the terms $t_1, \ldots, t_n$. For the same reason, σ is idempotent. Further, σ is a most general unifier for S.

Whenever an input equation set is satisfiable, the unification algorithm of Martelli and Montanari returns a *dag solved form*, which is a variant of the solved form defined above. In a dag solved form, we only require that a left-hand side variable u_i does not occur in right-hand sides t_j for $j \geq i$. A dag solved form can be transformed into an equivalent equation set in solved form, but the cost of this transformation is an exponential worst case complexity of the unification algorithm [23]. In the context of the next chapter, equation sets in solved form are conceptually easier to work with than dag solved forms, due to their tight relationship with most general unifiers. Since complexity analysis is beyond the scope of this thesis, we take the solved form approach to unifiers in cases where the actual unifier is needed.

3.15 Definition (MGU-function) *Let $\mathfrak{S}$ be the set of all equation sets, and $\tilde{\mathfrak{S}}$ be the set of all equation sets in solved form. We define the function $\mathsf{MGU} : \mathfrak{S} \to \tilde{\mathfrak{S}} \cup \{\bot\}$ as follows. Let S be an equation set. If S is satisfiable, the $\mathsf{MGU}(S)$ is an equation set in solved form equivalent to S. Otherwise, $\mathsf{MGU}(S)$ is $\bot$.*

3.16 Example *Let $S = \{fa \approx fu, v \approx u\}$. Then*

$$\mathsf{MGU}(S) = \{u \approx a, v \approx a\}$$

and

$$\mathsf{MGU}(S \cup \{b \approx v\}) = \bot.$$

3.2 Constraints

The closure check is an important part of a proof search procedure because it regulates the termination of the proof search. If the current skeleton is closable the search terminates with success. Otherwise, skeleton expansion continues until the skeleton reaches a closable state. Recall from Section 2.2.2 that the closure condition for LK^{v}-skeletons is defined in terms of unifiable spanning connection sets. In order to ascertain that a skeleton is *not* closable, one has to check every spanning connection set for unifiability. A naive approach is to calculate all spanning connection sets each time the closure check is performed. Since the number of possible combinations of connections

grows exponentially with skeleton expansion, such a global closure check is not feasible. The spanning connection sets for a skeleton π_k are also spanning for all subsequent skeletons in the expansion sequence. This is due to the context sharing rules of LK^{v}. We exploit this monotonicity in order to define constraints inductively following the expansion sequence.

In Section 3.2.1 constraint syntax is presented along with the satisfiability relation relating constraints and substitutions. In Section 3.2.2 we define global constraints and prove that they are correct. In Section 3.2.3 we present inductively defined constraints and define how they can be updated incrementally. In Section 3.2.4 we show the correctness of incremental constraints, and in Section 3.2.5 we present a way of preventing generation of redundant constraints. In Section 3.2.6 we present an example showing that in some cases incremental constraints are less redundant than global constraints.

3.2.1 Constraint Language

In this section the constraint syntax is presented and we define a satisfiability relation between constraints and substitutions.

3.17 Definition *The set of* atomic constraints *is the least set satisfying the following conditions.*

- *The symbol* $\bot$ *is an atomic constraint.*
- *A finite equation set is an atomic constraint.*

3.18 Example *The following are atomic constraints:*

$$\bot, \{\}, \{gu \approx ga,\ v \approx b\}, \{u \approx fffgv\}, \{u \approx v\}.$$

3.19 Definition *A* constraint *is a finite set of atomic constraints.*

3.20 Example *The following are constraints:*

$$\{\}, \{\{\}\}, \{\bot\}, \{\{u \approx a,\ v \approx b\},\ \{u \approx fffgv\}\}.$$

Atomic constraints are *conjunctive* and constraints are *disjunctive*. This is reflected by the satisfiability relation for constraints.

3.21 Definition (Satisfiability Relation) *Let* σ *be a substitution for instantiation terms,* μ *be an atomic constraint and* χ *a constraint.*

- σ *satisfies* μ*, denoted* $\sigma \models \mu$*, if and only if* $\mu \neq \bot$ *and* σ *solves every equation in* μ*.*

- *σ satisfies χ, denoted $\sigma \models \chi$, if and only if σ satisfies some atomic constraint in χ.*

The symbol $\bot$ is referred to as the *unsatisfiable atomic constraint*.

3.22 Example *The substitution $\{u/fa, v/a, w/gb\}$ satisfies the constraints $\{gu \approx gfv, fw \approx fgb\}$ and $\{\{\}\}$. The latter is a* trivial *constraint satisfied by any substitution. No substitution satisfies the constraints $\{\}$ or $\{\bot\}$.*

The atomic constraint $\{\}$, satisfiable by all substitutions, is syntactically equal to the unsatisfiable constraint $\{\}$. In the following there will, however, be no risk of mixing the two, since atomic constraints never occur outside constraints. Recall from page 20 that $\mathcal{I}$ is the set of all substitutions for instantiation terms.

3.23 Definition *Let χ be a constraint. The* satisfiability set *for χ, denoted $\mathsf{Sat}(\chi)$, is defined as*

$$\mathsf{Sat}(\chi) := \{\sigma \in \mathcal{I} \mid \sigma \models \chi\}.$$

Since substitutions are defined as total functions, the satisfiability set for any satisfiable constraint is infinite. Thus, constraints are finite objects representing possibly infinite sets of substitutions. The following lemma is a consequence of the satisfiability relation for constraints.

3.24 Lemma *Let χ_1 and χ_2 be constraints. Then,* $\mathsf{Sat}(\chi_1 \cup \chi_2) = \mathsf{Sat}(\chi_1) \cup \mathsf{Sat}(\chi_2)$.

3.25 Definition (Merging) *The merging operator $\otimes$ is defined for atomic constraints as follows. Let μ_1 and μ_2 be atomic constraints.*

- *If $\mu_1 = \bot$ or $\mu_2 = \bot$, then*

$$\mu_1 \otimes \mu_2 := \bot.$$

- *Otherwise,*

$$\mu_1 \otimes \mu_2 := \mathsf{Solve}(\mu_1 \cup \mu_2).$$

We extend the merging operator to constraints in the following way. Let χ_1 and χ_2 be constraints.

$$\chi_1 \otimes \chi_2 := \{\mu_1 \otimes \mu_2 \mid \mu_1 \in \chi_1 \text{ and } \mu_2 \in \chi_2\}$$

3.26 Example *Let $\chi_1 = \{\{u \approx v\}, \{u \approx a\}\}$, $\chi_2 = \{\{u \approx b\}\}$ and $\chi_3 = \{\}$ be constraints. Then,*

$$\chi_1 \otimes \chi_2 = \{\{u \approx v, u \approx b\}, \bot\}$$

and

$$\chi_1 \otimes \chi_3 = \{\}.$$

It is obvious the the property of being an (atomic) constraint is preserved by the merging operator. The following lemma is needed in later proofs.

3.27 Lemma $\mathsf{Sat}(\chi_1 \otimes \chi_2) = \mathsf{Sat}(\chi_1) \cap \mathsf{Sat}(\chi_2)$ *for all constraints* χ_1 *and* χ_2.

Proof Let σ be some substitution. It is sufficient to show that σ satisfies $\chi_1 \otimes \chi_2$ if and only if σ satisfies χ_1 and χ_2.

$$
\begin{aligned}
\sigma \models \chi_1 \otimes \chi_2 &\Leftrightarrow \sigma \models \{\mu_1 \otimes \mu_2 \mid \mu_1 \in \chi_1,\ \mu_2 \in \chi_2\} && \text{(by Def. 3.25)}\\
&\Leftrightarrow \sigma \models \mu_1 \otimes \mu_2 \text{ for some } \mu_1 \in \chi_1,\ \mu_2 \in \chi_2 && \text{(by Def. 3.21)}\\
&\Leftrightarrow \sigma \models \mathsf{Solve}(\mu_1 \cup \mu_2) \text{ for some } \mu_1 \in \chi_1,\ \mu_2 \in \chi_2 && \text{(by Def. 3.25)}\\
&\Leftrightarrow \sigma \models \mu_1 \cup \mu_2 \text{ for some } \mu_1 \in \chi_1,\ \mu_2 \in \chi_2 && \text{(by Def. 3.10)}\\
&\Leftrightarrow \sigma \models \mu_1 \text{ and } \sigma \models \mu_2 \text{ for some } \mu_1 \in \chi_1,\ \mu_2 \in \chi_2 &&\\
&\Leftrightarrow \sigma \models \chi_1 \text{ and } \sigma \models \chi_2 && \text{(by Def. 3.21)}
\end{aligned}
$$

■

3.2.2 Global Constraints

A naive closure check for LK^{v}-skeletons computes all possible spanning connection sets and tests each of them for unifiability. This process is captured in a global constraint. A constraint is defined for each leaf sequent of the skeleton, in which each unifiable connection is represented by an atomic constraint. Then, the constraint for the whole skeleton is the result of merging leaf sequent constraints. Since the merging operator tests for satisfiability, the resulting constraint contains only satisfiable atomic constraints, each of which corresponds to a unifiable spanning connection set for the skeleton.

In spite of being computationally inefficient, global constraint are included for two reasons: (1) The correctness proof for global constraints facilitates the correctness proof for incremental constraints, defined later in this chapter. (2) Although we show that global and incremental constraints are equivalent, i.e. that they are satisfied by exactly the same substitutions, there is a non-trivial syntactical difference between them. Incremental constraints are less redundant than global constraints.

3.28 Definition *The* Solve*-function is extended to connections as follows. For each connection* c *we define*

$$\mathsf{Solve}(c) := \mathsf{Solve}(\mathsf{Prim}(c)).$$

3.29 Definition (Global Constraints) *For each sequent* s *in a skeleton* π_k *we define a* global constraint*, denoted* $\mathsf{GC}_k(s)$*, as follows.*

- *If s is a leaf sequent, then*

$$\mathsf{GC}_k(s) := \{\mathsf{Solve}(c) \mid c \in \mathsf{Conn}(s)\}.$$

- *Otherwise, let* $\mathsf{Lvs}_k(s) = \{l_0, \ldots, l_n\}$. *Then,*

$$\mathsf{GC}_k(s) := (\mathsf{GC}_k(l_0) \otimes \mathsf{GC}_k(l_1)) \otimes \cdots \otimes \mathsf{GC}_k(l_n)$$

We define the global constraint for π_k, *denoted* $\mathsf{GC}(\pi_k)$, *as*

$$\mathsf{GC}(\pi_k) := \mathsf{GC}_k(r),$$

where r *is the root sequent of* π_k.

We now show that the global constraints are *correct*, i.e. that a substitution σ satisfies $\mathsf{GC}(\pi_k)$ if and only if it is closing for π_k.

3.30 Lemma *Let s be some sequent in a skeleton π_k. Then,*

$$\mathsf{Sat}(\mathsf{GC}_k(s)) = \bigcap_{l \in \mathsf{Lvs}_k(s)} \mathsf{Sat}(\mathsf{GC}_k(l)) \ .$$

Proof If s is leaf, the proof is trivial. Otherwise, assume without loss of generality that $\mathsf{Lvs}_k(s) = \{l_0, \ldots, l_n\}$. Then,

$$\begin{aligned} \mathsf{Sat}(\mathsf{GC}_k(s)) &= \mathsf{Sat}[(\mathsf{GC}_k(l_0) \otimes \mathsf{GC}_k(l_1)) \otimes \cdots \otimes \mathsf{GC}_k(l_n)] \\ &= \bigcap_{l \in \mathsf{Lvs}_k(s)} \mathsf{Sat}(\mathsf{GC}_k(l)) \ , \end{aligned}$$

in which the last step follows by applying Lemma 3.27 n times. ∎

3.31 Lemma *Let π_k be a skeleton and σ a substitution. Then, σ satisfies $\mathsf{GC}(\pi_k)$ if and only if σ is closing for π_k.*

Proof "If"-direction: Assume σ is closing for π_k. Then, there is a spanning set of connections C for π_k such that σ satisfies $\mathsf{Prim}(C)$. Pick an arbitrary connection c in C. Since $\mathsf{Prim}(c) \subseteq \mathsf{Prim}(C)$, σ satisfies $\mathsf{Prim}(c)$, and hence $\mathsf{Solve}(c)$, and thus σ satisfies $\mathsf{GC}_k(l)$ for a leaf l in π_k containing c. Since C is spanning for π_k, σ satisfies $\mathsf{GC}_k(l)$ for all leaf sequents l in π_k. Then, by Lemma 3.30 σ satisfies $\mathsf{GC}(\pi_k)$.

"Only if"-direction: Assume σ satisfies $\mathsf{GC}(\pi_k)$. Then, by Lemma 3.30 σ satisfies $\mathsf{GC}_k(l)$ for each leaf sequent l in π_k. By Definition 3.29 σ satisfies $\mathsf{Solve}(c)$, and hence $\mathsf{Prim}(c)$, for at least one connection c from each leaf sequent. Thus, σ is closing for π_k. ∎

3.32 Example *Let π_3 be the skeleton in Figure 3.4. The atomic constraints for the connections in the leaf sequents of π_3 are* $\mathsf{Solve}(Pv \vdash Pa) = \{v \approx a\}$, $\mathsf{Solve}(Pu \vdash Pa) = \{u \approx a\}$ *and* $\mathsf{Solve}(Pu \vdash Pb) = \{u \approx b\}$. *The global constraint for π_3 is*

$$\begin{aligned}\mathsf{GC}(\pi_3) &= \mathsf{GC}_3(\mathsf{s}_0)\\ &= \mathsf{GC}_3(\mathsf{s}_3) \otimes \mathsf{GC}_3(\mathsf{s}_{2''})\\ &= \{\{v{\approx}a\},\{u{\approx}a\}\} \otimes \{\{u{\approx}b\}\}\\ &= \{\mathsf{Solve}(\{v{\approx}a, u{\approx}b\}), \mathsf{Solve}(\{u{\approx}a, u{\approx}b\})\}\\ &= \{\{v{\approx}a, u{\approx}b\}, \bot\}\ ,\end{aligned}$$

thus the skeleton is closable.

$$\dfrac{\dfrac{\mathsf{s}_3 :\ \ \forall x Px, Pv, Pu \vdash Pa}{\mathsf{s}_{2'} :\ \ \forall x Px, Pu \vdash Pa} \qquad \mathsf{s}_{2''} :\ \ \forall x Px, Pu \vdash Pb}{\dfrac{\mathsf{s}_1 :\ \ \forall x Px, Pu \vdash Pa \wedge Pb}{\mathsf{s}_0 :\ \ \forall x Px \vdash Pa \wedge Pb}}$$

Figure 3.4: A closable skeleton.

3.2.3 Incremental Constraints

We will now define the constraint for a skeleton π_k inductively, following the expansion sequence $\pi_0, \pi_1, \cdots, \pi_k$. A proof search procedure takes *one sequent* as input, not an arbitrary skeleton. Thus, a proof search will always start at π_0 and expand one step at a time in order to reach the skeleton π_k. With an inductive constraint definition, skeleton expansion and constraint calculation can be done in parallel. The constraint for the skeleton is updated after each expansion step, and the closure check is reduced to checking whether this constraint is satisfiable.

The context sharing rules of LK^{v} ensures that a spanning connection set for a skeleton π_k is spanning also for all subsequent skeletons in the expansion sequence. This monotonicity property is exploited in order to update constraints *incrementally*. An overview of the symbols used for incremental constraints is found in Figure 3.5. The incremental proof search process is illustrated in Figure 3.6.

3.33 Definition *For each skeleton π_k in an LK^{v} expansion sequence, for each connection c_k^i in $\mathsf{NewConn}(\pi_k) = \{c_k^1, \ldots, c_k^{n_k}\}$, and each sequent s in π_k we define inductively an* incremental constraint, *denoted $\mathsf{C}_k^i(s)$, as follows.*

For each skeleton π_k:

n_k	number of new connections for π_k
c_k^i	i'th new connection for π_k
$\mathsf{C}(\pi_k)$	constraint for π_k

For each sequent s in π_k:

$\mathsf{INC}_k^i(s)$	increment set for s w.r.t. c_k^i
$\mathsf{C}_k^i(s)$	constraint for s up to and including $\mathsf{INC}_k^i(s)$

Figure 3.5: An overview of the symbols used in the inductive constraint definition (Definition 3.33). The subscript k refers to the skeleton π_k of the expansion sequence, and the superscript i refers to the i'th new connection for π_k.

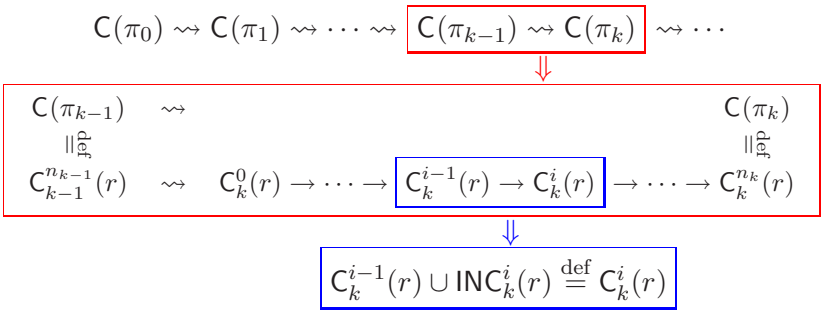


Figure 3.6: Constraints for skeletons in an LK^{v} expansion sequence, in which r is the initial sequent. After an expansion step from π_{k-1} to π_k, the constraint for π_k is updated incrementally with the increment set $\mathsf{INC}_k^i(r)$ for each connection c_k^i in $\mathsf{NewConn}(\pi_k) = \{c_k^1, \ldots, c_k^{n_k}\}$.

Initialization ($k = 0$ and $i = 0$): The skeleton π_0 contains a single sequent r. The constraint for r before any connections are taken into account, $\mathsf{C}_0^0(r)$, is defined as follows.

$$\mathsf{C}_0^0(r) := \emptyset$$

Inductive step 1 (increase of k and reset of i to 0): Assume that n_{k-1} is the number of new connections for π_{k-1}, and that $\mathsf{C}_{k-1}^{n_{k-1}}(s)$ is known for all sequents s in π_{k-1}. Let l' be the expanded leaf sequent of π_{k-1}. For all $k > 0$ and all sequents s in π_k we define $\mathsf{C}_k^0(s)$ as follows.

- *If s is* new *to π_k, then s is a premiss of the inference having l' as conclusion. We define*

$$\mathsf{C}_k^0(s) := \mathsf{C}_{k-1}^{n_{k-1}}(l').$$

- *Otherwise, s is also a sequent in π_{k-1} and we define*

$$\mathsf{C}_k^0(s) := \mathsf{C}_{k-1}^{n_{k-1}}(s).$$

Inductive step 2 (increase of i): Assume that $0 < i \leq n_k$, and that $\mathsf{C}_k^{i-1}(s)$ is known for all sequents s in π_k. Let B be the branch in π_k defined by $\mathsf{Leaf}(c_k^i)$. We define an increment set *w.r.t. the connection c_k^i, denoted $\mathsf{INC}_k^i(s)$, for each s in π_k in the following way.*

- *If s is* not *on B, then*

$$\mathsf{INC}_k^i(s) := \emptyset.$$

- *Otherwise, if $s = \mathsf{Leaf}(c_k^i)$, then*

$$\mathsf{INC}_k^i(s) := \{\mathsf{Solve}(c_k^i)\}$$

- *Otherwise, if s is the conclusion of a non-branching inference with premiss s', then*

$$\mathsf{INC}_k^i(s) := \mathsf{INC}_k^i(s').$$

- *Otherwise, s is the conclusion of a branching inference θ. Let s' and s'' be the premisses of θ, and assume s' is on B. We define*

$$\mathsf{INC}_k^i(s) := \mathsf{INC}_k^i(s') \otimes \mathsf{C}_k^{i-1}(s'').$$

We define

$$\mathsf{C}_k^i(s) := \mathsf{C}_k^{i-1}(s) \cup \mathsf{INC}_k^i(s).$$

Let r be the root sequent of π_k and n_k the number of new connections for π_k. The constraint for π_k, denoted $\mathsf{C}(\pi_k)$, is defined as

$$\mathsf{C}(\pi_k) := \mathsf{C}_k^{n_k}(r).$$

3.2.4 Correctness of Incremental Constraints

We now prove that our definition of incremental constraints is *correct*. This means two things:

1. If a substitution σ satisfies a constraint $\mathsf{C}_k^i(s)$, then σ is closing for the subskeleton of π_k with root s.
2. If σ is closing for the subskeleton of π_k with root s, then σ satisfies the constraint $\mathsf{C}_k^{n_k}(s)$.

This ensures that the generated constraints are only satisfied by closing substitutions for the current skeleton, and that *all* closing substitutions for the skeleton satisfy the constraint.

Remark. None of the proofs in this section are hard, but they are included for the sake of completeness and for illustration of how incremental constraints are calculated.

We first show correctness for leaf sequents. We show that after all new connections are taken into account, the global and the incremental constraint are equal for any leaf sequent in any skeleton. The correctness result then follows by the correctness proof for global constraints.

3.34 Lemma *Let π_k be a skeleton in an LK^{v} expansion sequence, and let $\mathsf{NewConn}(\pi_k) = \{c_k^1, \ldots, c_k^{n_k}\}$. Then, for each leaf sequent l in π_k, and for each $0 \leq i \leq n_k$:*

$$\mathsf{C}_k^i(l) = \mathsf{C}_k^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_k^1, \ldots, c_k^i\}|_l \,\}$$

Proof By induction on i. The base case, $i = 0$, is trivial.

Induction step: Pick some $0 \leq i < n_k$ and assume that the claim holds for $\mathsf{C}_k^i(l)$ (IH). We show that the claim holds for $\mathsf{C}_k^{i+1}(l)$. The proof depends on whether

(1) $\mathsf{Leaf}(c_k^{i+1}) = l$, or

(2) $\mathsf{Leaf}(c_k^{i+1}) \neq l$.

In case of (1), $\{c_k^1, \ldots, c_k^i\}|_l \cup \{c_k^{i+1}\} = \{c_k^1, \ldots, c_k^{i+1}\}|_l$, and thus

$$\begin{aligned}
\mathsf{C}_k^{i+1}(l) &= \mathsf{C}_k^i(l) \cup \mathsf{INC}_k^{i+1}(l) && \text{(by Def. 3.33)}\\
&= \mathsf{C}_k^i(l) \cup \{\, \mathsf{Solve}(c_k^{i+1}) \,\} && \text{(by Def. 3.33)}\\
&= \mathsf{C}_k^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_k^1, \ldots, c_k^i\}|_l \,\} \cup \{\, \mathsf{Solve}(c_k^{i+1}) \,\} && \text{(by IH)}\\
&= \mathsf{C}_k^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_k^1, \ldots, c_k^i\}|_l \cup \{c_k^{i+1}\} \,\}\\
&= \mathsf{C}_k^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_k^1, \ldots, c_k^{i+1}\}|_l \,\}\,.
\end{aligned}$$

In case of (2), $\{c_k^1, \ldots, c_k^i\}|_l = \{c_k^1, \ldots, c_k^{i+1}\}|_l$, and thus

$$\begin{aligned}
\mathsf{C}_k^{i+1}(l) &= \mathsf{C}_k^i(l) \cup \mathsf{INC}_k^{i+1}(l) && \text{(by Def. 3.33)}\\
&= \mathsf{C}_k^i(l) \cup \emptyset && \text{(by Def. 3.33)}\\
&= \mathsf{C}_k^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_k^1, \ldots, c_k^i\}|_l \,\} && \text{(by IH)}\\
&= \mathsf{C}_k^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_k^1, \ldots, c_k^{i+1}\}|_l \,\}\,,
\end{aligned}$$

which concludes the proof. ∎

3.35 Lemma *Let π_k be a skeleton in an LK^{v} expansion sequence, and let $\mathsf{NewConn}(\pi_k) = \{c_k^1, \ldots, c_k^{n_k}\}$. Then, for each leaf sequent l in π_k:*

$$\mathsf{C}_k^{n_k}(l) = \{\, \mathsf{Solve}(c) \mid c \in \mathsf{Conn}(l) \,\}$$

Proof By induction on k.

Base case ($k = 0$): π_0 contains a single sequent r, which is the only leaf sequent in π_0. Thus,

$$\begin{aligned}
\mathsf{C}_0^{n_0}(r) &= \mathsf{C}_0^0(r) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_0^1, \ldots, c_0^{n_0}\}|_r \,\} && \text{(by Lemma 3.34)}\\
&= \{\, \mathsf{Solve}(c) \mid c \in \{c_0^1, \ldots, c_0^{n_0}\}|_r \,\} && \text{(by Def. 3.33)}\\
&= \{\, \mathsf{Solve}(c) \mid c \in \mathsf{NewConn}(r) \,\} && \text{(by Lemma 3.9)}\\
&= \{\, \mathsf{Solve}(c) \mid c \in \mathsf{Conn}(r) \,\}\,. && \text{(by Def. 3.5)}
\end{aligned}$$

Induction step: Assume that the claim holds for π_k (IH). We show that the the claim holds for π_{k+1}. Let l be some leaf sequent in π_{k+1}. The proof depends on whether

(1) $l \in \mathsf{NewLvs}(\pi_{k+1})$, or

(2) $l \notin \mathsf{NewLvs}(\pi_{k+1})$.

In case of (1), let l' be the expanded leaf sequent of π_k. Then,

$$\begin{aligned}
\mathsf{C}_{k+1}^{n_{k+1}}(l) &= \mathsf{C}_{k+1}^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_{k+1}^1, \ldots, c_{k+1}^{n_{k+1}}\}|_l \,\} && \text{(by Lemma 3.34)}\\
&= \mathsf{C}_{k+1}^0(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \mathsf{NewConn}(l) \,\} && \text{(by Lemma 3.9)}\\
&= \mathsf{C}_k^{n_k}(l') \cup \{\, \mathsf{Solve}(c) \mid c \in \mathsf{NewConn}(l) \,\} && \text{(by Def. 3.33)}\\
&= \{\, \mathsf{Solve}(c) \mid c \in \mathsf{Conn}(l') \,\} \cup\\
&\quad \{\, \mathsf{Solve}(c) \mid c \in \mathsf{NewConn}(l) \,\} && \text{(by IH)}\\
&= \{\, \mathsf{Solve}(c) \mid c \in \mathsf{Conn}(l') \cup \mathsf{NewConn}(l) \,\}\\
&= \{\, \mathsf{Solve}(c) \mid c \in \mathsf{Conn}(l) \,\}\,. && \text{(by Def. 3.5)}
\end{aligned}$$

In case of (2), $\mathsf{NewConn}(l) = \emptyset$, and thus

$$\begin{aligned}
\mathsf{C}_{k+1}^{n_{k+1}}(l) &= \mathsf{C}_{k+1}^{0}(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \{c_{k+1}^1, \ldots, c_{k+1}^{n_{k+1}}\}|_l \,\} && \text{(by Lemma 3.34)} \\
&= \mathsf{C}_{k+1}^{0}(l) \cup \{\, \mathsf{Solve}(c) \mid c \in \mathsf{NewConn}(l) \,\} && \text{(by Lemma 3.9)} \\
&= \mathsf{C}_{k+1}^{0}(l) \cup \emptyset \\
&= \mathsf{C}_{k}^{n_k}(l) && \text{(by Def. 3.33)} \\
&= \{\, \mathsf{Solve}(c) \mid c \in \mathsf{Conn}(l) \,\}\,, && \text{(by IH)}
\end{aligned}$$

which concludes the proof. ∎

3.36 Lemma *Let π_k be a skeleton in an LK^{v} expansion sequence, and let n_k be the number of connections in $\mathsf{NewConn}(\pi_k)$. Then, for each leaf sequent l in π_k:*

$$\mathsf{Sat}\big(\mathsf{C}_k^{n_k}(l)\big) = \mathsf{Sat}(\mathsf{GC}_k(l))$$

Proof By Lemma 3.35 $\mathsf{C}_k^{n_k}(l) = \{\mathsf{Solve}(c) \mid c \in \mathsf{Conn}(l)\}$, which by Definition 3.29 equals $\mathsf{GC}_k(l)$. ∎

Having shown constraint correctness for leaf sequents, we now turn to non-leaf sequents. Incremental constraint correctness for non-leaf sequents cannot be shown by syntactical equality with global constraints, as was the case for leaf sequents. Instead we show that the global and incremental constraint for a sequent in a skeleton are equivalent, i.e. that their satisfiability sets are identical.

3.37 Lemma *Let π_k be a skeleton in an LK^{v} expansion sequence, and let $\mathsf{NewConn}(\pi_k) = \{c_k^1, \ldots, c_k^{n_k}\}$. Then, for each $0 \leq i \leq n_k$ and for each non-leaf sequent s in π_k:*

- *If s is the conclusion of a non-branching inference with premiss s', then*

$$\mathsf{Sat}\big(\mathsf{C}_k^i(s)\big) = \mathsf{Sat}\big(\mathsf{C}_k^i(s')\big)\,.$$

- *Otherwise, s is the conclusion of a branching inference θ. Let s' and s'' be the premisses of θ. Then,*

$$\mathsf{Sat}\big(\mathsf{C}_k^i(s)\big) = \mathsf{Sat}\big(\mathsf{C}_k^i(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big)\,.$$

Proof By induction on the pair $\langle k, i\rangle$. The base case, $\langle k, i\rangle = \langle 0, 0\rangle$, is trivial, since the only sequent in π_0 is a leaf sequent. For both induction steps, let s be the conclusion of an inference θ. We must distinguish between θ being a branching or non-branching inference. Since the two cases are

similar, we will throughout this proof show the branching case only. Let θ be of the form

$$\frac{s' \qquad s''}{s}\ \theta\ .$$

Induction step 1 (increase of k): Assume that the claim holds for $\langle k, n_k\rangle$. (IH). We show that the claim holds for $\langle k+1, 0\rangle$. Let l' be the expanded leaf sequent of π_k. The proof depends on whether

(1.1) $s = l'$, or

(1.2) $s \neq l'$.

In case of (1.1), s' and s'' are the leaf sequents *new* to π_{k+1}, and thus $\mathsf{C}^0_{k+1}(s') = \mathsf{C}^0_{k+1}(s'') = \mathsf{C}^0_{k+1}(s) = \mathsf{C}^{n_k}_k(s)$ by Definition 3.33. We get

$$\begin{aligned}\mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s)\big) &= \mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s)\big) \cap \mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s)\big)\\ &= \mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s'')\big)\ .\end{aligned}$$

In case of (1.2), both s' and s'' are sequents in π_k, and thus $\mathsf{C}^0_{k+1}(s') = \mathsf{C}^{n_k}_k(s')$ and $\mathsf{C}^0_{k+1}(s'') = \mathsf{C}^{n_k}_k(s'')$ by Definition 3.33. We get

$$\begin{aligned}\mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s)\big) &= \mathsf{Sat}(\mathsf{C}^{n_k}_k(s)) && \text{(by Def. 3.33)}\\ &= \mathsf{Sat}(\mathsf{C}^{n_k}_k(s')) \cap \mathsf{Sat}(\mathsf{C}^{n_k}_k(s'')) && \text{(by IH)}\\ &= \mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}^0_{k+1}(s'')\big)\ .\end{aligned}$$

Induction step 2 (increase of i): Pick some $0 \leq i < n_k$ and assume that the claim holds for $\langle k, i\rangle$ (IH). We show that the claim holds for $\langle k, i+1\rangle$. Let B be the branch in π_k defined by $\mathsf{Leaf}(c^{i+1}_k)$. The proof depends on whether

(2.1) s *is* on B, or

(2.2) s is *not* on B.

In case of (2.1), assume without loss of generality that s' is on B. Then, s''

is not on B, and thus (†) $\mathsf{C}_k^{i+1}(s'') = \mathsf{C}_k^i(s'')$ by Definition 3.33. We get

$$\begin{aligned}
\mathsf{Sat}\big(\mathsf{C}_k^{i+1}(s)\big) &= \mathsf{Sat}\big(\mathsf{C}_k^i(s) \cup \mathsf{INC}_k^{i+1}(s)\big) && \text{(by Def. 3.33)}\\
&= \mathsf{Sat}\big(\mathsf{C}_k^i(s)\big) \cup \mathsf{Sat}\big(\mathsf{INC}_k^{i+1}(s)\big) && \text{(by Lemma 3.24)}\\
&= \big[\mathsf{Sat}\big(\mathsf{C}_k^i(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big)\big] \cup \mathsf{Sat}\big(\mathsf{INC}_k^{i+1}(s)\big) && \text{(by IH)}\\
&= \big[\mathsf{Sat}\big(\mathsf{C}_k^i(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big)\big] \cup\\
&\quad \mathsf{Sat}\big(\mathsf{INC}_k^{i+1}(s') \otimes \mathsf{C}_k^i(s'')\big) && \text{(by Def. 3.33)}\\
&= \big[\mathsf{Sat}\big(\mathsf{C}_k^i(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big)\big] \cup\\
&\quad \big[\mathsf{Sat}\big(\mathsf{INC}_k^{i+1}(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big)\big] && \text{(by Lemma 3.27)}\\
&= \big[\mathsf{Sat}\big(\mathsf{C}_k^i(s')\big) \cup \mathsf{Sat}\big(\mathsf{INC}_k^{i+1}(s')\big)\big] \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big)\\
&= \mathsf{Sat}\big(\mathsf{C}_k^i(s') \cup \mathsf{INC}_k^{i+1}(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big) && \text{(by Lemma 3.27)}\\
&= \mathsf{Sat}\big(\mathsf{C}_k^{i+1}(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big) && \text{(by Def. 3.33)}\\
&= \mathsf{Sat}\big(\mathsf{C}_k^{i+1}(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^{i+1}(s'')\big)\ . && \text{(by †)}
\end{aligned}$$

In case of (2.2), neither s' nor s'' are on B. Thus, $\mathsf{C}_k^{i+1}(s') = \mathsf{C}_k^i(s')$ and $\mathsf{C}_k^{i+1}(s'') = \mathsf{C}_k^i(s'')$, and we get

$$\begin{aligned}
\mathsf{Sat}\big(\mathsf{C}_k^{i+1}(s)\big) &= \mathsf{Sat}\big(\mathsf{C}_k^i(s)\big) && \text{(by Def. 3.33)}\\
&= \mathsf{Sat}\big(\mathsf{C}_k^i(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^i(s'')\big) && \text{(by IH)}\\
&= \mathsf{Sat}\big(\mathsf{C}_k^{i+1}(s')\big) \cap \mathsf{Sat}\big(\mathsf{C}_k^{i+1}(s'')\big)\ , &&
\end{aligned}$$

which concludes the proof. ∎

The *depth* of a sequent is the number of inferences on the path to the farthest away leaf sequent above it, defined below.

3.38 Definition *Let π be a skeleton. We define for each sequent s in π the* depth *of s recursively as follows.*

- *If s is a leaf sequent, then* $\mathsf{Depth}(s) := 0$.
- *If s is the conclusion of a non-branching inference with premiss s', then* $\mathsf{Depth}(s) := \mathsf{Depth}(s') + 1$.
- *If s is the conclusion of a branching inference with premisses s' and s'', then* $\mathsf{Depth}(s) := \max\{\mathsf{Depth}(s'), \mathsf{Depth}(s'')\} + 1$.

3.39 Lemma *Let π_k be a skeleton in an* LK^{v} *expansion sequence, and let n_k be the number of connections in* $\mathsf{NewConn}(\pi_k)$. *Then, for each non-leaf sequent s in π_k:*

$$\mathsf{Sat}\big(\mathsf{C}_k^{n_k}(s)\big) = \mathsf{Sat}(\mathsf{GC}_k(s))$$

PROOF by induction on $\mathsf{Depth}(s)$. Let s be the conclusion of a branching inference θ. The proof depends on whether θ is branching or not. The two cases are similar, thus we will only show the branching case. Assume θ to be of the form

$$\frac{s' \qquad s''}{s}\ \theta\ .$$

Base case: $\mathsf{Depth}(s) = 1$, and s' and s'' are leaf sequents in π_k. We have $\mathsf{Lvs}_k(s) = \{s', s''\}$, and thus,

$$\begin{aligned}
\mathsf{Sat}(\mathsf{C}_k^{n_k}(s)) &= \mathsf{Sat}(\mathsf{C}_k^{n_k}(s')) \cap \mathsf{Sat}(\mathsf{C}_k^{n_k}(s'')) && \text{(by Lemma 3.37)}\\
&= \mathsf{Sat}(\mathsf{GC}_k(s')) \cap \mathsf{Sat}(\mathsf{GC}_k(s'')) && \text{(by Lemma 3.36)}\\
&= \mathsf{Sat}(\mathsf{GC}_k(s') \otimes \mathsf{GC}_k(s'')) && \text{(by Lemma 3.27)}\\
&= \mathsf{Sat}(\mathsf{GC}_k(s))\ . && \text{(by Def. 3.29)}
\end{aligned}$$

Induction step: Assume that the claim holds for all non-leaf sequents in π_k having depth d (IH). We show that the claim holds for sequents s having depth $d+1$. We get

$$\begin{aligned}
\mathsf{Sat}(\mathsf{C}_k^{n_k}(s)) &= \mathsf{Sat}(\mathsf{C}_k^{n_k}(s')) \cap \mathsf{Sat}(\mathsf{C}_k^{n_k}(s'')) && \text{(by Lemma 3.37)}\\
&= \mathsf{Sat}(\mathsf{GC}_k(s')) \cap \mathsf{Sat}(\mathsf{GC}_k(s'')) && \text{(by IH)}\\
&= \bigcap_{l \in \mathsf{Lvs}_k(s')} \mathsf{Sat}(\mathsf{GC}_k(l)) \ \cap \bigcap_{l \in \mathsf{Lvs}_k(s'')} \mathsf{Sat}(\mathsf{GC}_k(l)) && \text{(by Lemma 3.30)}\\
&= \bigcap_{l \in \mathsf{Lvs}_k(s)} \mathsf{Sat}(\mathsf{GC}_k(l))\\
&= \mathsf{Sat}(\mathsf{GC}_k(s))\ , && \text{(by Lemma 3.30)}
\end{aligned}$$

which concludes the proof. ∎

3.40 THEOREM *Let π_k be a skeleton in an LK^{v} expansion sequence. Then,* $\mathsf{Sat}(\mathsf{C}(\pi_k)) = \mathsf{Sat}(\mathsf{GC}(\pi_k))$.

PROOF Let r be the root sequent of π_k, and let n_k be the number of connections in $\mathsf{NewConn}(\pi_k)$. Then,

$$\begin{aligned}
\mathsf{Sat}(\mathsf{C}(\pi_k)) &= \mathsf{Sat}(\mathsf{C}_k^{n_k}(r)) && \text{(by Def. 3.33)}\\
&= \mathsf{Sat}(\mathsf{GC}_k(r)) && (\dagger)\\
&= \mathsf{Sat}(\mathsf{GC}(\pi_k))\ , && \text{(by Def. 3.29)}
\end{aligned}$$

in which (†) follows by Lemma 3.36 if r is a leaf sequent, otherwise, it follows by Lemma 3.39. ∎

Finally, we formulate this corollary, which follows easily by Theorem 3.40 and Lemma 3.31.

3.41 Corollary *Let π_k be a skeleton in an LK^{v} expansion sequence, and let σ be a substitution in $\mathcal{I}$. Then, $\sigma \in \mathsf{Sat}(\mathsf{C}(\pi_k))$ if and only if σ is closing for π_k.*

As a result of the above corollary, both correctness properties stated at the beginning of this section holds:

1. If a substitution σ satisfies a constraint $\mathsf{C}^i_k(s)$, then σ is closing for the subskeleton of π_k with root s.
2. If σ is closing for the subskeleton of π_k with root s, then σ satisfies the constraint $\mathsf{C}^{n_k}_k(s)$.

The first property implies that we can terminate a proof search as soon as the increment set for the root sequent is satisfiable. Thus, the closure check is done not only after each skeleton expansion, but also for each new connections resulting from an expansion. Incremental constraints allow for a more fine grained closure check. The second property ensures that if we have taken into account *all* new connections due to an expansion step and the constraint for the root sequent is unsatisfiable, then the current skeleton is *not* closable and more expansions are needed in order to find a proof.

3.2.5 Subsumption

The merging of two constraints is essentially a cross product operation; each atomic constraint in the first is merged with each atomic constraint in the second. When a new connection in a leaf sequent l is taken into account, this operation is performed at each β-inference on the path from l to the root sequent. The sizes of the existing constraints, i.e. the number of atomic constraints they contain, affect the overall efficiency of the merging operations. Thus, it is desirable to keep constraints small. Constraints generated for new connections do not necessarily have to represent new closing substitutions for a skeleton. If we have a constraint $\chi = \{\{u \approx a\}\}$ and an atomic constraint $\mu = \{u \approx a, v \approx b\}$, then the constraints χ and $\chi \cup \{\mu\}$ are satisfied by exactly the same substitutions, due to the disjunctive constraint satisfiability.

3.42 Definition *An atomic constraint μ_1 is* subsumed by *an atomic constraint μ_2 if $\mathsf{Sat}(\{\mu_1\}) \subseteq \mathsf{Sat}(\{\mu_2\})$. An atomic constraint μ is subsumed by a constraint χ if $\mathsf{Sat}(\{\mu\}) \subseteq \mathsf{Sat}(\chi)$.*

As a result, we can refrain from propagating constraints for new connections if they are subsumed by the existing constraint for the leaf sequent. We redefine increment sets as follows.

3.43 Definition (Subsumption Checking) *If* $s = \mathsf{Leaf}(c_k^i)$, *then*

$$\mathsf{INC}_k^i(s) := \begin{cases} \emptyset, & \textit{if } \mathsf{Solve}(c_k^i) \textit{ is subsumed by } \mathsf{C}_k^{i-1}(s) \\ \{\mathsf{Solve}(c_k^i)\}, & \textit{otherwise.} \end{cases}$$

For all other cases the definition is like Definition 3.33.

It is also possible to adapt subsumption checking to non-leaf sequents. This can be done in two ways: (1) When updating an existing constraint with an increment set, we only add atomic constraints which are not subsumed by the existing constraint. This is called *forward subsumption*. (2) When adding an atomic constraint μ, we remove from the existing constraint all atomic constraints which are subsumed by μ. This is called *backward subsumption*. Experimental results in [18] indicate that while forward subsumption has a favorable impact on the average proof search time, the computational overhead caused by backward subsumption is greater than the benefits.

3.2.6 Example

One does not have to be an experienced logician to establish that the sequent

$$\forall x Px, \forall x Rx \vee \forall x Qx \vdash Pa, Qb, Ra \wedge Rb, Rc$$

is valid. Unfortunate selection of expansion formulae may however generate the skeleton π_5 shown in Figure 3.7. The subscripted numbers in the sequent labels indicate the order in which the skeleton is expanded. The skeleton π_5 is obtained by expanding the formula $\forall x Rx$ in the sequent $\mathsf{s}_{1'}$, introducing the new connection $c_5^1 = Rw \vdash Rc$ in the leaf sequent s_5. The incremental constraints for the sequents in π_5 before the new connection is taken into account are listed in Table 3.1. We now compute the increment sets with relation to the connection c_5^1. For the leaf sequent s_5 we get

$$\mathsf{INC}_5^1(\mathsf{s}_5) = \{\mathsf{Solve}(c_5^1)\} = \{\{w \approx c\}\}.$$

For the sequent $\mathsf{s}_{1'}$ we get

$$\mathsf{INC}_5^1(\mathsf{s}_{1'}) = \mathsf{INC}_5^1(\mathsf{s}_5).$$

For the root sequent s_0 we get

$$\begin{aligned} \mathsf{INC}_5^1(\mathsf{s}_0) &= \mathsf{INC}_5^1(\mathsf{s}_{1'}) \otimes \mathsf{C}_5^0(\mathsf{s}_{1''}) \\ &= \{\{w \approx c\}\} \otimes \{\{u \approx a\},\ \{v \approx b\}\} \\ &= \{\{w \approx c,\ u \approx a\},\ \{w \approx c,\ v \approx b\}\}\ , \end{aligned}$$

and thus the skeleton is closable.

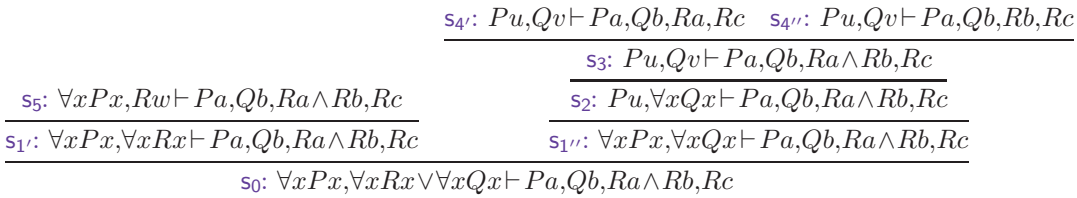


Figure 3.7: A closable skeleton π_5. The subscripted numbers in the sequent labels indicate the order in which the sequents were introduced.

s	$\mathsf{C}_5^0(s)$
s_0	$\{\}$
$\mathsf{s}_{1'}$	$\{\}$
$\mathsf{s}_{1''}$	$\{\{u \approx a\},\ \{v \approx b\}\}$
s_2	$\{\{u \approx a\},\ \{v \approx b\}\}$
s_3	$\{\{u \approx a\},\ \{v \approx b\}\}$
$\mathsf{s}_{4'}$	$\{\{u \approx a\},\ \{v \approx b\}\}$
$\mathsf{s}_{4''}$	$\{\{u \approx a\},\ \{v \approx b\}\}$
s_5	$\{\}$

Table 3.1: The incremental constraints for π_5 before any new connections are taken into account.

We now calculate the global constraints for π_5. The constraints for the leaf sequents are

$$\mathsf{GC}_5(\mathsf{s}_5) = \{\mathsf{Solve}(Rw \vdash Rc)\} = \{\{w \approx c\}\} \ ,$$
$$\mathsf{GC}_5(\mathsf{s}_{4'}) = \{\mathsf{Solve}(Pu \vdash Pa),\ \mathsf{Solve}(Qv \vdash Qb)\} = \{\{u \approx a\},\ \{v \approx b\}\} \text{ and}$$
$$\mathsf{GC}_5(\mathsf{s}_{4''}) = \mathsf{GC}_5(\mathsf{s}_{4'}) \ .$$

The global constraint for the skeleton is

$$\begin{aligned}\mathsf{GC}(\pi_5) &= \mathsf{GC}_5(\mathsf{s}_0) \\ &= (\mathsf{GC}_5(\mathsf{s}_5) \otimes \mathsf{GC}_5(\mathsf{s}_{4'})) \otimes \mathsf{GC}_5(\mathsf{s}_{4''}) \\ &= \{\{w \approx c, u \approx a\},\ \{w \approx c, v \approx b\}\} \otimes \mathsf{GC}_5(\mathsf{s}_{4''}) \\ &= \{\{w \approx c, u \approx a\},\ \{w \approx c, v \approx b\},\ \{w \approx c, u \approx a, v \approx b\}\} \ .\end{aligned}$$

Although the global and incremental constraint for π_5 are equivalent, they are not syntactically equal. The atomic constraint making them different, $\{w \approx c, u \approx a, v \approx b\}$, is in fact subsumed by the incremental constraint for π_5. The phenomenon arises because the *incremental* constraint for s_3 is *copied* into both new leaf sequents $\mathsf{s}_{4'}$ and $\mathsf{s}_{4''}$ in the expansion step from π_3 to π_4. On the contrary, the identical global constraints for $\mathsf{s}_{4'}$ and $\mathsf{s}_{4''}$ are *merged* when the *global* constraint is calculated. In the general case it is easily shown that a constraint χ merged with itself is satisfied by exactly the same substitutions as χ. The examples shows that incremental constraints are in some cases less redundant than global constraints.

3.3 Concluding Remarks

In this chapter I have presented a proof search procedure in which the closure check is calculated in parallel with skeleton expansion. Partial spanning connection sets are represented as syntactic constraints. When a skeleton is expanded, the constraints are updated incrementally with new closing possibilities resulting from new connections in the leaf sequents. A satisfiability relation between constraints and substitutions was defined, and I showed that the incremental constraint definition is correct, i.e. that an incremental constraint for a skeleton π is satisfied by exactly the substitutions which are closing for π.

Chapter 4

Incremental Proof Search with Variable Splitting

The skeletons of the sequent calculus LK^{v} are invariant under order of rule application, meaning that all permutation variants of a skeleton have identical leaf sequents. This is achieved by letting the LK^{v}-rules generate variable sharing skeletons, in which there is a strong dependency among variable occurrences. Since identical instantiation variables can occur in different branches of a skeleton, the term universe cannot be restricted branchwise. However, in some cases separate occurrences of the same instantiation variable are *independent*, meaning that it is sound to instantiate them differently. The task of detecting exactly when variable occurrences are independent is called the variable splitting problem.

$$\dfrac{\dfrac{\forall xPx, Pu \vdash Pa \qquad \forall xPx, Pu \vdash Pb}{\forall xPx, Pu \vdash Pa \wedge Pb}\,\beta}{\forall xPx \vdash Pa \wedge Pb}\,\gamma_u$$

Figure 4.1: A non-closable LK^{v}-skeleton for the sequent $\forall xPx \vdash Pa \wedge Pb$.

The skeleton in Figure 4.1 is regulated by the rules of LK^{v}. It is not closable, since we cannot instantiate u with both a and b. The γ-formula $\forall xPx$ is present in both branches. By re-expanding it in one of the them, we introduce the extra instantiation variable needed in order to close the skeleton. In Figure 4.2, the occurrences of u are marked differently in the two branches. The variable u is split into u^1 and u^2, and the skeleton is closable without re-expanding the γ-formula.

In Section 4.1 I shall introduce the variable splitting sequent calculus $\mathsf{LK}^{\mathsf{vs}}$, which is an extension of the calculus LK^{v}. Skeletons of $\mathsf{LK}^{\mathsf{vs}}$ are variable

$$\dfrac{\dfrac{\forall xPx, Pu^1 \vdash Pa \qquad \forall xPx, Pu^2 \vdash Pb}{\forall xPx, Pu \vdash Pa \wedge Pb}\,\beta}{\forall xPx \vdash \underset{1}{Pa} \wedge \underset{2}{Pb}}\,\gamma_u$$

Figure 4.2: The splitting of u into u^1 and u^2 permits closure of the skeleton without re-expanding the γ-formula.

sharing and they contain indexed formulae, like LK^{v}-skeletons. In addition, the indexing system is utilized to label formula occurrences according to which branch they occur in. When spanning connection sets are generated, instantiation variables occurring in a formula are *colored* by means of the formula label. Variable instantiation is done at the level of colored variables. This permits substitutions to *split* variables, i.e. to instantiate different colorings of the same instantiation variable differently.

Care must be taken in order to avoid unsound variable instantiation. In $\mathsf{LK}^{\mathsf{vs}}$ this is handled as follows. If a variable u occurs in two separate branches of a skeleton π and a substitution instantiates them differently, then there must exist a permutation variant of π in which the γ-formula introducing u is present in both branches. For each pair of colorings of the same variable instantiated differently by the substitution, new requirements are imposed on the permutation variant. A substitution is closing for a skeleton only if there is a permutation variant meeting all of the requirements imposed by the substitution. Such a permutation variant π' has the property that it can be simulated in a variable pure sequent calculus, i.e. where each γ-inference introduces a distinct variable. The check for existence of such a permutation variant is referred to as a cycle check for the substitution.

The incremental constraint definition for $\mathsf{LK}^{\mathsf{vs}}$-skeletons must capture the additional requirements of the closure condition of $\mathsf{LK}^{\mathsf{vs}}$. We will take two different approaches to constraints for splitting skeletons. The first variant will be an extension of the constraint language used in the previous chapter. Atomic constraints are still defined as equation sets, and the cycle check is performed in the root sequent only. The second approach performs the cycle checking incrementally as constraints resulting from new connections are propagated towards the root sequent. In order to achieve this, the equation sets of atomic constraints are in solved form. This is due to the fact that a most general unifier is needed in order to perform the cycle check.

The calculus $\mathsf{LK}^{\mathsf{vs}}$ is, with a few exceptions, identical to the free variable sequent calculus with uniform variable splitting originally introduced by Waaler and Antonsen in [32]. The parts on which they differ are improvements and error corrections mostly due to Antonsen [3]. The subject of

variable splitting is not yet fully explored. Among the open questions is which is the weakest set of restrictions possible to put on closing substitutions without compromising consistency [2]. Hence, we will assume that the current version of the calculus is correct and use it as a basis for the design of an incremental proof search procedure with variable splitting.

4.1 The Variable Splitting Technique

In this section I shall introduce the terminology used in the variable splitting technique. In Section 4.1.1 a relation on indices is introduced, capturing subformula dependencies between indices in a skeleton. In Section 4.1.2 syntax for variable splitting is defined, and in Section 4.1.3 the variable splitting sequent calculus $\mathsf{LK}^{\mathsf{vs}}$ is introduced. In Section 4.1.4 I discuss some concepts relevant for the design of proof search procedures for $\mathsf{LK}^{\mathsf{vs}}$, and in Section 4.1.5 I present some examples of the introduced concepts.

4.1.1 Relations on Indices

Recall from Proposition 2.33 that formulae occurring in an LK^{v}-skeleton have identical indices if and only if they are source identical. By definition of source identicallity it follows that all source identical formula occurrences are equal. Thus, formula occurrences have identical indices if and only if they are equal. As a result, it is well-defined to refer to a particular *formula* in an LK^{v}-skeleton by its index. We say that an index is a θ-index or of principal type θ in a skeleton π if the associated formula is of principal type θ. Likewise, an index is principal in π if the associated formula is principal in some inference in π. In the following, indices are often denoted i (possibly subscripted).

4.1 Definition *Let π be a skeleton. The* immediate descendant relation for π*, denoted $\ll_\pi$, is a binary relation on the set of indices occurring in π such that $i_1 \ll_\pi i_2$ if and only if there is an inference in π having principal formula with index i_1 and active formula with index i_2. The transitive closure of $\ll_\pi$, denoted $\ll_\pi^+$, is referred to as the* descendant relation for π.

4.2 Example *Let π be the following skeleton.*

$$\dfrac{\dfrac{\forall x P x^{2}, P u^{1.1} \vdash P a^{1}}{\forall x P x^{1} \vdash P a^{1}}\,\gamma_u \qquad \dfrac{\dfrac{\forall x P x^{3}, P v^{2.1}, P u^{1.1} \vdash P b^{1}}{\forall x P x^{2}, P u^{1.1} \vdash P b^{1}}\,\gamma_v}{\forall x P x^{1} \vdash P b^{1}}\,\gamma_u}{\forall x P x^{1} \vdash P a \wedge P b^{1}}\,\beta$$

Then, $\ll_\pi = \{(\frac{1}{3}, \frac{1}{4}), (\frac{1}{3}, \frac{1}{5}), (\frac{1}{1}, \frac{2}{1}), (\frac{1}{1}, \frac{1.1}{1}), (\frac{2}{1}, \frac{3}{1}), (\frac{2}{1}, \frac{2.1}{1})\}$.

4.3 Lemma *Let* π *be an* LK^{v}*-skeleton. Then,* $\ll_\pi^+$ *is irreflexive.*

Proof Assume $\ll_\pi^+$ reflexive, i.e. $i \ll_\pi^+ i$ for some index i. Then, either $i \ll_\pi i$, or there are indices $i_1, \ldots, i_n$ in π such that $i \ll_\pi i_1 \ll_\pi \ldots \ll_\pi i_n \ll_\pi i$. In either case there must be formula occurrences in π with identical indices and different sources, by Definition 4.1. But this is not possible, by Proposition 2.33. ∎

An immediate descendant relation for a skeleton captures dependencies between inferences due to the intrinsic structure of formulae in the skeleton. It is not possible to expand a proper subformula (or an instance of a proper subformula in case of δ- and γ-formulae) of a formula φ before φ itself is expanded. This restricts the the set of possible permutation variants for a skeleton.

4.4 Definition *The following notions are defined relative to a skeleton* π*.*

- *An index* i_1 *is a* descendant *of an index* i_2*, and* i_2 *is equivalently an* ancestor *of* i_1*, if* $i_1 \ll_\pi^+ i_2$*.*
- *An index* i_1 *is an* immediate descendant *of an index* i_2*, and* i_2 *is equivalently an* immediate ancestor *of* i_1*, if* $i_1 \ll_\pi i_2$*.*
- *An index* i *is a* common descendant *of indices* i_1 *and* i_2 *if* i *is a descendant of both* i_1 *and* i_2*.*
- *An index* i *is the* greatest common descendant *of indices* i_1 *and* i_2 *if* i *is a common descendant of* i_1 *and* i_2 *and* i *is an ancestor of all other common descendants of* i_1 *and* i_2*.*

It is evident that if two indices have a common descendant, then they have a unique greatest common descendant.

4.5 Example *Based in the immediate descendant relation in Example 4.2 the following statements hold:*

- *The immediate ancestors of* $\frac{1}{3}$ *are* $\frac{1}{4}$ *and* $\frac{1}{5}$*, which are also the only ancestors of* $\frac{1}{3}$*.*
- *The descendants of* $\frac{3}{1}$ *are* $\frac{2}{1}$ *and* $\frac{1}{1}$*.*
- *The indices* $\frac{2}{1}$ *and* $\frac{1}{1}$ *are common descendants of* $\frac{3}{1}$ *and* $\frac{2.1}{2}$*. The greatest common descendant of* $\frac{3}{1}$ *and* $\frac{2.1}{2}$ *is* $\frac{2}{1}$*.*

4.6 Definition *Indices i_1 and i_2 occurring in a skeleton π are β-related, denoted $i_1 \triangle i_2$, if they are not $\ll_\pi^+$-related and they have a greatest common descendant of principal type β.*

4.7 Definition *If an index i is principal in a β-inference in a skeleton π, then the immediate ancestors of i are referred to as the β-options for i, and we equivalently say that the immediate ancestors of i are dual indices. For all dual indices i_1 and i_2 in a skeleton we define $\beta(i_1, i_2)$ to be the index for which i_1 and i_2 are β-options.*

As a result of this definition, dual indices are β-related.

4.8 Example *The only dual indices in the skeleton of Example 4.2 are $\frac{1}{4}$ and $\frac{1}{5}$, and $\beta(\frac{1}{4}, \frac{1}{5}) = \frac{1}{3}$.*

4.1.2 Syntax

In order label an instantiation variable according to which branch it occurs in, we need to extend the LK^{v}-syntax. As we shall see later, it is the indices of the active formulae in β-inferences that contribute to the branch labels.

4.9 Definition *A splitting set is a finite set of indices such that no two indices in it are β-related.*

Splitting sets are denoted $A, B, \ldots$ when the content is not explicitly displayed. As a convention, empty splitting sets are not displayed.

4.10 Definition *A decorated formula is an object φA where φ is an indexed formula and A is a splitting set. A decorated formula φA is closed if the indexed formula φ is closed.*

4.11 Example *Assuming the indices $\frac{1}{6}$, $\frac{1.1}{11}$ are not β-related, the following are decorated formulae.*

$$\underset{1}{\exists x}(\underset{3}{Px} \underset{2}{\rightarrow} \underset{4}{\forall x}\underset{5}{Px})^{1}\{\tfrac{1}{6}, \tfrac{1.1}{11}\}, \quad \underset{19}{\forall x}\underset{20}{Px}^{3}$$

The splitting set of the latter decorated formula is empty.

4.12 Definition *A decorated sequent is an object $\Gamma \vdash \Delta$ in which Γ and Δ are disjoint sets of closed decorated formulae.*

4.13 Example *The following is a decorated sequent.*

$$\underset{1}{\forall x}\underset{2}{Px}^{2}\{\tfrac{1}{4}\}, \underset{2}{Pu}^{1.1}\{\tfrac{1}{4}\} \vdash \underset{4}{Pa}^{1}$$

4.14 Definition *A colored variable is an object uA where u is an instantiation variable and A is a splitting set.*

4.15 Definition *Let A be a splitting set and u an instantiation variable. The color assignment operator, denoted $\oplus$, is defined as*

$$u \oplus A := uA \ .$$

We extend $\oplus$ to instantiation terms and atomic formulae recursively in the following way.

- *If a is a constant symbol or a Skolem constant, then $a \oplus A = a$.*
- *If f is an n-ary function symbol or Skolem function and $t_1, \ldots, t_n$ are instantiation terms, then $f(t_1, \ldots, t_n) \oplus A = f(t_1 \oplus A, \ldots, t_n \oplus A)$.*
- *If P is an n-ary relation symbol and $t_1, \ldots, t_n$ are instantiation terms, then $P(t_1, \ldots, t_n) \oplus A = P(t_1 \oplus A, \ldots, t_n \oplus A)$.*

An instantiation term is *colored* if all variables in it are colored. We redefine equations[1] such that an equation from now on is an ordered pair of *colored* instantiation terms. A *substitution for colored instantiation terms* is a substitution having the set of colored variables as domain and the set of colored instantiation terms (over a given first-order language) as its codomain. Unification of colored instantiation terms and satisfiability of equation sets are defined in terms of substitutions for colored instantiation terms. In the following we will refer to substitutions for colored instantiation terms as substitutions. For an equation set S, $\mathsf{Var}(S)$ denotes the set of all colored variables occurring in the left-hand or right-hand side of some equation in S.

4.16 Definition *The merging of a splitting set A with a splitting set B, denoted $A \boxdot B$, is defined as the union of A and the set containing all indices in B which are* not *β-related to any index in A.*

It it obvious that the operator $\boxdot$ preserves the property of being a splitting set. The operator is in general not commutative, as the following example illustrates.

4.17 Example *Let i_1, i_2, i_3 and i_4 be indices such that $i_1 \triangle i_2$ and $i_3 \triangle i_4$. Let $A = \{i_1, i_3\}$ and $B = \{i_2\}$. Then, $A \boxdot B = \{i_1, i_3\}$ and $B \boxdot A = \{i_2, i_3\}$. In case there are no two β-related indices $i_1 \in A$, $i_2 \in B$, then $A \boxdot B = B \boxdot A$.*

[1] Se Definition 2.24 on page 21.

4.18 Definition *Let Γ be a set of decorated formulae and let φ^κ be an indexed formula with formula number m. We define*

$$\Gamma^{\varphi^\kappa} := \{\ \psi(A \cup \{{}^{\kappa}_{m}\}) \mid \psi A \in \Gamma\ \}\ ,$$

i.e. the set of decorated formulae obtained by adding the index of φ^κ to the splitting set of every decorated formula in Γ.

4.1.3 The Sequent Calculus $\mathsf{LK}^{\mathsf{vs}}$

The rules of the sequent calculus $\mathsf{LK}^{\mathsf{vs}}$ define relations on decorated sequents. In all the rules, the splitting set decorating the principal formula and the active formulae are identical. The non-branching rules are like the non-branching rules of LK^{v}. The branching $\mathsf{LK}^{\mathsf{vs}}$-rules are listed in Figure 4.3 and require some explanation. They are designed in such a way that they increase the splitting sets of extra formulae in the premisses according to which branch they are copied into. The indices of the left and right active formula is added to the splitting set of each extra formula in the left and right premiss, respectively. In this way, the splitting set of a decorated formula can be viewed as a branch label. A complete listing of the $\mathsf{LK}^{\mathsf{vs}}$-rules is found on page 86.

β-rules

$$\frac{\Gamma^{\varphi^\kappa} \vdash \varphi^\kappa A, \Delta^{\varphi^\kappa} \quad \Gamma^{\psi^\kappa} \vdash \psi^\kappa A, \Delta^{\psi^\kappa}}{\Gamma \vdash (\varphi \wedge \psi)^\kappa A, \Delta}\ \mathsf{R}\wedge$$

$$\frac{\Gamma^{\varphi^\kappa}, \varphi^\kappa A \vdash \Delta^{\varphi^\kappa} \quad \Gamma^{\psi^\kappa}, \psi^\kappa A \vdash \Delta^{\psi^\kappa}}{\Gamma, (\varphi \vee \psi)^\kappa A \vdash \Delta}\ \mathsf{L}\vee$$

$$\frac{\Gamma^{\varphi^\kappa} \vdash \varphi^\kappa A, \Delta^{\varphi^\kappa} \quad \Gamma^{\psi^\kappa}, \psi^\kappa A \vdash \Delta^{\psi^\kappa}}{\Gamma, (\varphi \to \psi)^\kappa A \vdash \Delta}\ \mathsf{L}\to$$

Figure 4.3: The β-rules of $\mathsf{LK}^{\mathsf{vs}}$. The symbol A is a splitting set. The index of the left (right) active formula is added to the splitting set of each extra formula in the left (right) premiss.

4.19 Definition (Set of $\mathsf{LK}^{\mathsf{vs}}$-skeletons) *The set of $\mathsf{LK}^{\mathsf{vs}}$-skeletons is defined inductively with relation to the $\mathsf{LK}^{\mathsf{vs}}$-rules as follows.*

- *If $\Gamma \vdash \Delta$ is a decorated sequent in which all copy histories are equal to 1, no two subformulae of decorated formulae in $\Gamma \cup \Delta$ have identical*

formula numbers, and all splitting sets are empty, then $\Gamma \vdash \Delta$ *is an* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton.*

- *The induction steps are defined like in Definition 2.14.*

The restriction on root sequents in $\mathsf{LK}^{\mathsf{vs}}$-skeletons are like the one for LK^{v}-skeletons in Definition 2.30, except that all formulae are decorated with empty splitting sets. Also, the $\mathsf{LK}^{\mathsf{vs}}$-rules handle indices of principal and active formulae in the same way as the LK^{v}-rules. Hence, all notions and results regarding the indexing system of LK^{v}-skeletons, including Proposition 2.33 and the definitions in Section 4.1.1, transfer to $\mathsf{LK}^{\mathsf{vs}}$-skeletons. The *splitting set of a branch* in an $\mathsf{LK}^{\mathsf{vs}}$-skeleton is defined as the union of all splitting sets occurring in the branch.

4.20 Lemma *The splitting set of a branch in an* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton contains no dual indices.*

Proof Suppose the splitting set of a branch B contains dual indices i', i''. Then, there must be a β-inference θ in B in which the principal formula has index $i = \beta(i', i'')$. By Proposition 2.33 there can be at most one inference in B in which the principal formula has index i. But i', i'' are introduced in different branches by θ. Hence, they cannot occur in the same branch. ∎

The motivation for labelling formulae in $\mathsf{LK}^{\mathsf{vs}}$-skeletons with splitting sets is to label different branch occurrences of an instantiation variable differently. In the process of generating primary equations for a connection, a color is generated from the splitting sets of the atomic formulae in the connection and transferred to terms by means of the the color assignment operator $\oplus$. Care must be taken when generating the color. Due to the context sharing rules, a connection c resulting from an inference θ occurs in every leaf sequent above θ in a skeleton. When c is copied as part of context in β-inferences, the splitting sets of c are increased with different indices in the left and right premisses. Thus, the splitting sets of every occurrence of c above θ are different. Nevertheless, we want each leaf sequent occurrence of c to produce identical primary equations. This is achieved as follows.

4.21 Definition *A connection* c *is a sequent of the form*

$$P(s_1, \ldots, s_n)A \vdash P(t_1, \ldots, t_n)B \ .$$

The corresponding colored connection, denoted $\bar{c}$*, is*

$$P(s_1, \ldots, s_n) \oplus (A \setminus B) \vdash P(t_1, \ldots, t_n) \oplus (B \setminus A).$$

A colored variable $u \oplus (A \setminus B)$ *is called a pruning of the unpruned variable* uA*.*

4.22 Definition *Let c be a connection $P(s_1,\dots,s_n)A \vdash P(t_1,\dots,t_n)B$. The set of primary equations for c, denoted $\mathsf{Prim}(c)$, is defined as*

$$\mathsf{Prim}(c) := \{s_i \oplus (A \setminus B) \approx t_i \oplus (B \setminus A) \mid 1 \leq i \leq n\}\,.$$

A set of connections is *spanning* for an $\mathsf{LK}^{\mathsf{vs}}$-skeleton π if it contains exactly one connection from each leaf sequent of π. The set of primary equations for a spanning set C is defined like in LK^{v}, i.e. $\mathsf{Prim}(C) := \bigcup_{c\in C} \mathsf{Prim}(c)$. The set $\mathsf{Var}(C)$ contains colored variables such that $uA \in \mathsf{Var}(C)$ if and only if uA occurs in $\bar{c}$ for some connection $c \in C$.

As mentioned in the introduction to this chapter, it is not sufficient for a substitution to satisfy the set of primary equations generated from a spanning connection set in order to be closing for an $\mathsf{LK}^{\mathsf{vs}}$-skeleton. The following notions provide a basis for the definition of closing substitutions.

4.23 Definition *Let π be an $\mathsf{LK}^{\mathsf{vs}}$-skeleton, and let C be a connection set for π. The set of balancing equations for C, denoted $\mathsf{Bal}(C)$, is the set of equations such that $uA \approx uB \in \mathsf{Bal}(C)$ if and only if*

- *$uA, uB \in \mathsf{Var}(C)$, and*
- *$A \boxdot B = B \boxdot A$.*

Source identical formulae may occur in different branches of a skeleton. If two formula occurrences in different branches are source identical and one is expanded and the other is not, then the skeleton is *imbalanced*; there is an inference in one branch which also could have been done in the other. A skeleton is *balanced* if it does not have the above described skewness. If we require skeletons to be balanced, we put undesirable restrictions on selection of expansion formulae in a proof search process. However, in an imbalanced skeleton too much liberty is given to substitutions in instantiating variables. The main purpose of balancing equations is to simulate identities between colored variables as they would have been if the skeleton was balanced. An in-depth treatment of balancing equations will appear in [3].

4.24 Definition ($\prec_\sigma$-relation) *Let π be an $\mathsf{LK}^{\mathsf{vs}}$-skeleton, let C be a spanning connection set for π, and let σ be a substitution. The dependency relation induced by σ on π wrt. C, denoted $\prec_\sigma$, is a binary relation on indices in π such that $i_1 \prec_\sigma i_2$ if and only if*

- *there are colored variables uA, uB in $\mathsf{Var}(C)$ such that u has index i_2 and $\sigma(uA) \neq \sigma(uB)$, and*
- *there are dual indices $i' \in A$, $i'' \in B$ such that $i_1 = \beta(i', i'')$.*

4.25 Definition *A substitution σ is* closing *for an* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton π if there is a spanning connection set C for π such that σ satisfies* $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$*, and the relation $(\ll_\pi \cup \prec_\sigma)^+$ wrt. C is irreflexive.*

4.26 Definition *An* $\mathsf{LK}^{\mathsf{vs}}$*-proof is a tuple $\langle \pi, C, \sigma \rangle$ where π is an* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton, C is a spanning set of connections for π and σ is a closing substitution for π wrt. C.*

4.27 Conjecture *The sequent calculus* $\mathsf{LK}^{\mathsf{vs}}$ *is sound and complete.*

As mentioned in the introduction to this chapter, an informal consistency proof for $\mathsf{LK}^{\mathsf{vs}}$ is that a substitution is closing for a skeleton only if it induces dependencies between inferences in such a way that there is a permutation variant which can be simulated in a variable pure sequent calculus. The standard variable pure sequent calculus is known to be sound (it is equivalent to the restricted free variable tableaux in [12]). Consistency and completeness of the variable splitting calculus will be treated in [3].

4.1.4 Towards a Search Procedure

In a proof search process, the question of whether a skeleton is closable controls termination of the proof search. A satisfiable equation set has an infinite set of unifiers. If we have to check each of them for reflexivity in order to determine that a skeleton is *not* closable, and thus perform the next skeleton expansion, it is impossible to design a proof search procedure for $\mathsf{LK}^{\mathsf{vs}}$. It turns out that in order to determine closability it is sufficient to check whether a most general unifier for the set of primary and balancing equations induces a irreflexive relation on the skeleton.

4.28 Lemma *Let π be an* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton, C a spanning connection set for π and σ a most general unifier for* $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$*. Let uA, uB be colored variables in* $\mathsf{Var}(C)$*. If the terms $(uA)\sigma$, $(uB)\sigma$ are non-unifiable, then $(uA)\sigma' \neq (uB)\sigma'$ for all unifiers σ' for* $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$*.*

Proof Assume for a contradiction that $(uA)\sigma' = (uB)\sigma'$. Since σ is a most general unifier for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$ and thus more general than σ', there is a substitution τ such that $\sigma' = \sigma\tau$. But then τ is a unifier for the non-unifiable terms $(uA)\sigma$, $(uB)\sigma$. ∎

Throughout the rest if this chapter we generate the $\prec_\sigma$-relation as follows when testing a most general unifier for irreflexivity.

4.29 Definition ($\prec_\sigma$-relation for mgus) *Let π be an* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton, let C be a spanning connection set for π, and let σ be a most general unifier*

for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$. *The* dependency relation induced by σ on π wrt. C, *denoted* $\prec_\sigma$, *is a binary relation on indices in* π *such that* $i_1 \prec_\sigma i_2$ *if and only if*

- *there are colored variables* uA, uB *in* $\mathsf{Var}(C)$ *such that* u *has index* i_2 *and* the terms $\sigma(uA)$ and $\sigma(uB)$ are non-unifiable, *and*
- *there are dual indices* $i' \in A$, $i'' \in B$ *such that* $i_1 = \beta(i', i'')$.

4.30 Lemma *Let* π *be an* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton,* C *a spanning connection set for* π *and* σ *a most general unifier for* $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$. *If* $(\ll_\pi \cup \prec_\sigma)^+$ *wrt.* C *is reflexive, then* $(\ll_\pi \cup \prec_{\sigma'})^+$ *wrt.* C *is reflexive for all unifiers* σ' *for* $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$.

Proof Let σ' be some unifier for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$, and assume that the relation $(\ll_\pi \cup \prec_\sigma)^+$ is reflexive. Then, $\prec_\sigma$ must be non-empty. (Otherwise, $(\ll_\pi \cup \prec_\sigma)^+$ is irreflexive by Lemma 4.3.) Pick some $(i_1, i_2) \in \prec_\sigma$. By Definition 4.29 there are variables $uA, uB \in \mathsf{Var}(C)$ such that u has index i_2 and the terms $(uA)\sigma$, $(uB)\sigma$ are non-unifiable. Since σ is more general than σ', $(uA)\sigma' \neq (uB)\sigma'$ by Lemma 4.28. But then $(i_1, i_2) \in \prec_{\sigma'}$ by Definition 4.24. Since (i_1, i_2) was arbitrarily chosen, $\prec_\sigma$ is a subset of $\prec_{\sigma'}$, and hence $(\ll_\pi \cup \prec_{\sigma'})^+$ is reflexive. ∎

As a consequence of Lemma 4.30, there are no two most general unifiers σ, σ' for an equation set $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$ such that $(\ll_\pi \cup \prec_\sigma)^+$ wrt. C is reflexive and $(\ll_\pi \cup \prec_{\sigma'})^+$ wrt. C is irreflexive. Lemma 4.30 also yields the following proposition.

4.31 Proposition *An* $\mathsf{LK}^{\mathsf{vs}}$*-skeleton* π *is closable if and only if there is a spanning connection set* C *for* π *such that the relation* $(\ll_\pi \cup \prec_\sigma)^+$ *wrt.* C *is irreflexive for a most general unifier* σ *for* $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$.

Another problem with the $\mathsf{LK}^{\mathsf{vs}}$-definition of closing substitution is the irreflexivity check itself. It is not conceptually intuitive. By Lemma 4.3 the relation $\ll_\pi$ is irreflexive and $\prec_\sigma$ is irreflexive by definition (it relates β- and γ-indices). Thus, in order for $(\ll_\pi \cup \prec_\sigma)^+$ to be reflexive, there must be a sequence of indices $i_1, \ldots, i_n$ such that (i_n, i_1) and each (i_k, i_{k+1}) are in $(\ll_\pi \cup \prec_\sigma)$. An *index graph* is a directed graph where the nodes are the indices occurring in $(\ll_\pi \cup \prec_\sigma)$ and the edges are the relation $(\ll_\pi \cup \prec_\sigma)$ itself. Hence, the reflexivity check for $(\ll_\pi \cup \prec_\sigma)^+$ is reduced to a cycle check of the index graph for $(\ll_\pi \cup \prec_\sigma)$. I will in the following display index graphs with the symbols in Figure 4.4.

Another observation is that the index graph constructed from $\ll_\pi$ alone is acyclic. This follows by a simple argument based on Proposition 2.33.

(Intuitively, the relation $\ll_\pi$ is a relation between (instances of) subformulae, i.e. a set of formula trees [1].) Also, $\ll_\pi$ can be updated incrementally in parallel with skeleton expansion. The cycle check can thus be performed by adding the members of $\prec_\sigma$ to the index tree for $\ll_\pi$ one by one. If the index tree was acyclic before a new edge e is added, then a cycle in the graph after e is added includes e.

As a result of the contraction copies in γ-expansions, an index graph can grow very large. However, we do not have to store the entire graph. A graph structure keeping track of only relations between *formula numbers* in the skeleton π is finite and can be constructed from the root sequent. The $\ll_\pi$ relation can then be calculated directly for each index according to these rules taken from [3].

Let $\kappa = \langle k_1, \ldots, k_n \rangle$ and $\tau = \langle t_1, \ldots, t_m \rangle$. Then $\begin{smallmatrix}\kappa \\ m\end{smallmatrix} \ll_\pi \begin{smallmatrix}\tau \\ n\end{smallmatrix}$ holds if

- m is related to n or $m = n$ in the formula number relation for π, and
- $n \leq m$, and
- $k_i = t_i$ for $1 \leq i \leq n$, and
- $k_n \leq t_n$.

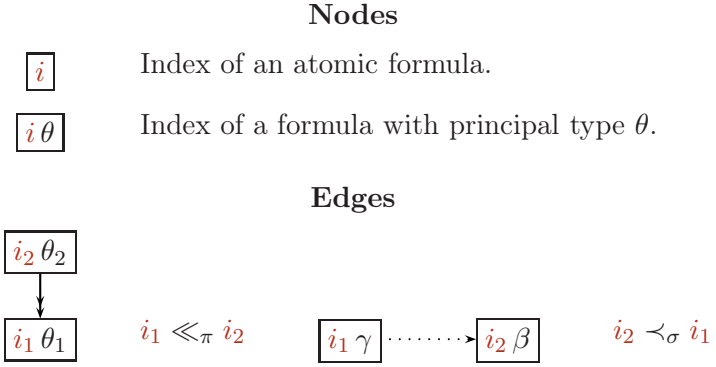


Figure 4.4: The node and edge symbols used in an index graph.

4.1.5 Examples

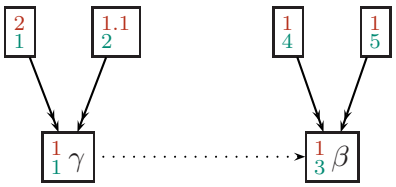
In this section we present some examples illustrating the main concepts of $\mathsf{LK}^{\mathsf{vs}}$-skeletons and closability.

4.32 Example *Let π be the $\mathsf{LK}^{\mathsf{vs}}$-skeleton in Figure 4.5. Let C be the spanning connection set consisting of the connections*

$$c_1 \;:\; Pu^{1.1}\{\begin{smallmatrix}1\\4\end{smallmatrix}\} \vdash Pa^1 \quad \textit{and} \quad c_2 \;:\; Pu^{1.1}\{\begin{smallmatrix}1\\5\end{smallmatrix}\} \vdash Pb^1$$

$$\dfrac{\dfrac{\forall x Px^{2}\{^{1}_{4}\}, Pu^{1.1}\{^{1}_{4}\} \vdash Pa^{1}}{\forall x Px^{1}\{^{1}_{4}\} \vdash Pa^{1}}\,\gamma_u \qquad \dfrac{\forall x Px^{2}\{^{1}_{5}\}, Pu^{1.1}\{^{1}_{5}\} \vdash Pb^{1}}{\forall x Px^{1}\{^{1}_{5}\} \vdash Pb^{1}}\,\gamma_u}{\underset{1}{\forall x} \underset{2}{Px^{1}} \vdash \underset{4}{Pa} \underset{3}{\wedge} \underset{5}{Pb^{1}}}\,\beta$$

$$\sigma = \{\, u\{^{1}_{4}\}/a,\ u\{^{1}_{5}\}/b \,\}$$



Abbreviations: $u = u^{1}_{1}$

Figure 4.5: A closable $\mathsf{LK}^{\mathsf{vs}}$-skeleton. See Example 4.32 for details.

from the left and right leaf sequent of π. The set of primary equations for C is $\mathsf{Prim}(C) = \{u\{^{1}_{4}\} \approx a, u\{^{1}_{5}\} \approx b\}$, and the set of balancing equations is empty. The substitution $\sigma = \{u\{^{1}_{4}\}/a, u\{^{1}_{5}\}/b\}$ satisfies $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$. The relation $\prec_\sigma$ wrt. C is $\{(^{1}_{3}, ^{1}_{1})\}$, but the index graph for $\ll_\pi \cup \prec_\sigma$ wrt. C is acyclic, as shown in Figure 4.5. Hence, σ is closing for π.

4.33 Example *Let π be the $\mathsf{LK}^{\mathsf{vs}}$-skeleton in Figure 4.6. The root sequent*

$$\exists x(Rx \to Px), \forall x Rx, \exists x Qx \vdash \exists x(Px \wedge Qx)$$

is not valid. A falsifying model is e.g. $|M| = \{0, 1\}$ such that $0 \in P^M$, $1 \in Q^M$ and $0, 1 \in R^M$. The connection set C containing

$$\begin{array}{lcl} c_1 & : & Ru^{1.1}\{^{1}_{3}\} \vdash Ra^{1} \\ c_2 & : & Pa^{1}\{^{1.1}_{11}\} \vdash Pv^{1.1}\{^{1}_{4}\} \\ c_3 & : & Qb^{1}\{^{1}_{4}, ^{1.1}_{12}\} \vdash Qv^{1.1}\{^{1}_{4}\} \end{array}$$

is the only spanning connection set for π. The set of primary equations for C is

$$\mathsf{Prim}(C) = \{u\{^{1}_{3}\} \approx a,\ a \approx v\{^{1}_{4}\},\ b \approx v\}$$

and the set of balancing equations is

$$\mathsf{Bal}(C) = \{v \approx v\{^{1}_{4}\}\}\ .$$

Let $\sigma = \{\, u\{^{1}_{3}\}/a,\ v\{^{1}_{4}\}/a,\ v/b \,\}$, which is a most general unifier for $\mathsf{Prim}(C)$. The relation $\prec_\sigma$ is empty, and thus the index graph for $\ll_\pi \cup \prec_\sigma$ is acyclic. But since σ does not satisfy $\mathsf{Bal}(C)$, it is not closing for π. The set $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$ is not satisfiable, and hence the skeleton is not closable.

4.34 Example *Let π be the $\mathsf{LK}^{\mathsf{vs}}$-skeleton in Figure 4.7. The root sequent*

$$\forall x((Px \wedge Sa) \vee (Qx \wedge Rb)) \vdash \exists x((Px \vee Rx) \wedge (Qx \vee Sx))$$

is not valid. A falsifying model is e.g. $|M| = \{0,1\}$ such that $a^M = 1$, $b^M = 0$, $0 \in P^M$, $1 \in Q^M$, $0 \in R^M$ and $1 \in S^M$. The connection set C containing

$$\begin{array}{lcl} c_1 & : & Pu^{1.1}\{{}^{1.1}_{11}\} \vdash Pv^{1.1}\{{}^{1.1}_{3}\} \\ c_2 & : & Rb^{1.1}\{{}^{1.1}_{11}\} \vdash Rv^{1.1}\{{}^{1.1}_{6}\} \\ c_3 & : & Sa^{1.1}\{{}^{1.1}_{14}\} \vdash Sv^{1.1}\{{}^{1.1}_{3}\} \\ c_4 & : & Qu^{1.1}\{{}^{1.1}_{14}\} \vdash Qv^{1.1}\{{}^{1.1}_{6}\} \end{array}$$

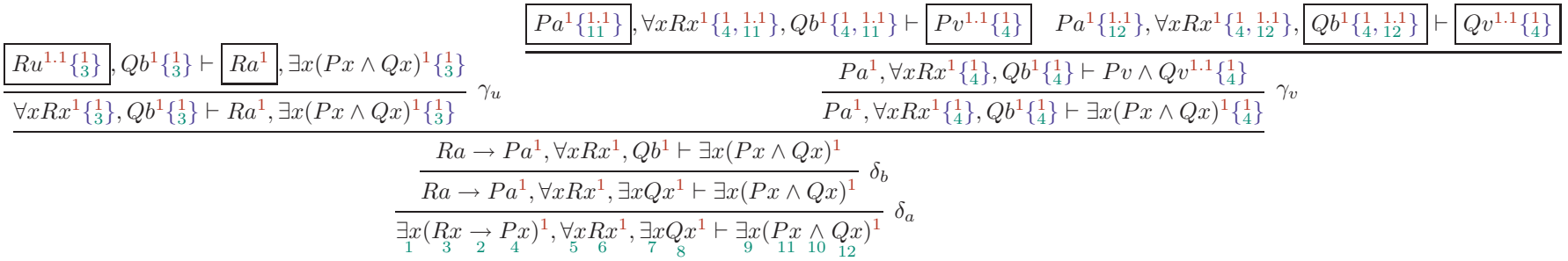
is the only spanning connection set for π. The set of primary equations for C is

$$\mathsf{Prim}(C) = \{u\{{}^{1.1}_{11}\} \approx v\{{}^{1.1}_{3}\},\ b \approx v\{{}^{1.1}_{6}\},\ a \approx v\{{}^{1.1}_{3}\},\ u\{{}^{1.1}_{14}\} \approx v\{{}^{1.1}_{6}\}\}\ ,$$

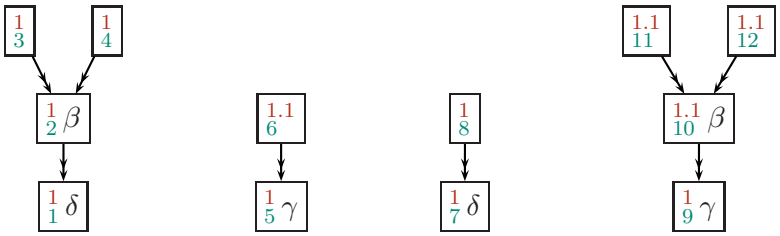
and the set of balancing equations for C is empty. The substitution

$$\sigma = \{\, u\{{}^{1.1}_{11}\}/a,\ u\{{}^{1.1}_{14}\}/b,\ v\{{}^{1.1}_{3}\}/a,\ v\{{}^{1.1}_{6}\}/b \,\}$$

is a most general unifier for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$. The relation induced by σ on π wrt. C is $\prec_\sigma = \{({}^{1.1}_{10}, {}^{1}_{1}),\ ({}^{1.1}_{2}, {}^{1}_{9})\}$, which generates a cycle in the index graph for $\ll_\pi \cup \prec_\sigma$ as shown in Figure 4.7. Thus, σ is not closing for π. Since σ is a most general unifier for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$, the skeleton is not closable.



$$\sigma = \{\, u\{\tfrac{1}{3}\}/a,\ v\{\tfrac{1}{4}\}/a,\ v/b \,\}$$



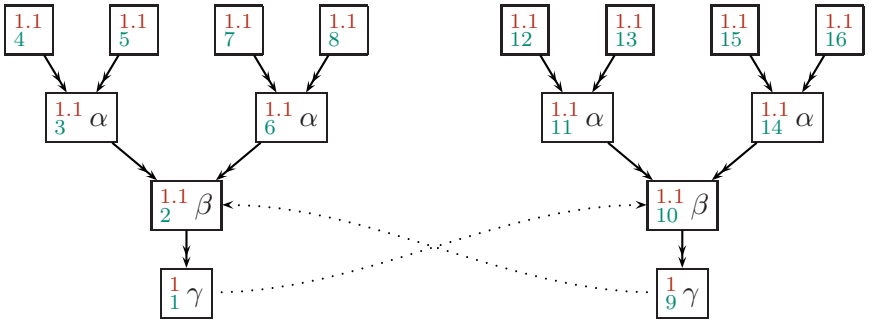
Abbreviations: $a = a_1 \quad b = a_7 \quad u = u^1_5 \quad v = u^1_9$

Figure 4.6: A non-closable $\mathsf{LK}^{\mathsf{vs}}$-skeleton. Contraction copies of γ-formulae are not displayed. See Example 4.33 for details.

$$\dfrac{\dfrac{\dfrac{Pu^{1.1}\{^{1.1}_{11}\} \vdash Pv^{1.1}\{^{1.1}_{3}\}}{Pu \wedge Sa^{1.1}\{^{1.1}_{11}\} \vdash Pv \vee Rv^{1.1}\{^{1.1}_{3}\}} \quad \dfrac{Rb^{1.1}\{^{1.1}_{11}\} \vdash Rv^{1.1}\{^{1.1}_{6}\}}{Qu \wedge Rb^{1.1}\{^{1.1}_{11}\} \vdash Pv \vee Rv^{1.1}\{^{1.1}_{6}\}}\,\beta}{(Pu \wedge Sa) \vee (Qu \wedge Rb)^{1.1}\{^{1.1}_{11}\} \vdash Pv \vee Rv^{1.1}} \quad \dfrac{\dfrac{Sa^{1.1}\{^{1.1}_{14}\} \vdash Sv^{1.1}\{^{1.1}_{3}\}}{Pu \wedge Sa^{1.1}\{^{1.1}_{14}\} \vdash Qv \vee Sv^{1.1}\{^{1.1}_{3}\}} \quad \dfrac{Qu^{1.1}\{^{1.1}_{14}\} \vdash Qv^{1.1}\{^{1.1}_{6}\}}{Qu \wedge Rb^{1.1}\{^{1.1}_{14}\} \vdash Qv \vee Sv^{1.1}\{^{1.1}_{6}\}}\,\beta}{(Pu \wedge Sa) \vee (Qu \wedge Rb)^{1.1}\{^{1.1}_{14}\} \vdash Qv \vee Sv^{1.1}}\,\beta}{(Pu \wedge Sa) \vee (Qu \wedge Rb)^{1.1} \vdash (Pv \vee Rv) \wedge (Qv \vee Sv)^{1.1}}\,\gamma_v$$

$$\dfrac{(Pu \wedge Sa) \vee (Qu \wedge Rb)^{1.1} \vdash \exists x((Px \vee Rx) \wedge (Qx \vee Sx))^{1}}{\forall x((Px \wedge Sa) \vee (Qx \wedge Rb))^{1} \vdash \exists x((Px \vee Rx) \wedge (Qx \vee Sx))^{1}}\,\gamma_u$$

$$\sigma = \{\, u\{^{1.1}_{11}\}/a,\ u\{^{1.1}_{14}\}/b,\ v\{^{1.1}_{3}\}/a,\ v\{^{1.1}_{6}\}/b \,\}$$



Abbreviations: $u = u^1_1$ $v = u^1_9$

Figure 4.7: A non-closable $\mathsf{LK}^{\mathsf{vs}}$-skeleton. Contraction copies of γ-formulae are not displayed. See Example 4.34 for details.

4.2 Constraints for Splitting Skeletons

The increased restrictions put upon closing substitutions by the closure condition of $\mathsf{LK}^{\mathsf{vs}}$ must be reflected by the constraint definitions. I shall use two different approaches. First, I present a constraint language almost identical to the one for LK^{v}-constraints. These constraints capture the primary and secondary equations of spanning connections sets and calculates them inductively. The cycle check is however only performed when a satisfiable atomic constraint reaches the root sequent. Thus, while primary and secondary equations are calculated incrementally, the cycle check is global. Then, I present a variant of incremental constraints which calculates the dependency relation and performs cycle checking incrementally during propagation of increment sets. With this approach, a satisfiable atomic constraint reaches the root sequent only if the induced index graph is acyclic. The notions defined in Section 3.1 regarding expansion sequences, set of leaf sequents and connections, new connections, etc. transfer to $\mathsf{LK}^{\mathsf{vs}}$-skeletons in the obvious way.

In Section 4.2.1 the constraint language is introduced, and in Section 4.2.2 global constraints are defined. In Section 4.2.3 I introduce incremental splitting constraints and present an example of constraint calculation. Finally, and alternative constraint definition with an incremental cycle check is presented in Section 4.2.4.

4.2.1 Constraint Language

Constraints and atomic constraints are defined in the same way as in Section 3.2.1. Also, the satisfiability relation for constraints is like in Definition 3.21. When two atomic constraints are merged, new balancing equations are generated as follows.

4.35 Definition *Let S_1 and S_2 be equation sets. Then, $\mathsf{Bal}(S_1, S_2)$ is the set of equations such that $uA \approx uB \in \mathsf{Bal}(S_1, S_2)$ if and only if $uA \in \mathsf{Var}(S_1)$, $uB \in \mathsf{Var}(S_2)$ and $A \boxdot B = B \boxdot A$.*

The merging operator is defined as follows.

4.36 Definition (Merging) *Let μ_1 and μ_2 be atomic constraints. The merging of μ_1 and μ_2, denoted $\mu_1 \otimes \mu_2$, is defined as follows.*

- *If $\mu_1 = \bot$ or $\mu_2 = \bot$, then*

$$\mu_1 \otimes \mu_2 := \bot \,.$$

- *Otherwise,*

$$\mu_1 \otimes \mu_2 := \mathsf{Solve}(\mu_1 \cup \mu_2 \cup \mathsf{Bal}(\mu_1, \mu_2)) \ .$$

The merging operator is extended to constraints in the same way as in Definition 3.25.

As we showed in Section 3.1.3, equation sets in solved form can be utilized as substitutions. This is formalized as follows.

4.37 Definition *Let S be an equation set in solved form. The application of S on a colored variable uA, denoted $S(uA)$, is defined as follows.*

- *If there is an equation $uA \approx t$ in S, then*

$$S(uA) := t \ .$$

- *Otherwise,*

$$S(uA) := uA \ .$$

Since the left-hand sides of equations in an equation set in solved form are distinct, the above definition is well-defined. The cycle checking operator is defined as follows.

4.38 Definition *For each $\mathsf{LK}^{\mathsf{vs}}$-skeleton π_k we define a cycle check operator, denoted Check_{π_k}, for atomic constraints as follows. Let μ be an atomic constraint.*

- *If $\mu = \bot$, then*

$$\mathsf{Check}_{\pi_k}(\mu) := \bot \ .$$

- *Otherwise, let $S = \mathsf{MGU}(\mu)$. If the index graph for $\ll_{\pi_k} \cup \prec_S$ contains a cycle, then*

$$\mathsf{Check}_{\pi_k}(\mu) := \bot \ .$$

- *Otherwise,*

$$\mathsf{Check}_{\pi_k}(\mu) := \mu \ .$$

The cycle check operator is extended to constraints as follows. Let χ be a constraint.

$$\mathsf{Check}_{\pi_k}(\chi) := \{\mathsf{Check}_{\pi_k}(\mu) \mid \mu \in \chi\}$$

4.2.2 Global Constraints

The global constraint definition and its correctness proof facilitates the correctness proof for the incremental constraints in the next section. When generating atomic constraints for connections, we have to include both the primary and secondary equations.

4.39 Definition *For each connection c we define an atomic constraint, denoted* $\mathsf{Atom}(c)$*, as follows.*

$$\mathsf{Atom}(c) := \mathsf{Solve}(\mathsf{Prim}(c) \cup \mathsf{Bal}(c))$$

Global constraints are defined as follows.

4.40 Definition (Global Constraints) *For each sequent s in a skeleton π_k we define a* global constraint*, denoted* $\mathsf{GC}_k(s)$*, as follows.*

- *If s is a leaf sequent, then*

$$\mathsf{GC}_k(s) := \{\mathsf{Atom}(c) \mid c \in \mathsf{Conn}(s)\}.$$

- *Otherwise, let* $\mathsf{Lvs}_k(s) = \{l_0, \ldots, l_n\}$*. Then,*

$$\mathsf{GC}_k(s) := (\mathsf{GC}_k(l_0) \otimes \mathsf{GC}_k(l_1)) \otimes \cdots \otimes \mathsf{GC}_k(l_n)$$

We define the global constraint for π_k, denoted $\mathsf{GC}(\pi_k)$*, as*

$$\mathsf{GC}(\pi_k) := \mathsf{Check}_{\pi_k}(\mathsf{GC}_k(r)),$$

where r is the root sequent of π_k.

The following lemma is needed in the correctness proof for global $\mathsf{LK}^{\mathsf{vs}}$-constraints.

4.41 Lemma *Let π_k be an $\mathsf{LK}^{\mathsf{vs}}$-skeleton, let $C = \{c_1, \ldots, c_n\}$ be a set of connections, and let $\mu = (\mathsf{Atom}(c_1) \otimes \mathsf{Atom}(c_2)) \otimes \ldots \otimes \mathsf{Atom}(c_n)$. Then a substitution σ satisfies μ if and only if σ satisfies* $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$*.*

Proof By induction on n. The base case ($n = 1$) is trivial.

Induction step: Let $\mu' = (\mathsf{Atom}(c_1) \otimes \mathsf{Atom}(c_2)) \otimes \ldots \otimes \mathsf{Atom}(c_n)$, let $\mu = \mu' \otimes \mathsf{Atom}(c_{n+1})$, let $C' = \{c_1, \ldots, c_n\}$, and let $C = C' \cup \{c_{n+1}\}$. We assume the claim holds for μ' (IH), and show that it holds for μ.

"If"-direction: Assume σ satisfies $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$. Then σ satisfies $\mathsf{Prim}(C') \cup \mathsf{Prim}(c_{n+1})$. Since $\mathsf{Bal}(C')$ and $\mathsf{Bal}(c_{n+1})$ are subsets of $\mathsf{Bal}(C)$, σ satisfies μ' and $\mathsf{Atom}(c_{n+1})$ by IH and Definition 4.39. Pick some $uA \approx uB$

in $\mathsf{Bal}(\mu', \mathsf{Atom}(c_{n+1}))$. Since the set of colored variables in C' and c_{n+1} are subsets of $\mathsf{Var}(C)$, $uA \approx uB$ is in $\mathsf{Bal}(C)$ and hence it is solved by σ. But then σ satisfies $\mu = \mu' \cup \mathsf{Atom}(c_{n+1}) \cup \mathsf{Bal}(\mu', \mathsf{Atom}(c_{n+1}))$.

"Only if"-direction: Assume σ satisfies μ. Then, σ satisfies μ', and hence $\mathsf{Prim}(C')$ and $\mathsf{Bal}(C')$ by IH. σ also satisfies $\mathsf{Atom}(c_{n+1})$, and hence $\mathsf{Prim}(c_{n+1})$ and $\mathsf{Bal}(c_{n+1})$. Then σ satisfies $\mathsf{Prim}(C)$. Pick some $uA \approx uB$ in $\mathsf{Bal}(C)$. If $uA \approx uB$ is in $\mathsf{Bal}(C')$ or $\mathsf{Bal}(c_{n+1})$, then it is solved by σ. If $uA \approx uB$ is in neither sets, then uA is in $\mathsf{Var}(C')$ and uB is in $\mathsf{Var}(\{c_{n+1}\})$ or vice versa. By Definition 4.35 $uA \approx uB$ is in $\mathsf{Bal}(\mu', \mathsf{Atom}(c_{n+1}))$, and hence it is solved by σ. ∎

We cannot show correctness of global $\mathsf{LK}^{\mathsf{vs}}$-constraints in the same way as for LK^{v}-constraints. The reason is that there are substitutions which satisfy the global constraint for a skeleton π_k, but are not closing for π_k. Assume that colored variables uA, uB in $\mathsf{Var}(C)$ contain dual indices, and that $(uA)\sigma = fv$, $(uB)\sigma = fw$ for a most general unifier σ for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$. The terms fv and fw are unifiable. If $\tau = \sigma\{v/a, w/b\}$ is a unifier for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$, then the terms $(uA)\tau$, $(uB)\tau$ are non-unifiable, and the pair uA, uB contributes to the $\prec_\tau$-relation. If the index graph for $\ll_{\pi_k} \cup \prec_\sigma$ is acyclic and the graph for $\ll_{\pi_k} \cup \prec_\tau$ is cyclic, then τ satisfies $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$ without being closing for π_k. In a proof search process, the important thing is whether a skeleton is closable or not. This can be expressed by means of satisfiability of global constraints as follows.

4.42 Lemma *Let π_k be an $\mathsf{LK}^{\mathsf{vs}}$-skeleton. Then, $\mathsf{GC}(\pi_k)$ is satisfiable if and only if π_k is closable.*

Proof Let r be the root sequent of π_k, and let $\mathsf{Lvs}(\pi_k) = \{l_1, \ldots, l_n\}$.

"If"-direction: Assume σ is closing for π_k. Then, there is a spanning set $C = \{c_1, \ldots, c_n\}$ for π_k such that σ satisfies $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$. Let $\mu = (\mathsf{Atom}(c_1) \otimes \mathsf{Atom}(c_2)) \otimes \ldots \otimes \mathsf{Atom}(c_n)$. By Lemma 4.41 σ satisfies μ, and thus σ satisfies $\mathsf{GC}_k(r)$. Since σ is closing for π_k, the index graph for $\ll_{\pi_k} \cup \prec_\sigma$ is acyclic. Then, by Lemma 4.30 the index graph $\ll_{\pi_k} \cup \prec_{\sigma'}$ for any most general unifier σ' for $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$ is acyclic, and hence $\mathsf{Check}_{\pi_k}(\mu) = \mu$. But then σ satisfies $\mathsf{GC}(\pi_k)$.

"Only if"-direction: Assume $\mathsf{GC}(\pi_k)$ is satisfiable. Then, $\mathsf{Check}_{\pi_k}(\mu)$ is satisfiable for some μ in $\mathsf{GC}_k(r)$, and hence the index graph for each most general unifier for μ is acyclic. (If one is cyclic, then all are by Lemma 4.30.) Let σ be a most general unifier for μ. By Definition 4.40 $\mu = (\mathsf{Atom}(c_1) \otimes \mathsf{Atom}(c_2)) \otimes \ldots \otimes \mathsf{Atom}(c_n)$ for some spanning set $C = \{c_1, \ldots, c_n\}$ for π_k. By Lemma 4.41 σ satisfies $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$, and hence σ is closing for π_k. ∎

4.43 Example *Let π_k be the skeleton in Figure 4.7, let r be the root sequent of π_k, and let the leaf sequents of π_k be (from left to right) l_1, l_2, l_3 and l_4.*

Let $C = \{c_1, \ldots, c_4\}$ be the connection set as listed in Example 4.34. The global constraint for π_k is

$$\begin{aligned}\mathsf{GC}(\pi_k) &= \mathsf{Check}_{\pi_k}(\mathsf{GC}_k(r)) \\ &= \mathsf{Check}_{\pi_k}([[\mathsf{GC}_k(l_1) \otimes \mathsf{GC}_k(l_2)] \otimes \mathsf{GC}_k(l_3)] \otimes \mathsf{GC}_k(l_4)) \\ &= \mathsf{Check}_{\pi_k}(\{[[\mathsf{Atom}(c_1) \otimes \mathsf{Atom}(c_2)] \otimes \mathsf{Atom}(c_3)] \otimes \mathsf{Atom}(c_4)\}) \\ &= \mathsf{Check}_{\pi_k}(\{\mathsf{Prim}(C)\}) \\ &= \{\bot\}\end{aligned}$$

The set $\mathsf{Prim}(C)$ is the same equation set as in Example 4.34, for which there is only one most general unifier. See the index graph in Figure 4.7 for intuitions on why the cycle check fails.

4.2.3 Incremental Constraints

The definition of incremental constraints is similar to the one used in Section 3.2.3. For convenience, I have included the complete definition and put the differences from the LK^{v}-definition in black boxes.

4.44 Definition *For each skeleton π_k in an $\mathsf{LK}^{\mathsf{vs}}$ expansion sequence, for each connection c_k^i in $\mathsf{NewConn}(\pi_k) = \{c_k^1, \ldots, c_k^{n_k}\}$, and each sequent s in π_k we define inductively an incremental constraint, denoted $\mathsf{C}_k^i(s)$, as follows.*

Initialization ($k = 0$ and $i = 0$): The skeleton π_0 contains a single sequent r. The constraint for r before any connections are taken into account, $\mathsf{C}_0^0(r)$, is defined as follows.

$$\mathsf{C}_0^0(r) := \emptyset$$

Inductive step 1 (increase of k and reset of i to 0): Assume that n_{k-1} is the number of new connections for π_{k-1}, and that $\mathsf{C}_{k-1}^{n_{k-1}}(s)$ is known for all sequents s in π_{k-1}. Let l' be the expanded leaf sequent of π_{k-1}. For all $k > 0$ and all sequents s in π_k we define $\mathsf{C}_k^0(s)$ as follows.

- *If s is* new *to π_k, then s is a premiss of the inference having l' as conclusion. We define*

$$\mathsf{C}_k^0(s) := \mathsf{C}_{k-1}^{n_{k-1}}(l').$$

- *Otherwise, s is also a sequent in π_{k-1} and we define*

$$\mathsf{C}_k^0(s) := \mathsf{C}_{k-1}^{n_{k-1}}(s).$$

Inductive step 2 (increase of i): Assume that $0 < i \leq n_k$, and that $\mathsf{C}_k^{i-1}(s)$ is known for all sequents s in π_k. Let B be the branch in π_k defined by $\mathsf{Leaf}(c_k^i)$. We define an increment set w.r.t. the connection c_k^i, denoted $\mathsf{INC}_k^i(s)$, for each s in π_k in the following way.

- *If s is* not *on B, then*
$$\mathsf{INC}^i_k(s) := \emptyset.$$

- *Otherwise, if $s = \mathsf{Leaf}(c^i_k)$, then*
$$\boxed{\mathsf{INC}^i_k(s) := \{\mathsf{Atom}(c^i_k)\}\ .}$$

- *Otherwise, if s is the conclusion of a non-branching inference with premiss s', then*
$$\mathsf{INC}^i_k(s) := \mathsf{INC}^i_k(s').$$

- *Otherwise, s is the conclusion of a branching inference θ. Let s' and s'' be the premisses of θ, and assume s' is on B. We define*
$$\mathsf{INC}^i_k(s) := \mathsf{INC}^i_k(s') \otimes \mathsf{C}^{i-1}_k(s'').$$

We define
$$\mathsf{C}^i_k(s) := \mathsf{C}^{i-1}_k(s) \cup \mathsf{INC}^i_k(s)\ .$$

Let r be the root sequent of π_k and n_k the number of new connections for π_k. The constraint for π_k, denoted $\mathsf{C}(\pi_k)$, is defined as
$$\boxed{\mathsf{C}(\pi_k) := \mathsf{Check}_{\pi_k}(\mathsf{C}^{n_k}_k(r))\ .}$$

In order to show correctness for incremental constraints for $\mathsf{LK}^{\mathsf{vs}}$-skeletons, we need a definition of what it means for a connection set to be spanning for a sequent in a skeleton.

4.45 Definition *Let π_k be a skeleton in an $\mathsf{LK}^{\mathsf{vs}}$ expansion sequence, let s be a sequent in π_k, let $\mathsf{NewConn}(\pi_k) = \{c^1_k, \ldots, c^{n_k}_k\}$, and let C be a set of connections C.*

- *C is spanning for s if it contains exactly one connection from each leaf sequent in $\mathsf{Lvs}_k(s)$.*

- *C is spanning for s up to c^i_k if it is spanning for s and does not contain any of the connections $c^{i+1}_k, \ldots, c^{n_k}_k$.*

The following lemma is needed in the correctness proof.

4.46 Lemma *Let π_k be a skeleton in an $\mathsf{LK}^{\mathsf{vs}}$ expansion sequence, let s be a sequent in π_k, let $\mathsf{NewConn}(\pi_k) = \{c^1_k, \ldots, c^n_k\}$, and let σ be a substitution. Then, σ satisfies $\mathsf{C}^i_k(s)$ if and only if σ satisfies $\mathsf{Prim}(C) \cup \mathsf{Bal}(C)$ for a set of connections C spanning for s up to c^i_k.*

PROOF By induction on the pair $\langle k, i\rangle$, and on sequent depth within each case. The base case $\langle 0, 0\rangle$ is trivial, so is the induction step $\langle k, n_k\rangle \to \langle 0, k+1\rangle$. In the induction step $\langle k, i\rangle \to \langle k, i+1\rangle$, the proof is trivial for leaf sequents (depth 0). The induction step depth $d \to d+1$ uses a similar argument as that used in Lemma 4.41 for global constraints. ∎

4.47 THEOREM *Let π_k be a skeleton in an $\mathsf{LK}^{\mathsf{vs}}$ expansion sequence. Then, $\mathsf{C}(\pi_k)$ is satisfiable if and only if π_k is closable.*

PROOF The proof is based on Lemma 4.46 and is similar to the correctness proof for global constraints. ∎

A proof search is terminated as soon as the constraint associated with the root sequent represents a closing substitution. Hence, the cycle check can be applied directly to the increment set for the root sequent for each new connection, i.e. $\mathsf{Check}_{\pi_k}(\mathsf{INC}^i_k(r))$.

4.48 EXAMPLE *Let π_4 be the skeleton in Figure 4.8. The root sequent*

$$\forall x Px \vdash ((Pa \wedge Pb) \wedge Pc) \wedge Pd$$

is valid. The incremental constraints for each sequent in the skeleton are displayed in the table in Figure 4.8. Let $\mathsf{NewConn}(\pi_4) = \{c_4^1, c_4^2\}$, where the connections are

$$\begin{array}{lcl} c_4^1 & : & Pu^{1.1}\{{}^{1}_{4}, {}^{1}_{5}, {}^{1}_{6}\} \vdash Pa^1 \\ c_4^2 & : & Pu^{1.1}\{{}^{1}_{4}, {}^{1}_{5}, {}^{1}_{7}\} \vdash Pb^1 \end{array}$$

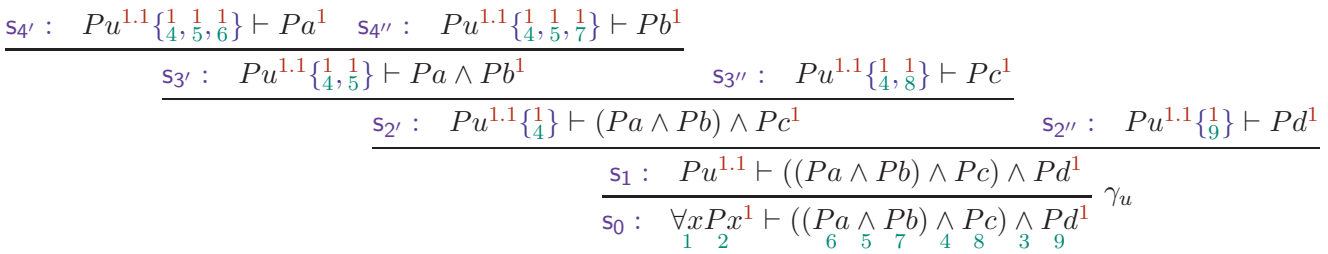
The increment set for the root sequent s_0 resulting from the new connection c_4^2 is

$$\begin{aligned} \mathsf{INC}^2_4(\mathsf{s}_0) &= \mathsf{INC}^2_4(\mathsf{s}_1) \\ &= \mathsf{INC}^2_4(\mathsf{s}_{2'}) \otimes \mathsf{C}^1_4(\mathsf{s}_{2''}) \\ &= \big(\mathsf{INC}^2_4(\mathsf{s}_{3'}) \otimes \mathsf{C}^1_4(\mathsf{s}_{3''})\big) \otimes \mathsf{C}^1_4(\mathsf{s}_{2''}) \\ &= \big(\big(\mathsf{C}^1_4(\mathsf{s}_{4'}) \otimes \mathsf{INC}(\mathsf{s}_{4''})\big) \otimes \mathsf{C}^1_4(\mathsf{s}_{3''})\big) \otimes \mathsf{C}^1_4(\mathsf{s}_{2''}) \\ &= \{\{u\{{}^{1}_{4}, {}^{1}_{5}, {}^{1}_{6}\} \approx a,\ u\{{}^{1}_{4}, {}^{1}_{5}, {}^{1}_{7}\} \approx b,\ u\{{}^{1}_{4}, {}^{1}_{8}\} \approx c,\ u\{{}^{1}_{9}\} \approx d\}\}\ . \end{aligned}$$

The substitution

$$\sigma = \{u\{{}^{1}_{4}, {}^{1}_{5}, {}^{1}_{6}\}/a,\ u\{{}^{1}_{4}, {}^{1}_{5}, {}^{1}_{7}\}/b,\ u\{{}^{1}_{4}, {}^{1}_{8}\}/c,\ u\{{}^{1}_{9}\}/d\}$$

is a most general unifier for the only atomic constraint in $\mathsf{INC}^2_4(\mathsf{s}_0)$. The index graph for $\ll_{\pi_4} \cup \prec_\sigma$ is acyclic, as illustrated in Figure 4.9.



	C^0_0	C^0_1	C^0_2	C^1_2	C^0_3	C^1_3	C^0_4	C^1_4	C^2_4
s_0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{uA \approx a, uB \approx b, uC \approx c, uD \approx d\}$
s_1	-	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{uA \approx a, uB \approx b, uC \approx c, uD \approx d\}$
$\mathsf{s}_{2'}$	-	-	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{uA \approx a, uB \approx b, uC \approx c\}$
$\mathsf{s}_{2''}$	-	-	$\emptyset$	$\{uD \approx d\}$					
$\mathsf{s}_{3'}$	-	-	-	-	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{uA \approx a, uB \approx b\}$
$\mathsf{s}_{3''}$	-	-	-	-	$\emptyset$	$\{uC \approx c\}$			
$\mathsf{s}_{4'}$	-	-	-	-	-	-	$\emptyset$	$\{uA \approx a\}$	
$\mathsf{s}_{4''}$	-	-	-	-	-	-	$\emptyset$	$\emptyset$	$\{uB \approx b\}$

Abbreviations:
$A = \{^{1}_{4},\,^{1}_{5},\,^{1}_{6}\}$
$B = \{^{1}_{4},\,^{1}_{5},\,^{1}_{7}\}$
$C = \{^{1}_{4},\,^{1}_{8}\}$
$D = \{^{1}_{9}\}$
$u = u^{1}_{1}$

Figure 4.8: A closable $\mathsf{LK}^{\mathsf{vs}}$-skeleton and a table showing the incremental constraints for each skeleton expansion and for each new connection. Contraction copies of γ-formulae are not displayed. For a description, see Example 4.48.

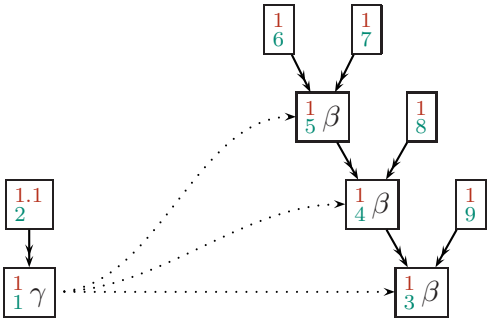


Figure 4.9: The acyclic index graph for the relation $\ll_{\pi_4} \cup \prec_\sigma$, where π_4 is the skeleton in Figure 4.8.

4.2.4 Incremental Cycle Check

The incremental constraints in Section 4.2.3 provide a way of calculating constraints for $\mathsf{LK}^{\mathsf{vs}}$-skeletons in parallel with skeleton expansion. When an increment set is generated for a new connection in a leaf sequent and propagated towards the root, the necessary balancing equations are generated at each β-inference by the merging operator. In comparison, cycle checking is done only when an increment set reaches the root sequent. I shall in this section outline an alternative constraint definition in which cycle checking of constraints is done at β-inferences by the merging operator.

As shown in Lemma 4.30, it is sufficient to cycle check a most general unifier for an equation set in order to determine whether the equation set represents closing substitutions for a skeleton. In Section 3.1.3 we defined equation sets in solved form and defined the MGU-function, which returns for each satisfiable equation set an equivalent equation set in solved form. Atomic constraints will in this section be defined as equation sets in solved form[2]. Since an equation set in solved form is an equational representation of a most general unifier for itself, the equations in the atomic constraint can be utilized directly in order to generate members of the $\prec$-relation. We define atomic constraints as follows.

4.49 Definition *The set of* atomic constraints *is the least set satisfying the following conditions.*

- *The symbol* $\bot$ *is an atomic constraint.*

[2]For now we ignore the fact that unification algorithms with equation sets in solved form as output have exponential worst case complexity. This issue is briefly addressed in Section 4.3.

- *A tuple $\langle S, \prec \rangle$, in which S is an equation set in solved form and $\prec$ is a binary relation on indices, is an atomic constraint.*

Constraints are defined in the same way as in Section 4.2.1. When two atomic constraints μ_1, μ_2 are merged in a β-inference, their equation sets are given as input to the MGU-function generating the equation set for $\mu_1 \otimes \mu_2$.

4.50 Lemma *Let σ be a substitution, and let u and v be variables in the domain of σ. If the terms $u\sigma$, $v\sigma$ are non-unifiable, then $u\sigma'$, $v\sigma'$ are non-unifiable for all substitutions σ' such that σ is more general σ'.*

Proof Assume for a contradiction that $u\sigma'$, $v\sigma'$ are unifiable. Then, $(u\sigma')\rho = (v\sigma')\rho$ for some substitution ρ. Since σ is more general than σ', $\sigma' = \sigma\tau$ for some substitution τ. But then $\tau\rho$ is a unifier for the non-unifiable terms $u\sigma$, $v\sigma$. ∎

If we view equation sets in solved form as substitutions, then the equation sets of μ_1 and μ_2 are both more general than the equation set of $\mu_1 \otimes \mu_2$. Hence, non-unifiability of terms $(uA)\sigma$, $(uB)\sigma$ is preserved when atomic constraints are propagated towards the root sequent. It is this monotonicity which makes an incremental cycle check worthwhile. In the same way as we generate necessary balancing equations, merging of atomic constraints must also generate the necessary members of the $\prec$-relation. The definition is relative to an $\mathsf{LK}^{\mathsf{vs}}$-skeleton π_k, since we need the $\ll_{\pi_k}$-relation in order to determine β-options and β-indices.

4.51 Definition *Let S, S_1 and S_2 be equation sets in solved form. Then, $\mathsf{Gen}(S, S_1, S_2)$ is the binary relation on indices such that $\langle i_1, i_2 \rangle \in \mathsf{Gen}(S, S_1, S_2)$ if and only if*

- *there are colored variables uA, uB in $\mathsf{Var}(S)$ such that the index of u is i_2 and the terms $S(uA)$, $S(uB)$ are non-unifiable, and*
- *the terms $S_1(uA)$, $S_1(uB)$ are unifiable and the terms $S_2(uA)$, $S_2(uB)$ are unifiable, and*
- *there are dual indices $i' \in A$, $i'' \in B$ such that $i_1 = \beta(i', i'')$.*

We must of course redefine the merging operator in order to incorporate cycle checking and incremental generation of the $\prec$-relation. Again, the definition is relative to an $\mathsf{LK}^{\mathsf{vs}}$-skeleton.

4.52 Definition (Merging) *Let μ_1 and μ_2 be atomic constraints. The merging of μ_1 and μ_2, denoted $\mu_1 \otimes \mu_2$, is defined as follows.*

- *If* $\mu_1 = \bot$ *or* $\mu_2 = \bot$*, then*

$$\mu_1 \otimes \mu_2 := \bot \, .$$

- *Otherwise, let* $\mu_1 = \langle S_1, \prec_1 \rangle$*,* $\mu_2 = \langle S_2, \prec_2 \rangle$ *and* $S = \mathsf{MGU}(S_1 \cup S_2 \cup \mathsf{Bal}(S_1, S_2))$*. If* $S = \bot$*, then*

$$\mu_1 \otimes \mu_2 := \bot \, .$$

- *Otherwise, let* $\prec = \prec_1 \cup \prec_2 \cup \mathsf{Gen}(S, S_1, S_2)$*. If the index graph for* $\ll_\pi \cup \prec$ *contains a cycle, then*

$$\mu_1 \otimes \mu_2 := \bot \, .$$

- *Otherwise,*

$$\mu_1 \otimes \mu_2 := \langle S, \prec \rangle \, .$$

The merging operator is extended to constraints in the same way as in Definition 3.25.

Incremental constraints incorporating the new cycle checking merging operator are defined like in Definition 4.44, except that the cycle check in the root node is no longer necessary. Figure 4.10 shows part of an $\mathsf{LK}^{\mathsf{vs}}$-skeleton π_k obtained by expanding a leaf sequent l in π_{k-1}. The β-inference is on the path from l to the root sequent of the skeleton. The increment set for s is the result of merging the increment set for s' with the incremental constraint for s''.

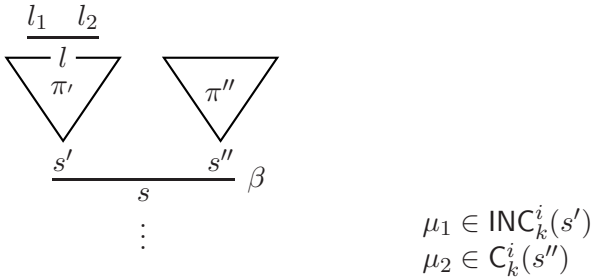


Figure 4.10: Part of an $\mathsf{LK}^{\mathsf{vs}}$-skeleton π_k. The β-inference is on the path from the expanded leaf sequent l of π_{k-1} to the root sequent. The increment set for s is the result of merging the increment set of s' with the incremental constraint for s''.

Let $\mu_1 = \langle S_1, \prec_1 \rangle$ and $\mu_2 = \langle S_2, \prec_2 \rangle$ be atomic constraints in $\mathsf{INC}^i_k(s')$ and $\mathsf{C}^i_k(s'')$, respectively. The equation set component S of $\mu_1 \otimes \mu_2$ is generated in the same way as with the merging operator in Section 4.2.1. If S is not

satisfiable, then $\mu_1 \otimes \mu_2$ is $\bot$. Otherwise, a relation $\prec$ is generated, which contains $\prec_1$ and $\prec_2$ as subsets. This is justified by the monotonicity property mentioned earlier in this section; if the terms $S_j(uA)$, $S_j(uB)$ for $j \in \{1,2\}$ are non-unifiable, then so are $S(uA)$, $S(uB)$, since S_j is more general than S. In addition, $\prec$ contains new members resulting from the merging of S_1 and S_2. Let uA, uB be colored variables such that $i_1 \in A$, $i_2 \in B$ are dual indices. If the terms $S_1(uA)$, $S_1(uB)$ and the terms $S_2(uA)$, $S_2(uB)$ are unifiable and the terms $S(uA)$, $S(uB)$ are non-unifiable, then uA, uB give rise to a member of $\prec$ which is not in $\prec_1$ or $\prec_2$. The Gen-function generates exactly those members. Finally, the relation $\prec$ is cycle checked. If it contains a cycle, then $\mu_1 \otimes \mu_2$ is set to $\bot$. As a result, $\mu_1 \otimes \mu_2$ is satisfiable only if the $\prec$-relation is acyclic.

The new definition of atomic constraints requires a new function for generating atomic constraints from connections in leaf sequents. As the following lemma suggests, we do not have to generate the $\prec$-relation when generating atomic constraints for connections.

4.53 Lemma *Let π be an $\mathsf{LK}^{\mathsf{vs}}$-skeleton, let $C = \{c\}$ be a connection set, and let σ be a substitution. Then, $\prec_\sigma$ wrt. C is empty.*

Proof A connection c is a sequent of the form $P\vec{s}A \vdash P\vec{t}B$. The corresponding colored connections is $P\vec{s} \oplus (A \setminus B) \vdash P\vec{t} \oplus (B \setminus A)$. The colors $(A \setminus B)$ and $(B \setminus A)$ are subsets of the splitting set for the branch the connection is in. By Lemma 4.20 the splitting set for the branch contains no dual indices, and thus there are no dual indices in the two colors. Then, by Definition 4.24 $\prec_\sigma$ wrt. $\{c\}$ is empty. ∎

We define the Atom-function as follows.

4.54 Definition *For each connection c we define an atomic constraint, denoted $\mathsf{Atom}(c)$, as follows. Let $S = \mathsf{MGU}(\mathsf{Prim}(c) \cup \mathsf{Bal}(c))$.*

- *If $S = \bot$, then*
$$\mathsf{Atom}(c) := \bot \ .$$
- *Otherwise,*
$$\mathsf{Atom}(c) := \langle S, \emptyset \rangle \ .$$

4.55 Conjecture *Let π_k be an $\mathsf{LK}^{\mathsf{vs}}$-skeleton. Then, $\mathsf{C}(\pi_k)$ is satisfiable if and only if π_k is closable.*

The correctness proof is done by induction on $\langle k, i \rangle$ and the sequent depth, similar to the proof for the incremental constraints in Section 4.2.3 . The only problem is the generation of balancing equations. In the constraint definition where the atomic constraints are equations sets, *all* colored variables

occurring in a spanning connection set is also in the corresponding atomic constraint. This is however not the case when equation sets are represented as equivalent equation sets in solved form. Let $\mathsf{Prim}(C)$ contain trivial equations $uA \approx uA$, $uB \approx uB$ such that the colored variables uA, uB do not occur in any other equation in $\mathsf{Prim}(C)$, and let S be the equation set in solved form corresponding to $\mathsf{Prim}(C)$. Then, uA, uB are in $\mathsf{Var}(C)$ but *not* in $\mathsf{Var}(S)$. If $A \boxdot B = B \boxdot A$, then the balancing equation $uA \approx uB$ is included when generating balancing equations for C. Since the variables uA, uB are not in $\mathsf{Var}(S)$, we have no guarantee that $uA \approx uB$ is solved by a most general unifier for S.

The equations $uA \approx uA$, $uB \approx uB$ are trivial, and intuitively they should not contribute to restrictions put upon closing substitutions. This may suggest that the definition of closing substitutions for $\mathsf{LK}^{\mathsf{vs}}$ is too strict and that the above described problem can be solved by a new definition of closing substitutions for the calculus.

4.3 Concluding Remarks and Future Work

The variable splitting calculus $\mathsf{LK}^{\mathsf{vs}}$ [1, 32] is a free variable sequent calculus generating variable sharing skeletons, which provide invariance under order of rule application and facilitates connection driven proof search [31]. Due to the context sharing rules of the calculus, instantiation variables are copied into different branches in a skeleton by β-inferences. With rigid variable instantiation, every occurrence of a variable is instantiated with the same term. This prohibits branchwise restriction of the term search space, and thus early detection of unprovability in some cases. By means of an indexing system and branch labels for formulae, the calculus $\mathsf{LK}^{\mathsf{vs}}$ labels different branch occurrences of a variable differently. Thus, substitutions may instantiate different branch occurrences of a variable with different terms. This is called variable splitting.

Naive instantiation of variable occurrences easily leads to an unsound calculus, as is illustrated by the examples in Section 4.1.5. Thus, in order for a substitution to close an $\mathsf{LK}^{\mathsf{vs}}$-skeleton, it must have additional properties besides satisfying the primary equations generated from a spanning connection set. The requirements are defined in terms of *balancing equations* and *dependency relations* induced by substitutions on $\mathsf{LK}^{\mathsf{vs}}$-skeletons.

In Chapter 3 the concept of constraints was introduced, inspired by the incremental closure framework of Giese [18]. Constraints are syntactic objects generated from unification problems corresponding to the closure condition of a calculus. Constraint semantics is defined by means of a satisfiability relation between constraints and substitutions. With an inductive constraint

definition for sequents in skeletons, the question of whether a skeleton is closable can be expressed as a satisfiability check for the constraint associated with the root sequent. Further, the inductive constraint definition facilitates an *incremental* calculation of constraints alongside skeleton expansion.

Skeleton expansion may introduce new connections in the leaf sequents of a skeleton. For each new connection, an increment set is defined recursively for the root sequent, following the structure of the skeleton. Hence, a proof search can be terminated as soon as the increment set for the root sequent wrt. a new connection is satisfiable.

The requirements put upon closing substitutions for $\mathsf{LK}^{\mathsf{vs}}$-skeletons is more strict than those for LK^{v}-skeletons. This affects the way constraints for $\mathsf{LK}^{\mathsf{vs}}$-skeletons are defined. In addition to the primary equations for partially spanning connection sets, constraints must represent the corresponding balancing equations. This is solved by letting atomic constraints consist of both primary and secondary equations. In this way, the satisfiability of both equation sets is tested when constraints are merged at β-inferences. The merging operator must however generate *new* balancing equations resulting from the merging process. In addition, a constraint definition for $\mathsf{LK}^{\mathsf{vs}}$ must have a way of *cycle checking* the dependency relation induced by the equation set.

In Section 4.2.3 I presented an incremental constraint definition for $\mathsf{LK}^{\mathsf{vs}}$-skeletons and showed that it is *correct*, i.e. that the incremental constraint associated with a skeleton is satisfiable if and only if the skeleton is closable. However, with this definition the cycle check is performed only at the root sequent of a skeleton. Hence, while primary and secondary equations are generated *incrementally* by constraint merging at each β-inference, the dependency relation is generated and cycle checked *globally* at the root sequent. The cycle check will however only be preformed whenever a satisfiable atomic constraint reaches the root sequent.

I addressed this problem in Section 4.2.4, motivated by the fact that the dependency relation generated for an atomic constraint grows monotonically when the constraint is propagated towards the root sequent. A new constraint language was introduced, in which atomic constraints consist of an equation set *in solved form* and a dependency relation. The merging operator was redefined in order to generate new members of dependency relations resulting from constraint merging. Also, the cycle checking is performed at each β-inference. Hence, the cycle check is performed incrementally, and a satisfiable constraint reaches the root sequent only if the induced dependency relation is acyclic. Another effect of this approach is that when two atomic constraints are merged, the input to the unification algorithm consists of two equation sets, which are already in solved form, and a set of balancing equations. In this way, the work done by a unification at one β-inference can be utilized by a unification further below on the branch.

The constraint definitions given in this chapter are by no means the final answer to how incremental proof search in the splitting calculus should be conducted. There are several questions to be answered, some of which are listed below.

Dag solved form. The definition of equation set in solved form used throughout this thesis results in a unification algorithm with exponential worst case complexity [23]. Since unification is performed very frequently during an incremental proof search, it is desirable to reduce the running time of unification. In the incremental constraint definition in Section 3.2.3, the unifier itself is of no importance. Here we are merely interested in the existence of a unifier. On the contrary, the incremental constraint definitions in Chapter 4 utilizes the equivalence between equation sets in solved form and most general unifiers in order to calculate dependency relations in the cycle check. The efficiency of the unification algorithm in [27] is partly due to the use of *dag solved form*, in which a certain reuse of terms is introduced in order to reduce the worst case space requirement. I believe that it is possible to define the equation sets of atomic constraints as dag solved forms, and define the operations which generates dependency relations as operations on dag solved forms.

Complexity analysis. A proof search procedure utilizing an incremental constraint definition is undoubtedly far more efficient than the naive procedure incorporating a global closure check after each expansion step. Exactly how much more efficient is it? How does it compare to other proof search procedures? The answers to these questions are closely related to the complexity of the splitting calculus itself. Variable splitting reduces the need for expanding contraction copies of γ-formulae. A variable splitting proof search procedure will perform at its best on valid input sequents which require heavy re-expansion of γ-formulae in a calculus without variable splitting.

Implementation. A prototype implementation will indicate how well the search procedure performs in practice, and it would facilitate fine-tuning of the underlying algorithms and data structures. Which approach performs best in practice; the incremental or the global cycle check?

Dependency relation ($\prec$) represented as equations. It is worth investigating whether the dependencies expressed by the $\prec$-relation can be expressed with disjunctive equations or sets of equations. In Example 4.34 the cycle in the index graph is introduced because the two variables are split by an ancestor of the γ-formula introducing the other variable. Equations expressing that a closing substitution may split one of the variables but not both would have prevented closure in this case. Can this be applied in general?

New closure conditions for $\mathsf{LK}^{\mathsf{vs}}$. As mentioned in the introduction to this chapter, the variable splitting calculus has not reached its final form.

Further research on this field will probably result in a refined closure condition for the calculus, which may result in improvements of the search procedure.

Other logics. Work is underway with the goal of adapting the variable splitting technique to sequent calculi for intuitionistic and modal logic among others. Is it possible to design efficient proof search procedures for such calculi based on incremental constraint definitions?

I sincerely hope to continue working with these topics and, hopefully, contribute to shed light on some of the open questions. Finally, I close this thesis with the saying of a Buddhist Master in the film *Himalaya*:

> *"Whenever there are two trails open in front of you, always choose the hardest one; the one which will squeeze the best out of you."*

Appendix A

Listing of the $\mathsf{LK}^{\mathsf{vs}}$-rules

See Figure A.1 on page 86.

α-rules

$$\frac{\Gamma, \varphi^\kappa A, \psi^\kappa A \vdash \Delta}{\Gamma, (\varphi \wedge \psi)^\kappa A \vdash \Delta}\ \mathsf{L}\wedge \qquad \frac{\Gamma \vdash \varphi^\kappa A, \psi^\kappa A, \Delta}{\Gamma \vdash (\varphi \vee \psi)^\kappa A, \Delta}\ \mathsf{R}\vee$$

$$\frac{\Gamma, \varphi^\kappa A \vdash \psi^\kappa A, \Delta}{\Gamma \vdash (\varphi \to \psi)^\kappa A, \Delta}\ \mathsf{R}\to \qquad \frac{\Gamma \vdash \varphi^\kappa A, \Delta}{\Gamma, (\neg\varphi)^\kappa A \vdash \Delta}\ \mathsf{L}\neg$$

$$\frac{\Gamma, \varphi^\kappa A \vdash \Delta}{\Gamma \vdash (\neg\varphi)^\kappa A, \Delta}\ \mathsf{R}\neg$$

β-rules

$$\frac{\Gamma^{\varphi^\kappa} \vdash \varphi^\kappa A, \Delta^{\varphi^\kappa} \quad \Gamma^{\psi^\kappa} \vdash \psi^\kappa A, \Delta^{\psi^\kappa}}{\Gamma \vdash (\varphi \wedge \psi)^\kappa A, \Delta}\ \mathsf{R}\wedge$$

$$\frac{\Gamma^{\varphi^\kappa}, \varphi^\kappa A \vdash \Delta^{\varphi^\kappa} \quad \Gamma^{\psi^\kappa}, \psi^\kappa A \vdash \Delta^{\psi^\kappa}}{\Gamma, (\varphi \vee \psi)^\kappa A \vdash \Delta}\ \mathsf{L}\vee$$

$$\frac{\Gamma^{\varphi^\kappa} \vdash \varphi^\kappa A, \Delta^{\varphi^\kappa} \quad \Gamma^{\psi^\kappa}, \psi^\kappa A \vdash \Delta^{\psi^\kappa}}{\Gamma, (\varphi \to \psi)^\kappa A \vdash \Delta}\ \mathsf{L}\to$$

δ-rules

$$\frac{\Gamma \vdash \varphi[x/f_m\vec{u}]^\kappa A, \Delta}{\Gamma \vdash \forall x\varphi^\kappa A, \Delta}\ \mathsf{R}\forall \qquad \frac{\Gamma, \varphi[x/f_m\vec{u}]^\kappa A \vdash \Delta}{\Gamma, \exists x\varphi^\kappa A \vdash \Delta}\ \mathsf{L}\exists$$

γ-rules

$$\frac{\Gamma, \forall x\varphi^{\kappa'} A, \varphi[x/u_m^\kappa]^{\kappa.1} A \vdash \Delta}{\Gamma, \forall x\varphi^\kappa A \vdash \Delta}\ \mathsf{L}\forall \qquad \frac{\Gamma \vdash \exists x\varphi^{\kappa'} A, \varphi[x/u_m^\kappa]^{\kappa.1} A, \Delta}{\Gamma \vdash \exists x\varphi^\kappa A, \Delta}\ \mathsf{R}\exists$$

Figure A.1: A complete listing of the $\mathsf{LK}^{\mathsf{vs}}$-rules. Legend: κ is a *copy history* (see page 21), A is a *splitting set* (see page 57), m is the *formula number* of the principal formula (see page 21), f_m is a *Skolem function* and u_m^κ is an *instantiation variable* (see page 19), $\vec{u}$ is the set of instantiation variables occurring in the principal formula (see page 22), $\kappa.1$ denotes the concatenation of κ and 1, and κ' denotes the copy history like κ except that the last element is increased by one (see page 22). Γ^{φ^κ} denotes the set of decorated formulae obtained by adding the index of φ^κ to the splitting set of every decorated formula in Γ, see page 59.

Bibliography

[1] Roger Antonsen. Free variable sequent calculi. Master's thesis, University of Oslo, Language, Logic and Information, Department of Linguistics, May 2003.

[2] Roger Antonsen. Uniform variable splitting. In *Contributions to the Doctoral Programme of the Second International Joint Conference on Automated Reasoning (IJCAR 2004), Cork, Ireland, 04 July - 08 July, 2004*, volume 106, pages 1–5. CEUR Workshop Proceedings, 2004. Online: `http://ceur-ws.org/Vol-106/01-antonsen.pdf`.

[3] Roger Antonsen. Report on the splitting calculus. Unpublished draft, University of Oslo, Department of Informatics, 2004-2005.

[4] Matthias Baaz and Christian. G. Fermüller. Non-elementary speedups between different versions of tableaux. In Reiner Hähnle Peter Baumgartner and J. Possega, editors, *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, volume 918 of *LNAI*, pages 217–230. Springer, 1995.

[5] Bernhard Beckert. *Tableau-based Theorem Proving: A Unified View.* PhD thesis, Universität Karlsruhe, Fakultät für Informatik, Institut für Logik, Komplexität und Deduktionssysteme, 1998.

[6] Bernhard Beckert. Depth-first proof search without backtracking for free variable semantic tableaux. In Neil Murray, editor, *Position Papers, International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Saratoga Springs, NY, USA*, Technical Report 99-1, pages 33–54. Institute for Programming and Logics, Department of Computer Science, University at Albany – SUNY, 1999.

[7] Bernhard Beckert and Reiner Hähnle. An improved method for adding equality to free variable semantic tableaux. In D. Kapur, editor, *Proceedings, 11th International Conference on Automated Deduction (CADE), Saratoga Springs, NY*, LNCS 607, pages 507–521. Springer, 1992.

[8] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. The *even more* liberalized δ-rule in free variable semantic tableaux. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Proceedings of the third Kurt Gödel Colloquium KGC'93, Brno, Czech Republic*, volume 713 of *LNCS*, pages 108–119. Springer-Verlag, August 1993.

[9] Wolfgang Bibel. Computationally improved versions of Herbrand's theorem. In *Logic Colloquium '81*, pages 11–28. North-Holland, 1982.

[10] Wolfgang Bibel. *Automated Theorem Proving 2. Edition.* Vieweg Verlag, 1987.

[11] Susanna S. Epp. *Discrete mathematics with applications.* PWS Publishing Company, second edition, 1995.

[12] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving.* Graduate Texts in Computer Science. Springer-Verlag, Berlin, 2nd edition, 1996. 1st ed., 1990.

[13] Jean H. Gallier. *Logic for computer science: foundations of automatic theorem proving.* Harper & Row Publishers, Inc., 1985.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[15] Gerhard Gentzen. 'Untersuchungen über das logische Schließen'. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934-1935. English translation in M.E. Szabo *The Collected Papers of Gerhard Gentzen*, North-Holland, Amsterdam, 1969.

[16] Martin Giese. A first-order simplification rule with constraints. In *Proc. Int. Workshop on First-Order Theorem Proving, St. Andrews, Scotland*, 2000.

[17] Martin Giese. Incremental Closure of Free Variable Tableaux. In *Proc. Intl. Joint Conf. on Automated Reasoning, Siena, Italy*, number 2083 in LNCS, pages 545–560. Springer-Verlag, 2001.

[18] Martin Giese. *Proof Search without Backtracking for Free Variable Tableaux.* PhD thesis, Fakultät für Informatik, Universität Karlsruhe, July 2002.

[19] Jean-Yves Girard. *Proof Theory and Logical Complexity.* Bibliopolis, Naples, 1987.

[20] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types.* Cambridge University Press, 1989.

[21] Reiner Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 3, pages 100–178. Elsevier Science, 2001.

[22] Reiner Hähnle and Peter H. Schmitt. The liberalized δ-rule in free variable semantic tableaux. *Journal of Automated Reasoning*, 13(2):211–222, oct 1994.

[23] Claude Kirchner. Constraint solving on terms: Syntactic methods, 1999.

[24] Stephen Cole Kleene. *Introduction to Metamathematics*. Wolters-Noordhoff, Groningen, 1971.

[25] Christoph Kreitz and Jens Otten. Connection-based theorem proving in classical and non-classical logics. *Journal of Universal Computer Science*, 5(3):88–112, 1999.

[26] Alberto Martelli and Ugo Montanari. Unification in linear time and space: A structured presentation. Technical Report B76-16, Ist. di Elaborazione delle Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, July 1976.

[27] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[28] Mike Paterson and Mark N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, April 1978.

[29] Raymond M. Smullyan. *First-Order Logic*, volume 43 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer-Verlag, New York, 1968.

[30] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2nd edition, 2000. 1st ed. 1996.

[31] Arild Waaler. Connections in nonclassical logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 22, pages 1487–1578. Elsevier Science, 2001.

[32] Arild Waaler and Roger Antonsen. A free variable sequent calculus with uniform variable splitting. In M. C. Mayer and F. M. Pirri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference, TABLEAUX 2003, Rome, Italy, September 9-12, 2003 Proceedings*, volume 2796 of *LNCS*, pages 214–229. Springer-Verlag, September 2003.

[33] Lincoln A. Wallen. *Automated deduction in nonclassical logics*. MIT Press, 1990.

[34] Mark Allen Weiss. *Data Structures & Algorithm Analysis in Java.* Addison-Wesley, 1999.

Index