INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

A Simple Deduction System for First-Order Logic with Equality, Free Constructors and Induction

Jean Goubault-Larrecq

N° 3653

Mars 1999

THÈME 2

Rapport de recherche

ISSN 0249-6399

A Simple Deduction System for First-Order Logic with Equality, Free Constructors and Induction

Jean Goubault-Larrecq*

Thème 2 — Génie logiciel
et calcul symbolique

Projet Coq

Rapport de recherche n°3653 — Mars 1999

Abstract: Proof systems like Coq feature inductive datatypes, where datatype constructors are *free* in the sense that two terms built from constructors only are semantically equal if and only if they are syntactically identical. Although free constructors are an essential ingredient of modern formalized mathematics, no automated first-order prover has been specialized with built-in rules for dealing with free constructors, until now. We propose a sequent system for a logic where terms can be built only from variables and free constructors. Thus the logic will be kept simple, as equality in the logic will be syntactical equality. We show how partial functions can be introduced into the logic, in the form of predicates obeying some functionality constraints. We prove that the resulting system is sound and complete with respect to a natural first-order semantics of datatypes, and enjoys cut elimination. We then develop a tableau calculus to find proofs in this sequent system. This involves solving an extended form of unification, which unfortunately is undecidable: we show that it indeed includes second-order predicate unification as a subproblem. Nonetheless, we describe a non-terminating procedure to solve such unification problems, and give a few hints so as to improve its efficiency in practice. Finally, we extend the system to include first-order structural induction principles on datatype values. This does not pose any difficulty, apart from the expected problem of induction loading—namely, that we may need to generalize a proposition before we can prove it by induction.

Key-words: automated deduction, constructors, Coq, datatypes, equality, first-order logic, induction, partial functions, second-order unification, sequent system, undecidability, unification.

(Résumé : tsvp)

This work has been done in the context of Dyade (R&D joint venture between Bull and Inria).

* Jean.Goubault@{dyade,inria}.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

Un système de preuve simple pour la logique du premier ordre avec égalité, constructeurs libres et récurrence

Résumé : Les systèmes de preuve comme Coq proposent des types de données inductifs, où les constructeurs de données sont *libres* au sens où deux termes construits uniquement à l'aide de constructeurs sont sémantiquement égaux si et seulement s'ils sont syntaxiquement égaux. Bien que les constructeurs libres soient un ingrédient essentiel des mathématiques formalisées modernes, aucun prouver automatique au premier ordre n'a été spécialisé jusqu'ici par des règles intégrées au prouveur pour traiter des constructeurs libres. Nous proposons un système de séquents pour une logique où les termes peuvent être construits uniquement à partir de variables et de constructeurs libres. Ainsi la logique sera-t-elle forcée à rester simple, car l'égalité dans la logique sera l'égalité syntaxique. Nous montrons comment l'on peut ajouter des fonctions (partielles) dans la logique, sous la forme de prédicats obéissant à certaines contraintes de fonctionnalité. Nous prouvons que le système qui en résulte est correct et complet par rapport à une sémantique naturelle des types de données au premier ordre, et admet l'élimination des coupures. Nous développons ensuite un calcul de tableaux pour trouver des preuves dans ce système de séquents. Ceci demande de résoudre une forme d'unification étendue, qui est malheureusement indécidable : nous montrons qu'elle inclut en effet l'unification prédicative du second ordre comme cas particulier. Néanmoins, nous décrivons une procédure non terminante qui résout de tels problèmes d'unification, and donnons quelques idées pour améliorer son efficacité en pratique. Finalement, nous étendons le système pour qu'il inclue des principes de récurrence structurelle du premier ordre sur les valeurs des types de données. Ceci ne pose pas de difficulté, si l'on exclut le problème bien connu du chargement des récurrences — à savoir qu'il faut éventuellement généraliser une proposition avant de pouvoir la prouver par récurrence.

Mots-clé : Coq, démonstration automatique, égalité, fonctions partielles, indécidabilité, logique du premier ordre, récurrence, types de données, unification, unification du second ordre.

Introduction

First-order logic with equality is one of the most pervasive logics in use, whether in mathematics, logic or computer science. For real applications, however, automated provers need to be complemented with axioms of interest, e.g. axioms for arithmetic, or for associative-commutative operations. It is then interesting to build these axioms inside the prover itself, for performance reasons. This has been done for a variety of theories, whether equational [Plo72] or not [Sti85]. Recently, Baaz, Egly and Fermüller [BEF97] have proposed to build in various forms of induction in tableaux theorem provers, by coding minimality principles as Skolem functions.

One form of theory that has not yet been investigated is that of *datatype constructors*. Consider for instance the specification of the type `nat` of unary integers and `natlist` of lists of unary integers in Standard ML [HMT90]:

```
datatype nat = O | S of nat;
datatype natlist = nil | cons of nat * natlist;
```

In first-order logical form, these declarations specify a many-sorted signature consisting of a constant `O` of sort `nat`, a unary function symbol `S` of signature $\mathtt{nat}\to\mathtt{nat}$, a constant `nil` of sort `natlist` and a binary function symbol `cons` of signature $\mathtt{nat}\times\mathtt{natlist}\to\mathtt{natlist}$. (We make here a little abuse, in that these declarations actually declarer higher-order constants, not first-order function symbols.) These declarations also imply that the only ground terms of sort `nat` are those built up using `O` and `S`, and that the only ground terms of sort `natlist` are those built up using `nil` and `cons` atop ground terms of sort `nat`. The latter can be expressed by structural induction schemes:

$$\begin{array}{rl}\forall P_{\mathtt{nat}\to o}\cdot & P(\mathtt{O}) \land (\forall n_{\mathtt{nat}} \cdot P(n) \supset P(\mathtt{S}(n))) \supset \forall n_{\mathtt{nat}} \cdot P(n) \\ \forall P_{\mathtt{natlist}\to o}\cdot & P(\mathtt{nil}) \land (\forall n_{\mathtt{nat}} \cdot \forall \ell_{\mathtt{natlist}} \cdot P(\ell) \supset P(\mathtt{cons}(n,\ell))) \\ & \supset \forall \ell_{\mathtt{natlist}} \cdot P(\ell)\end{array}$$

But these declarations also imply that distinct ground terms always denote distinct values, something which can be described by Peano-like axioms which we shall call *non-confusion axioms*:

$$\begin{array}{rl}\forall n_{\mathtt{nat}}\cdot & \neg \mathtt{S}(n) \approx \mathtt{O} \\ \forall m_{\mathtt{nat}}, n_{\mathtt{nat}}\cdot & \mathtt{S}(m) \approx \mathtt{S}(n) \supset m \approx n \\ \forall \ell_{\mathtt{natlist}}, m_{\mathtt{nat}}\cdot & \neg \mathtt{cons}(m,\ell) \approx \mathtt{nil} \\ \forall \ell_{\mathtt{natlist}}, \ell'_{\mathtt{natlist}}, m_{\mathtt{nat}}, n_{\mathtt{nat}}\cdot & \mathtt{cons}(m,\ell) \approx \mathtt{cons}(n,\ell') \supset \ell \approx \ell' \land m \approx n\end{array}$$

where $\approx$ is the equality symbol, plus a few other axioms stating that there are no cycles, e.g. $\forall n_{\mathtt{nat}} \cdot \neg n \approx \mathtt{S}(n)$. (If we have induction, then the latter are derivable from the former; in general, see [Mah88].) Constants and functions like `O`, `S`, `nil`, `cons` that satisfy these distinctness conditions are usually referred to as *constructors* of the datatypes `nat` and `natlist`.

These theories are very natural, and have been considered elegant foundations for inductive datatypes both in programming language design and in proof assistants. As far as the latter are concerned, let us say that the exposition above is essentially what a proof assistant like Coq [BBC+97], based on the Calculus of Inductive Constructions, understands by inductive datatypes of naturals and lists of naturals. (We say "essentially", because the interaction with dependent types in Coq allows for considerably more complex inductive type declarations.) Recently, practical frameworks for analyzing and proving properties of cryptographic protocols [Bol96, Pau97] also used similar inductive definitions for messages exchanged over communication lines; messages are typically described as the following ML datatype `msg`:

```
datatype base = K of key | ...;
datatype msg = B of base | P of msg * msg | C of msg * key;
```

where `key` is a previously defined datatype of encryption keys, and the informal ellipsis `...` is meant to indicate that there may be other `base` messages than keys—i.e., there is no induction principle on `base`. Note that encrypting a message m with a key k is done by applying the constructor $\mathtt{C}$, *viz.* $\mathtt{C}(m, k)$; that $\mathtt{C}$ is a constructor in particular automatically implies the desired features that an encrypted message cannot be confused with a pair or a base message ($\neg\mathtt{C}(m, k) \approx \mathtt{P}(m_1, m_2)$, $\neg\mathtt{C}(m, k) \approx \mathtt{B}(b)$), that there are no semantical overlaps between encrypted messages ($\mathtt{C}(m_1, k_1) \approx \mathtt{C}(m_2, k_2) \supset m_1 \approx m_2 \wedge k_1 \approx k_2$), and that there are no cycles, in particular that we cannot decipher a message by encrypting it repetitively ($\neg\mathtt{C}(\mathtt{C}(\ldots\mathtt{C}(m, k_1)\ldots, k_{n-1}), k_n) \approx m$).

Although the notion of free constructors is a natural one, working with non-confusion axioms atop a generic first-order prover is awkward at best, and it would be fruitful to build such axioms in automated provers. This is all the more frustrating, as the semantics of constructors is extremely simple: just take any (many-sorted) Herbrand model, and insist that equality denote exactly the identity on ground terms. Solving existentially quantified equations, for one, is just first-order unification [Rob65], a well-understood problem that has very efficient algorithmic solutions [MM82]; compare this with the rather more complicated inversion tactics needed in Coq to do basic reasoning with non-confusion.

We introduce a new sequent system for theories with equality and constructors, with non-confusion axioms built in the sequent system itself, as a basis for automated theorem provers in such theories. This sequent system accommodates various forms of induction, structural or not. It has the advantage of being remarkably simple; notably, as equality reasoning is entirely based on properties of substitutions on terms, and on the existence and uniqueness of most general unifiers, this allows us to insist that any proof of an equality $u \approx v$ be by reflexivity—i.e., $u \approx v$ is provable if and only if u and v are the same term. This may come as a surprise to the reader that this is enough to get a complete enough theory (in particular, it can prove every theorem of ordinary first-order logic with equality), but it is in the spirit of Martin Streicher and Thorsten Altenkirch's discovery that, in type theory, it is consistent to admit the principle that all proofs of $u \approx v$ be equal to some reflexivity proof [Str93].

The plan of the paper is as follows: we introduce all preliminary notions and notations in Section 1, and proceed to explain how we came to the actual rules of our sequent system $\mathrm{LKc}_\approx$ in Section 2. $\mathrm{LKc}_\approx$ encodes a sound, complete proof system for first-order logic with equality and constructors. We show this in Section 3, as well as the fact that cuts can be eliminated. The latter is indispensable if we are to infer a practical tableau system from $\mathrm{LKc}_\approx$: we do just this in Section 4. A proof is found when we have found a closed tableau for the input formula, and provided that all unification constraints (generated by closing rules, notably) are satisfiable. In the same section, we examine the problem of constraint generation, and constraint solving: we show that constraint solving is an undecidable problem, and give a sound and complete procedure for solving constraints. We consider the extension of $\mathrm{LKc}_\approx$ to a system $\mathrm{LKc}^{ind}_\approx$ that handles structural induction in Section 5; this does not add much complication to $\mathrm{LKc}_\approx$. We give a short tour of related ideas in Section 6, and conclude in Section 7.

1 Preliminaries

We first assume a given finite collection $\mathcal{S}$ of sorts τ, τ', τ_1, …, and a fixed collection $\mathcal{C}$ of function symbols c, d, …, which we shall call *constructors*. Each constructor c is given a unique *arity* $\alpha(c)$, which is an expression of the form $\tau_1 \times \ldots \times \tau_n \to \tau$. Let $\mathcal{V}$ be a set of so-called *variables* x_τ, y_τ, z_τ, …; we assume that for each sort τ, there are infinitely many variables with τ as a subscript. The *pre-terms* are given by the following grammar:

$$t ::= x_\tau \mid c(t, \ldots, t)$$

We define a *typing judgment* $\triangleright t : \tau$ by the following rules:

$$\frac{}{\triangleright x_\tau : \tau} \qquad \frac{\triangleright t_1 : \tau_1 \quad \ldots \quad \triangleright t_n : \tau_n \quad \alpha(c) = \tau_1 \times \ldots \times \tau_n \to \tau}{\triangleright c(t_1, \ldots, t_n) : \tau}$$

We call *terms* the pre-terms t such that $\triangleright t : \tau$ for some sort τ. It is easy to see that, in this case, τ is unique; we shall call it the *type* of the term t. In case $n = 0$, we write c instead of $c()$, and also write $\alpha(c)$ as τ instead of $\rightarrow \tau$. When the type τ of x_τ is understood, or when it does not matter, we shall also drop the sort subscript and write x.

The set $\mathrm{fv}(t)$ of *free variables* of t is defined by $\mathrm{fv}(x) =_{\mathrm{df}} \{x\}$ and $\mathrm{fv}(c(t_1, \ldots, t_n)) =_{\mathrm{df}} \bigcup_{i=1}^n \mathrm{fv}(t_i)$, as usual. A term t such that $\mathrm{fv}(t) = \emptyset$ is called *closed* or *ground*.

A *substitution* σ is any map from variables to terms of the same type that is almost everywhere the identity—i.e., $\triangleright\sigma(x_\tau) : \tau$ and for all but finitely many variables x, $\sigma(x) = x$. The *domain* $\mathrm{dom}\,\sigma$ of σ is the set of variables x such that $\sigma(x) \neq x$. The *yield* $\mathrm{yld}\,\sigma$ is $\bigcup_{x \in \mathrm{dom}\,\sigma} \mathrm{fv}(\sigma(x))$. The notation $[x_1 := t_1, \ldots, x_m := t_m]$ denotes the substitution mapping each x_i to t_i, $1 \leq i \leq m$, and mapping every other variable y to y, whenever the t_i's are terms of the same type as x_i; in particular, $[]$ is the *identity substitution.* Let Σ be the set of all substitutions.

Substitution application $t \mapsto t\sigma$ is defined by $x\sigma =_{\mathrm{df}} \sigma(x)$, $c(t_1, \ldots, t_n)\sigma =_{\mathrm{df}} c(t_1\sigma, \ldots, t_n\sigma)$. The *composition* $\sigma\sigma'$ of σ and σ' is defined as the unique substitution such that $t(\sigma\sigma') = (t\sigma)\sigma'$ for all terms t; this defines a monoid law, with the identity substitution $[]$ as unit. The relation $\succeq$ defined by $\sigma \succeq \sigma\sigma'$ is then a preorder, the *instantiation preorder*; we also say that σ is *more general than* $\sigma\sigma'$. We write $\sigma \equiv \sigma'$ if and only if $\sigma \succeq \sigma'$ and $\sigma' \succeq \sigma$.

A *system of equations* E is any finite set of *equations* $s \approx t$, where s and t are terms of the same type. The notation E, E' denotes the union of E and E'. A substitution σ *unifies* E if and only if $s\sigma = t\sigma$ for every $s \approx t$ in E—we abbreviate this situation by the notation $\sigma \Vdash E$. A substitution σ is *idempotent* if and only if $\mathrm{dom}\,\sigma \cap \mathrm{yld}\,\sigma = \emptyset$; this implies that $\sigma\sigma = \sigma$. It is well-known [Wal88] that every unifiable system of equations E has an idempotent *most general unifier* σ_0, i.e. $\sigma_0 \Vdash E$ and for every σ, $\sigma \Vdash E$ implies $\sigma_0 \succeq \sigma$. We shall also call most general unifiers *mgus* for short.

Conversely, any substitution $\sigma =_{\mathrm{df}} [x_1 := t_1, \ldots, x_m := t_m]$ with $\mathrm{dom}\,\sigma = \{x_1, \ldots, x_m\}$ defines a unique system of equations $\overline{\sigma} =_{\mathrm{df}} \{x_1 \approx t_1, \ldots, x_m \approx t_m\}$. We let $E \mapsto mgu(E)$ be an arbitrary function mapping each unifiable system E of equations to some idempotent unifier of E.

Let $\perp$ be an element outside Σ, designed to represent non-unifiable systems. Extend $\succeq$ so that $\sigma \succeq \perp$ for every $\sigma \in \Sigma$. We may then extend the mgu function so that $mgu(E) = \perp$ for every non-unifiable system E. Let also $\overline{\perp} =_{\mathrm{df}} \perp$, and extend again the mgu function so that $mgu(\perp, E) =_{\mathrm{df}} \perp$, $mgu(E, \perp) =_{\mathrm{df}} \perp$. This makes $\Sigma_\perp =_{\mathrm{df}} \Sigma \cup \{\perp\}$ equipped with the pre-order $\succeq$ a meet-semi-lattice with bottom element $\perp$, meet $\sqcap$ defined by $\sigma \sqcap \sigma' =_{\mathrm{df}} mgu(\overline{\sigma}, \overline{\sigma'})$ for every $\sigma, \sigma' \in \Sigma_\perp$. To make our notations more uniform, we shall again write $mgu(\sigma, \sigma')$ for $\sigma \sqcap \sigma'$.

We build atomic formulas using predicate symbols P, Q, ..., taken from a fixed set $\mathcal{P}$; each predicate symbol is equipped with an *arity* $\alpha(P) = \tau_1 \times \ldots \times \tau_n \rightarrow o$, where o is the type of propositions, and is assumed not to be a sort. The *atoms* are either *non-equality atoms* A, B, ..., which are expressions of the form $P(t_1, \ldots, t_n)$ where P is as above, and $\triangleright t_1 : \tau_1$, ..., $\triangleright t_n : \tau_n$; or *equalities* $s \approx t$, where s and t have the same type, whatever it is. We assume that $\approx$ is not in $\mathcal{P}$.

Formulas $F, G, \ldots$, are then built upon atoms using $\mathbf{0}$ (false), $\supset$ (implication), $\wedge$ (conjunction), $\vee$ (disjunction), $\forall x\cdot$ (universal quantification) and $\exists x\cdot$ (existential quantification) in the standard way; $\neg F$ abbreviates $F \supset \mathbf{0}$, $\supset$ associates to the right and binds looser than $\vee$, which binds looser than $\wedge$. Moreover, we equate α-equivalent formulas, i.e., formulas that differ only by a change of bound variables; and we shall assume that formulas are *rectified*, that is, that no variable appears both bound and free, or bound by two different occurrences of quantifiers.

Our algebra of sorts and the typing restrictions on terms and atoms are designed to keep type-checking and unification problems simple. Undoubtedly, these restrictions can be lifted (see [JK90]), but this comes at a price, and it is not our purpose to pay for it here.

Formally, here is what we call a language. The ϕ component will be explained later on.

Definition 1.1 *A* language *is a tuple* $(\mathcal{C}, \mathcal{P}, \alpha, \phi)$, *where:*

- $\mathcal{C} \cap \mathcal{P} = \emptyset$, $(\approx) \notin \mathcal{P}$;
- *the* arity function α *maps each element* $c \in \mathcal{C}$ *to an expression of the form* $\tau_1 \times \ldots \times \tau_n \rightarrow \tau$, *and each element* $P \in \mathcal{P}$ *to an expression of the form* $\tau_1 \times \ldots \times \tau_n \rightarrow o$;
- ϕ *maps each* $P \in \mathcal{P}$ *such that* $\alpha(P) = \tau_1 \times \ldots \times \tau_n \rightarrow o$ *to a set of subsets of* $\{1, \ldots, n\}$; *these subsets are called the* functionalities *of* P;

Moreover, we assume that for every sort τ, there is at least one ground term of type τ.

The latter is in the spirit of the Herbrand restriction that the Herbrand base contain at least one constant. This is easy to ensure: just add one constant constructor to empty sorts.

The canonical semantics of this language is as follows. An *interpretation* I is given by a family $I(\tau)$ of pairwise disjoint non-empty sets indexed by sorts τ, a function $I(c)$ from $I(\tau_1) \times \ldots \times I(\tau_n)$ to $I(\tau)$ for each constructor c, of arity $\tau_1 \times \ldots \times \tau_n \to \tau$, a subset $I(P)$ of $I(\tau_1) \times \ldots \times I(\tau_n)$ for each predicate symbol P, of arity $\tau_1 \times \ldots \times \tau_n \to o$, and a subset $I(\approx)$ of $\bigcup_\tau I(\tau) \times I(\tau)$.

Valuations ρ map variables x_τ to elements of $I(\tau)$. We define the semantics $I[\![t]\!]\rho$ of terms t by $I[\![x]\!]\rho =_{\mathrm{df}} \rho(x)$, $I[\![c(t_1, \ldots, t_n)]\!]\rho =_{\mathrm{df}} I(c)(I[\![t_1]\!]\rho, \ldots, I[\![t_n]\!]\rho)$. The semantics of formulas F is given by: $I, \rho \models P(t_1, \ldots, t_n)$ if and only if $(I[\![t_1]\!]\rho, \ldots, I[\![t_n]\!]\rho) \in I(P)$, and $I, \rho \models s \approx t$ if and only if $(I[\![s]\!]\rho, I[\![t]\!]\rho) \in I(\approx)$, and logical connectives are defined in the usual Tarskian way. We write $I \models F$ if and only if $I, \rho \models F$ for every valuation ρ.

An *equational* interpretation I is such that $I, \rho \models s \approx t$ if and only if $I[\![s]\!]\rho = I[\![t]\!]\rho$, i.e. it is an interpretation in which $I(\approx)$ is exactly the diagonal set $\{(v, v) \mid v \in \bigcup_\tau I(\tau)\}$.

Now we shall be interested in interpretations that are *free*, in that they will obey non-confusion in the following way [Com91]:

Definition 1.2 *An interpretation I is* free *if and only if:*

- *if $I[\![c(s_1, \ldots, s_m)]\!]\rho = I[\![d(t_1, \ldots, t_n)]\!]\rho$, then $c = d$, $m = n$, and $I[\![s_i]\!]\rho = I[\![t_i]\!]\rho$ for every i, $1 \leq i \leq m$;*
- *and if $I[\![x]\!]\rho = I[\![t]\!]\rho$, then either $x = t$ or x is not free in t.*

2 Design of LKc$_\approx$

Before we introduce the rules of the sequent system LKc$_\approx$, we wish to motivate it.

Because our domains are free, an equation $c(s_1, \ldots, s_m) \approx d(t_1, \ldots, t_n)$ can only hold when c and d are the same constructor (in particular, $m = n$), and the equations $s_1 \approx t_1$, $\ldots$, $s_m \approx t_m$ hold; also, $x \approx t$ can only hold when $x = t$ or when x is not free in t. The astute reader will have recognized the basic rules for Martelli-Montanari-style first-order unification [MM82]. Consequently, to prove an implication of the form $s_1 \approx t_1 \supset s_2 \approx t_2$, we may replace $s_1 \approx t_1$ by its most general unifier σ; then, if $s_2\sigma = t_2\sigma$, the implication is proved. For example, $\mathtt{S}(x) \approx \mathtt{S}(y) \supset x \approx y$ holds because $mgu(\mathtt{S}(x) \approx \mathtt{S}(y)) \equiv [x := y]$, and $x[x := y] = y[x := y]$.

This justifies that we consider sequents of the form $\sigma; \Gamma \vdash \Delta$. The substitution part σ in effect collects a preprocessed form of some equalities on the left of $\vdash$: the above sequent has the same semantics as the usual sequent $\overline{\sigma}, \Gamma \vdash \Delta$. Formally, we let:

Definition 2.1 *An* LKc$_\approx$ *sequent is a triple $\sigma; \Gamma \vdash \Delta$, where $\sigma = \bot$ or σ is an idempotent substitution in Σ, and Γ and Δ are multisets of formulas.*

If σ is the identity substitution $[]$, we also write $; \Gamma \vdash \Delta$ instead of $[]; \Gamma \vdash \Delta$. We shall write S, S' for the multiset union of S and S', and we shall silently coerce elements to one-element multisets.

On the other hand, the example above also justifies the following *reflexivity rule*:

$$\frac{\sigma \neq \bot,\ s\sigma = t\sigma}{\sigma; \Gamma \vdash s \approx t, \Delta}\ (\approx R)$$

When $\sigma = \bot$, the equalities that σ represents are non-unifiable, i.e. they are contradictory. For example, consider the equation $\mathtt{0} \approx \mathtt{S}(x)$: if it occurs on the left of $\vdash$, then the sequent is proved. Hence the following absurdity rule:

$$\frac{}{\bot; \Gamma \vdash \Delta}\ (\bot L)$$

We must not forget to actually process the equalities in the Γ part, and have them mix with the σ part. This is done by the following rule:

$$\frac{mgu(s \approx t, \overline{\sigma}); \Gamma \vdash \Delta}{\sigma; \Gamma, s \approx t \vdash \Delta} (\approx L)$$

and we must also allow the system to conclude when the same formula occurs in the Γ and in the Δ parts—up to equalities represented by σ:

$$\frac{\sigma \neq \bot, \ A\sigma = B\sigma}{\sigma; \Gamma, A \vdash B, \Delta} (\approx)$$

(Recall that, by convention, A and B are non-equality atoms.) Before we go on, please notice that our use of unification has nothing to do with proof search—a topic which we shall touch upon in Section 4—, but is just a tool for equality reasoning.

We then add the other usual sequent rules for the logical connectives, with the only change that they now all have an additional σ on the left, e.g.:

$$\frac{\sigma; \Gamma, F \vdash G, \Delta}{\sigma; \Gamma \vdash F \supset G, \Delta} (\supset R) \qquad \frac{\sigma; \Gamma, G \vdash \Delta \quad \sigma; \Gamma \vdash F, \Delta}{\sigma; \Gamma, F \supset G \vdash \Delta} (\supset L)$$

This is all we need to get a sound and complete sequent system for first-order logic with equality, as we shall see in Section 3. Notice that, assuming that our claim holds, this entails the surprising fact that the only proof of $\sigma; \Gamma \vdash s \approx t$ when $\sigma; \Gamma$ is not contradictory is—after some applications of left rules—by reflexivity, i.e. by rule $(\approx R)$.

Another fact to bear in mind is that we do not have any function symbols, apart from constructors. This is in the tradition of early logicians' work, and in particular of the Principia [WR27], so it does not entail any loss of generality. However, doing so traditionally exacts a penalty: functions have to be coded as predicates—say, the binary $+$ function as a ternary $\oplus(x, y, z)$ meaning $z \approx x + y$. This encoding is a heavy burden to an automated prover, not so much because functions have been replaced by predicates *per se*, but mostly because it will have to reason modulo two new kinds of axioms: *functionality axioms* that express the uniqueness of the result of the function—e.g., $\forall x, y, z, z' \cdot \oplus(x, y, z) \wedge \oplus(x, y, z') \supset z \approx z'$—and *totality axioms* that express the existence of the result of the function—e.g., $\forall x, y \cdot \exists z \cdot \oplus(x, y, z)$.

Of course, we might have just introduced function symbols into our calculus. Some function symbols would be constructors, and the others would be uninterpreted. But then equality reasoning would start to become complex again. Recall that the title of this report includes the word "simple": it is one of our main motivations here to strive for simplicity.

And there is a relatively simple way to build in functionality axioms when functions are coded as predicates. To this end, we equip each predicate symbol P with a set $\phi(P)$ of *functionalities*. Intuitively, a functionality f is a set of argument positions for P such that, given some values at these positions, the values at the positions outside f are uniquely determined. For instance, we shall say that the predicate $\oplus$ above has functionality $\{1, 2\}$, since whenever its first and second arguments are given, its third is determined uniquely. We shall assume that it also has the functionalities $\{1, 3\}$ and $\{2, 3\}$, because subtraction is also a (partial) function; that is, $\phi(\oplus) = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$. Formally, if $\alpha(P) = \tau_1 \times \ldots \times \tau_n \to o$, $\phi(P)$ is a set of subsets of $\{1, \ldots, n\}$, as announced in Definition 1.1.

Semantically, the interpretations that we shall consider will have to respect functionalities:

Definition 2.2 *An interpretation I* respects functionalities *if and only if, for every $P \in \mathcal{P}$, where $\alpha(P) = \tau_1 \times \ldots \times \tau_n \to \tau$, for every $f \in \phi(P)$, for every $v_1, v_1' \in I(\tau_1)$, $\ldots$, $v_n, v_n' \in I(\tau_n)$, if $(v_1, \ldots, v_n) \in I(P)$ and $(v_1', \ldots, v_n') \in I(P)$, and $v_i = v_i'$ for every $i \in f$, then $v_i = v_i'$ for every $i \in \{1, \ldots, n\}$.*

To express this intention in the proof system, we add one rule to our sequent system. Given $f \in \phi(P)$, let $\overline{f}^n$ denote $\{1, \ldots, n\} \setminus f$; given a finite set $f =_{\mathrm{df}} \{i_1, \ldots, i_k\}$ of integers, and families of terms $(s_i)_{i\geq 0}$, $(t_i)_{i\geq 0}$, write $\vec{s}_f \approx \vec{t}_f$ for $s_{i_1} \approx t_{i_1}, \ldots, s_{i_k} \approx t_{i_k}$. The desired rule should express that (apart from a few notational inaccuracies):

$$P(s_1, \ldots, s_n) \wedge P(t_1, \ldots, t_n) \supset \vec{s}_f \approx \vec{t}_f \supset \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n}$$

This is best expressed as a left rule, mimicking $(\supset L)$,where $f = \{i_1, \ldots, i_k\} \in \phi(P)$:

$$\frac{\sigma; \Gamma', \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta \quad \sigma; \Gamma' \vdash s_{i_1} \approx t_{i_1}, \Delta \quad \ldots \quad \sigma; \Gamma' \vdash s_{i_k} \approx t_{i_k}, \Delta}{\sigma; \underbrace{\Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n)}_{\Gamma'} \vdash \Delta}$$

However, following our motto that the only non-trivial proof of an equality $s_{i_j} \approx t_{i_j}$ is by reflexivity, we can simplify this to:

$$\frac{\sigma; \Gamma', \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta \quad \vec{s}_f\sigma = \vec{t}_f\sigma}{\sigma; \underbrace{\Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n)}_{\Gamma'} \vdash \Delta} \ (\approx\uparrow)$$

where $\vec{s}_f\sigma = \vec{t}_f\sigma$ abbreviates the list of identities $s_{i_1}\sigma = t_{i_1}\sigma, \ldots, s_{i_k}\sigma = t_{i_k}\sigma$.

We can simplify again by apply $(\approx L)$ eagerly, to get the following rule:

$$\frac{mgu(\vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n}, \overline{\sigma}); \Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \Delta \quad \sigma \neq \bot \quad \vec{s}_f\sigma = \vec{t}_f\sigma}{\sigma; \Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \Delta}$$

but we won't, as this would force a prover to apply $(\approx L)$ just above $(\approx\uparrow)$, restricting the generality of LKc$_\approx$; it would also force us to duplicate all the arguments that apply to $(\approx L)$ for $(\approx\uparrow)$ in subsequent proofs.

The resulting calculus LKc$_\approx$ is shown in Figure 1. Although you should be convinced that these rules are sound, completeness is by no means obvious. It will turn out that soundness is not trivial at all either. We shall investigate these properties in Section 3. (Recall that the title claims that the *deduction system* should be simple, not necessarily that its soundness and completeness proofs should be.)

We have not dealt with totality axioms. In fact, there does not seem to be a much smarter way to deal with totality than to include totality axioms, possibly in a disguised way, explicitly into the left-hand sides of sequents. In effect, this predicative encoding of functions makes us consider all functions that are not constructors as partial. This might actually be an advantage in some cases. First, this concurs with mathematical practice, where most functions are first defined as partial functions, then proved total. Second, this is actually a quite satisfactory solution to the problem of manipulating partial functions in program specifications, an outstanding problem that has spurred a lot of research (see e.g., [Jon90]). In our motivating example that dealt with cryptographic protocols, all functions that are not constructors (e.g., hash-tables) are in fact partial by nature. It would nonetheless be interesting to be able to build in more trivial totality axioms inside LKc$_\approx$, and this is left as future work.

3 Soundness, Completeness

Let's examine the meta-mathematical properties of LKc$_\approx$: soundness, completeness with respect to the Tarskian semantics. We shall also show that, in LKc$_\approx$, we never need (Cut). This is needed if we wish to be able to find proofs automatically in this system.

Before we get on to soundness, though, we have to tackle a few details related to unification and the $E \mapsto mgu(E)$ function.

$$\frac{}{\bot ; \Gamma \vdash \Delta}\,(\bot L) \qquad \frac{\sigma \neq \bot, \quad A\sigma = B\sigma}{\sigma ; \Gamma, A \vdash B, \Delta}\,(\approx)$$

$$\frac{\sigma \neq \bot, \quad mgu(s \approx t, \overline{\sigma}); \Gamma \vdash \Delta}{\sigma ; \Gamma, s \approx t \vdash \Delta}\,(\approx L) \qquad \frac{\sigma \neq \bot, \quad s\sigma = t\sigma}{\sigma ; \Gamma \vdash s \approx t, \Delta}\,(\approx R)$$

$$\frac{\begin{array}{c} f \in \phi(P) \quad \sigma \neq \bot \quad \vec{s}_f\sigma = \vec{t}_f\sigma \\ \sigma ; \Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n), \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta \end{array}}{\sigma ; \Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \Delta}\,(\approx\uparrow)$$

$$\frac{\sigma ; \Gamma, G \vdash \Delta \quad \sigma ; \Gamma \vdash F, \Delta}{\sigma ; \Gamma, F \supset G \vdash \Delta}\,(\supset L) \qquad \frac{\sigma ; \Gamma, F \vdash G, \Delta}{\sigma ; \Gamma \vdash F \supset G, \Delta}\,(\supset R)$$

$$\frac{}{\sigma ; \Gamma, \mathbf{0} \vdash \Delta}\,(\mathbf{0}L)$$

$$\frac{\sigma ; \Gamma, F, G \vdash \Delta}{\sigma ; \Gamma, F \wedge G \vdash \Delta}\,(\wedge L) \qquad \frac{\sigma ; \Gamma \vdash F, \Delta \quad \sigma ; \Gamma \vdash G, \Delta}{\sigma ; \Gamma \vdash F \wedge G, \Delta}\,(\wedge R)$$

$$\frac{\sigma ; \Gamma, F \vdash \Delta \quad \sigma ; \Gamma, G \vdash \Delta}{\sigma ; \Gamma, F \vee G \vdash \Delta}\,(\vee L) \qquad \frac{\sigma ; \Gamma \vdash F, G, \Delta}{\sigma ; \Gamma \vdash F \vee G, \Delta}\,(\vee R)$$

$$\frac{\sigma ; \Gamma, \forall x \cdot F, F[x := t] \vdash \Delta}{\sigma ; \Gamma, \forall x \cdot F \vdash \Delta}\,(\forall L) \qquad \frac{\sigma ; \Gamma \vdash F[x := y_\tau], \Delta}{\sigma ; \Gamma \vdash \forall x_\tau \cdot F, \Delta}\,(\forall R)$$

$$(y \text{ not free in } \overline{\sigma}, \Gamma, \forall x_\tau \cdot F, \Delta)$$

$$\frac{\sigma ; \Gamma, F[x := y_\tau] \vdash \Delta}{\sigma ; \Gamma, \exists x \cdot F \vdash \Delta}\,(\exists L) \qquad \frac{\sigma ; \Gamma \vdash F[x := t], \exists x \cdot F, \Delta}{\sigma ; \Gamma \vdash \exists x \cdot F, \Delta}\,(\exists R)$$

$$(y \text{ not free in } \overline{\sigma}, \Gamma, \exists x_\tau \cdot F, \Delta)$$

$$\frac{\sigma ; \Gamma \vdash F, \Delta \quad \sigma ; \Gamma, F \vdash \Delta}{\sigma ; \Gamma \vdash \Delta}\,(Cut)$$

Figure 1: System $\mathrm{LKc}_{\approx}$

3.1 Most General Unifiers

We shall use the unification algorithm of Figure 2 as a guide for proving certain properties of most general unifiers. This is a simple variant of Martelli and Montanari's algorithm (see [JK90]). We say that a variable x is *solved in* E if and only if E is of the form $E', x \approx s$, where x is not free in s and not free in E'. E is *solved* if all the equations of E are of the form $x \approx s$, where x is solved in E. Let $\#E$ be the number of free unsolved variables in E. Let also $|E|$ be the *size* of E, defined as the sum of $|s \approx t|$, $(s \approx t) \in E$, where $|s \approx t| =_{\mathrm{df}} |s| + |t|$, and the size of terms is defined by: $|x| =_{\mathrm{df}} 1$, $|c(t_1, \ldots, t_n)| =_{\mathrm{df}} 1 + |t_1| + \ldots + |t_n|$. Here, $\perp$ is a token denoting the absence of unifiers.

$$
\begin{array}{rlcll}
\textbf{(Delete)} & E, s \approx s & \to & E & \\
\textbf{(Check1)} & E, x \approx t & \to & \perp & (x \neq t, x \in \mathrm{fv}(t)) \\
\textbf{(Check2)} & E, t \approx x & \to & \perp & (x \neq t, x \in \mathrm{fv}(t)) \\
\textbf{(Bind1)} & E, x \approx t & \to & E[x := t], x \approx t & (x \notin \mathrm{fv}(t), x \text{ not solved in } E, x \approx t) \\
\textbf{(Bind2)} & E, t \approx x & \to & E[x := t], x \approx t & (x \notin \mathrm{fv}(t)) \\
\textbf{(Clash)} & E, c(s_1, \ldots, s_m) \approx d(t_1, \ldots, t_n) & \to & \perp & (c \neq d) \\
\textbf{(Decomp)} & E, c(s_1, \ldots, s_m) \approx c(t_1, \ldots, t_m) & \to & E, s_1 \approx t_1, \ldots, s_m \approx t_m &
\end{array}
$$

Figure 2: Rules for first-order unification

Any sequence of steps $E_0 \to E_1 \to \ldots \to E_n \to \ldots$ obtained with the rules of Figure 2 must terminate, since each application of a rule makes $(\#E, |E|)$ decrease in the lexicographic ordering. To show this, observe that whenever $E \to E'$, and x is solved in E, then x remains solved in E'; since no fresh variables are ever created, $\#E$ never increases; moreover, $\#E$ decreases strictly in the case of the **(Bind1)** and **(Bind2)** rules, where x becomes solved, and $|E|$ decreases in all other rules.

Each rule preserves the sets of all unifiers, in the sense that whenever $E \to E'$, the set of unifiers of E is exactly that of E'. Moreover, an irreducible system E—one to which no rule applies—must be of the form $\perp$, or $x_1 \approx t_1, \ldots, x_k \approx t_k$, and must be solved. The substitution $\sigma =_{\mathrm{df}} [x_1 := t_1, \ldots, x_k := t_k]$ is therefore idempotent, and clearly unifies E.

In short, for any maximal sequence $E_0 \to E_1 \to \ldots \to E_n$ of rule applications, E_n must be $\perp$ or a solved system $x_1 \approx t_1, \ldots, x_k \approx t_k$; in the first case, E_0 is not unifiable, in the second case $[x_1 := t_1, \ldots, x_k := t_k]$ is a most general unifier of E_0.

We can define $mgu(E_0)$ as $\perp$ in the first case, and as $[x_1 := t_1, \ldots, x_k := t_k]$ in the second case, but we wish to prove results independently of a particular algorithm. To do this, we shall explore ways of producing all idempotent unifiers of a given system E_0. We first observe:

Lemma 3.1 *Let $E =_{\mathrm{df}} x_1 \approx t_1, \ldots, x_k \approx t_k$ be solved, and σ be an idempotent unifier of E such that* $\mathrm{dom}\, \sigma \subseteq \{x_1, \ldots, x_k\}$*. Then σ is exactly $[x_1 := t_1, \ldots, x_k := t_k]$.*

Proof. For every i, $1 \leq i \leq k$, since σ unifies x_i with t_i, we must have $\sigma(x_i) = t_i\sigma$.

We first claim that x_i is in $\mathrm{dom}\, \sigma$. Indeed, assume on the contrary that $\sigma(x_i) = x_i$. Then $x_i = t_i\sigma$. In particular, t_i is a variable y, and: (a) $\sigma(y) = x_i$. Since E is solved, $t_i \neq x_i$, hence $y \neq x_i$. Moreover, (a) implies that $y \in \mathrm{dom}\, \sigma$, so that y is some x_j, $1 \leq j \leq k$, $j \neq i$. But then E contains the equations $x_i \approx t_i$ and $x_j \approx t_j$, that is, the equations $x_i \approx x_j$ and $x_j \approx t_j$. Therefore x_j is not solved in E, contradicting the fact that E is solved.

So: (b) $\mathrm{dom}\, \sigma$ is exactly equal to $\{x_1, \ldots, x_k\}$.

We now claim that $\sigma(x_i) = t_i$ for every i, $1 \leq i \leq k$. Since σ unifies $x_i \approx t_i$: (c) $\sigma(x_i) = t_i\sigma$. Since E is solved, $\mathrm{fv}(t_i) \cap \{x_1, \ldots, x_k\} = \emptyset$. By (b), it follows that $\mathrm{fv}(t_i) \cap \mathrm{dom}\, \sigma = \emptyset$. In particular, by (c), $\sigma(x_i) = t_i$, for every i, $1 \leq i \leq n$.

The Lemma then follows from (b) and (c). $\square$

Given a solved system of equations $E =_{\mathrm{df}} x_1 \approx t_1, \ldots, x_k \approx t_k$, call *domain* $\mathrm{dom}\, E$ of E the set $\{x_1, \ldots, x_k\}$.

Lemma 3.2 *Consider the following transformation rule on systems of equations:*

$$\textbf{(Swap)} \quad E, x \approx y \quad \to \quad E[y := x], y \approx x$$

The **(Swap)** *rule transforms solved systems* E_1 *into solved systems* E_2 *that have the same set of unifiers, and such that* $\operatorname{dom} E_2 = (\operatorname{dom} E_1 \setminus \{x\}) \cup \{y\}$.

Proof. Let $E_1 =_{\mathrm{df}} E, x \approx y$ and $E_2 =_{\mathrm{df}} E[y := x], y \approx x$. Since E_1 is solved, $x \neq y$.

We first show that E_2 is solved. Write E as $x_1 \approx t_1, \ldots, x_n \approx t_n$, where each x_i, $1 \leq i \leq n$, is solved in E_1. To show that E_2 is solved, notice that each equation of E_2 is of the form $x_i[y := x] \approx t_i[y := x]$ or $y \approx x$.

- Equations $x_i[y := x] \approx t_i[y := x]$: since x_i is solved in E_1, x_i in particular does not occur in $x \approx y$, hence $x_i \neq y$. So each such equation is of the form $x_i \approx t_i[y := x]$. We claim that x_i is then solved in E_2. Let E' be E minus the equation $x_i \approx t_i$. Since E_1 is solved, we have: (a) x_i is not free in E', (b) x_i is not free in t_i, and (c) x_i is not free in $x \approx y$. In particular: (d) $x_i \neq x$. Then $E_2 = E'[y := x], (x_i \approx t_i)[y := x], y \approx x = E'[y := x], x_i \approx t_i[y := x], y \approx x$ (since $x_i \neq y$), and x_i is not free in $E'[y := x]$ (by (d) and since x_i is not free in E' by (a)), x_i is not free in $t_i[y := x]$ (by (d) and since x_i is not free in t_i by (b)), x_i is not free in $y \approx x$ (by (c)). So indeed x_i is solved in E_2.
- Equation $y \approx x$: we show that y is solved in E_2. Indeed, y is not free in $E[y := x]$, since $y \neq x$; moreover, y is not free in the right-hand side of $y \approx x$ for the same reason; so y is solved in $E_2 = E[y := x], y \approx x$.

It follows that E_2 is solved.

That E_1 and E_2 have the same sets of unifiers is obvious, noticing that any unifier of either must unify x and y first. Finally, $x \in \operatorname{dom} E_1$, $y \notin \operatorname{dom} E_1$ (since E_1 is solved), $x \notin \operatorname{dom} E_2$ (since E_2 is solved), $y \in \operatorname{dom} E_2$. Moreover, for every variable z, z is in $\operatorname{dom} E_1 \setminus \{x, y\}$ if and only if z is in $\operatorname{dom} E_2 \setminus \{x, y\}$. The result follows. □

Lemma 3.3 *Let* $E =_{\mathrm{df}} x_1 \approx t_1, \ldots, x_k \approx t_k$ *be solved, and* σ *be an idempotent most general unifier of* E. *There is a finite sequence of* **(Swap)** *steps leading from* E *to* $\overline{\sigma}$.

Proof. By induction on the cardinality of $\operatorname{dom}\sigma \setminus \operatorname{dom} E$. If this cardinality is 0, then $\operatorname{dom}\sigma \subseteq \operatorname{dom} E$, so by Lemma 3.1, $\sigma = [x_1 := t_1, \ldots, x_k := t_k]$, that is, $E = \overline{\sigma}$.

Otherwise, let y be a variable in $\operatorname{dom}\sigma \setminus \operatorname{dom} E$. Let σ_0 denote the idempotent most general unifier $[x_1 := t_1, \ldots, x_k := t_k]$ of E. It is well-known that, since σ_0 and σ are both most general, there exists a renaming ϱ such that $\sigma = \sigma_0\varrho$. Recall that a *renaming* is a bijective mapping from variables to variables of the same sort. Let x be $\varrho(y)$.

Observe that: (a) $y \notin \operatorname{dom}\sigma_0$. Indeed, $y \in \operatorname{dom}\sigma \setminus \operatorname{dom} E$ by assumption, and $\operatorname{dom} E = \operatorname{dom}\sigma_0$ by construction.

Then: (b) $y\sigma = x$. Indeed, by (a) $y\sigma_0 = y$, so $y\sigma = y\sigma_0\varrho = y\varrho = x$.

We claim that: (c) $y \neq x$. Indeed, if $y = x$ then $y\sigma = y$ by (b), so y would not be in $\operatorname{dom}\sigma$, contradicting the fact that we have picked y from $\operatorname{dom}\sigma \setminus \operatorname{dom} E$.

It follows that: (d) $x \notin \operatorname{dom}\sigma$. Indeed, by definition $y \in \operatorname{dom}\sigma$, but then x is free in $y\sigma$ by (b): since σ is idempotent, (d) follows.

Also: (e) $\varrho(x) \neq x$. Indeed, if $\varrho(x) = x$, since by definition $\varrho(y) = x$, the fact that ϱ is bijective would imply $y = x$, which is impossible by (c).

We now claim that: (f) $x \in \operatorname{dom}\sigma_0$. Indeed, otherwise $x\sigma_0 = x$, so $x\sigma = x\sigma_0\varrho = x\varrho$. But $x\varrho \neq x$ by (e), so $x\sigma \neq x$. Therefore $x \in \operatorname{dom}\sigma$, contradicting (d). So (f) holds.

Then: (g) $x\sigma_0 = y$. Indeed, by (d), $x\sigma = x$. On the other hand, $x\sigma = x\sigma_0\varrho$, so $x = x\sigma_0\varrho$. In particular, $x\sigma_0\varrho$ is a variable, and therefore $x\sigma_0$ must also be a variable z, with $\varrho(z) = x$. Since $\varrho(y) = x$ and ϱ is bijective, $z = y$, so (g) follows.

By (f) and (g), together with the fact that $E = \overline{\sigma}_0$, we may write E as $E_0, x \approx y$, with $x \notin \operatorname{dom} E_0$.

Consider now the substitution $\sigma_0[y := x, x := y]$. For every $z \in \operatorname{dom} \sigma_0$, $z\sigma_0$ does not have any free variable in $\operatorname{dom} \sigma_0$, since σ_0 is idempotent, in particular by (f), x is not free in $z\sigma_0$; so for every $z \in \operatorname{dom} \sigma_0$, $z\sigma_0[y := x, x := y] = z\sigma_0[y := x]$. When $z = y$ (in particular $z \notin \operatorname{dom} \sigma_0$ by (a)), then $z\sigma_0[y := x, x := y] = x$. And when $z \notin \operatorname{dom} \sigma_0$ and $z \neq y$ (and also $z \neq x$ by (f)), then $z\sigma_0[y := x, x := y] = z$. It follows that $\overline{\sigma_0[y := x, x := y]} = E_0[y := x], y \approx x$; let E' denote this system of equations. In particular: (h) E' is obtained from E by the **(Swap)** rule.

Moreover, $\operatorname{dom} E' = (\operatorname{dom} E \setminus \{x\}) \cup \{y\}$, so $\operatorname{dom} \sigma \setminus \operatorname{dom} E' = \operatorname{dom} \sigma \setminus ((\operatorname{dom} E \setminus \{x\}) \cup \{y\}) = (\operatorname{dom} \sigma \setminus (\operatorname{dom} E \setminus \{x\})) \setminus \{y\}$ (by algebra). On the other hand, $\operatorname{dom} \sigma \setminus (\operatorname{dom} E \setminus \{x\}) = (\operatorname{dom} \sigma \setminus \operatorname{dom} E) \cup (\operatorname{dom} \sigma \cap \{x\})$ (by algebra), but $x \notin \operatorname{dom} \sigma$ by (d). So $\operatorname{dom} \sigma \setminus (\operatorname{dom} E \setminus \{x\}) = \operatorname{dom} \sigma \setminus \operatorname{dom} E$. Therefore $\operatorname{dom} \sigma \setminus \operatorname{dom} E' = (\operatorname{dom} \sigma \setminus \operatorname{dom} E) \setminus \{y\}$. Since y was chosen inside $\operatorname{dom} \sigma \setminus \operatorname{dom} E$, it follows that $\operatorname{dom} \sigma \setminus \operatorname{dom} E'$ is strictly smaller than $\operatorname{dom} \sigma \setminus \operatorname{dom} E$, hence the induction hypothesis applies to E': there is a finite sequence of **(Swap)** rules leading from E' to $\overline{\sigma}$. Together with (h), this concludes the proof. □

3.2 Soundness

We extend the $\models$ notation as follows: $I, \rho \models \sigma$, where $\sigma \in \Sigma_\perp$, if and only if $\sigma \neq \perp$ and $I[\![x]\!]\rho = I[\![\sigma(x)]\!]\rho$ for every variable x. $I, \rho \models (\sigma; \Gamma \vdash \Delta)$ is defined as: if $I, \rho \models \sigma$, and for every F in Γ, $I, \rho \models F$, then there is a formula G in Δ such that $I, \rho \models G$. Again, $I \models (\sigma; \Gamma \vdash \Delta)$ if and only if $I, \rho \models (\sigma; \Gamma \vdash \Delta)$ for every valuation ρ.

The goal of this section is to show the following:

Theorem 3.4 (Soundness) *If $\sigma; \Gamma \vdash \Delta$ is provable in* LKc$_\approx$*, then for every free equational interpretation I that respects functionalities, for every valuation ρ: $I, \rho \models (\sigma; \Gamma \vdash \Delta)$.*

Proof. We prove the Theorem by structural induction on the proof, looking at the last rule used:

- $(\perp L)$: by definition $I, \rho \not\models \perp$, so $I, \rho \models (\perp; \Gamma \vdash \Delta)$ for every ρ, vacuously.

- $(\approx)$: assume $\sigma \neq \perp$, $A\sigma = B\sigma$. To show that $I, \rho \models (\sigma; \Gamma, A \vdash B, \Delta)$ for every ρ, we need to prove a few auxiliary lemmas.

  Lemma 3.5 *Whenever $I, \rho \models \sigma$, then $I[\![s]\!]\rho = I[\![s\sigma]\!]\rho$ for every term s.*

  Proof. By structural induction on s. If s is a variable x, then $I[\![x]\!]\rho = \rho(x)$ (by definition) $= I[\![\sigma(x)]\!]\rho$ (since $I, \rho \models \sigma$) $= I[\![s\sigma]\!]\rho$. If $s = c(s_1, \ldots, s_n)$, then $I[\![s]\!]\rho = I(c)(I[\![s_1]\!]\rho, \ldots, I[\![s_n]\!]\rho) = I(c)(I[\![s_1\sigma]\!]\rho, \ldots, I[\![s_n\sigma]\!]\rho)$ (by induction) $= I[\![s\sigma]\!]\rho$. □

  Lemma 3.6 *For every s, t, if $s\sigma = t\sigma$ and $I, \rho \models \sigma$, then $I[\![s]\!]\rho = I[\![t]\!]\rho$.*

  Proof. $I[\![s]\!]\rho = I[\![s\sigma]\!]\rho$ (by Lemma 3.5) $= I[\![t\sigma]\!]\rho$ (since $s\sigma = t\sigma$ by assumption) $= I[\![t]\!]\rho$ (by Lemma 3.5). □

  Lemma 3.7 *For every non-equality atoms A, B, whenever $A\sigma = B\sigma$ and $I, \rho \models \sigma$, then $I, \rho \models A$ if and only if $I, \rho \models B$.*

  Proof. Let A be $P(s_1, \ldots, s_n)$, and B be $P(t_1, \ldots, t_n)$ (observe that A and B must have the same predicate symbol, since they are unifiable); $A\sigma = B\sigma$ implies $s_i\sigma = t_i\sigma$ for every i, $1 \leq i \leq n$. Then $I, \rho \models A$ if and only if $(I[\![s_1]\!]\rho, \ldots, I[\![s_n]\!]) \in I(P)$, if and only if $(I[\![s_1\sigma]\!]\rho, \ldots, I[\![s_n\sigma]\!]) \in I(P)$ (by Lemma 3.5), if and only if $(I[\![t_1\sigma]\!]\rho, \ldots, I[\![t_n\sigma]\!]) \in I(P)$ (since $s_i\sigma = t_i\sigma$ for every i), if and only if $(I[\![t_1]\!]\rho, \ldots, I[\![t_n]\!]) \in I(P)$ (by Lemma 3.5), if and only if $I, \rho \models B$. □

  The soundness of $(\approx)$ is then proved as follows. If $I, \rho \models \sigma$ and $I, \rho \models F$ for every F in Γ, A, then in particular $I, \rho \models A$, and by Lemma 3.7 $I, \rho \models B$; therefore $I, \rho \models (\sigma; \Gamma, A \vdash B, \Delta)$.

- ($\approx R$): If $I, \rho \models \sigma$ and $I, \rho \models F$ for every $F \in \Gamma$, then since $s\sigma = t\sigma$, by Lemma 3.6 $I, \rho \models s \approx t$, hence $I, \rho \models (\sigma; \Gamma \vdash s \approx t, \Delta)$.

- ($\approx L$): again, we first make a few auxiliary claims. Write $I, \rho \models E$ to say that $I, \rho \models s \approx t$ for every equation $s \approx t$ in E.

  Proposition 3.8 *For every substitution σ, $I, \rho \models \sigma$ if and only if $I, \rho \models \overline{\sigma}$.*

  Lemma 3.9 *For every system of equations E, for every ρ, if $I, \rho \models E$, then E is unifiable and $I, \rho \models \sigma$, for all idempotent mgus σ of E.*

  Proof. We show the claim by induction on $(\#E, |E|)$ ordered lexicographically. When E is not solved, consider each of the rules of Figure 2:

  - **(Delete)**: $E = E', s \approx s$ for some term s and some system E'. If $I, \rho \models E$, then $I, \rho \models E'$. By induction hypothesis, there is an mgu σ of E', which must also be one of E since the rules preserve mgus; besides, for every idempotent mgu σ of E, σ is an idempotent mgu of E', and by induction hypothesis again $I, \rho \models \sigma$.
  - **(Check1)**, **(Check2)**: E contains an equation of the form $x \approx t$ or $t \approx x$ with $x \neq t$ and x free in t. Then $I, \rho \models E$ implies $I, \rho \models x \approx t$, and therefore $I[\![x]\!]\rho = I[\![t]\!]\rho$. By Definition 1.2, the latter implies that $x = t$ or that x is not free in t, both of which are impossible. So it cannot be the case that $I, \rho \models E$, and this case is vacuously true.
  - **(Bind1)**, **(Bind2)**: $E = E', x \approx t$ or $E = E', t \approx x$ where x is not free in t, and x is not solved in E. Let E_1 be $E'[x := t], x \approx t$, and assume $I, \rho \models E$. In particular, $I, \rho \models x \approx t$, therefore $I, \rho \models [x := t]$. Then, for every equation $s_1 \approx s_2$ in E', $I, \rho \models s_1 \approx s_2$, and therefore $I, \rho \models s_1[x := t] \approx s_2[x := t]$, using Lemma 3.5 once on s_1 and once on s_2. It follows that $I, \rho \models E'[x := t]$, and therefore that $I, \rho \models E_1$. By the induction hypothesis, E' has an mgu σ, which is also one of E; moreover, every idempotent mgu σ of E is an idempotent mgu of E_1, and by induction hypothesis again, $I, \rho \models \sigma$.
  - **(Clash)**, **(Decomp)**: $E = E', c(s_1, \ldots, s_m) \approx d(t_1, \ldots, t_n)$. Assume $I, \rho \models E$. Then $I[\![c(s_1, \ldots, s_m)]\!]\rho = I[\![d(t_1, \ldots, t_n)]\!]\rho$, and therefore, by Definition 1.2, $c = d$, $m = n$ and $I[\![s_i]\!]\rho = I[\![t_i]\!]\rho$, $1 \leq i \leq n$. Let then E_1 be $E', s_1 \approx t_1, \ldots, s_n \approx t_n$, which is obtained from E by **(Decomp)**. In particular, $I, \rho \models E_1$. By induction hypothesis, there is an mgu σ of E_1, hence of E. Furthermore, every idempotent mgu σ of E must also be one of E_1, hence $I, \rho \models \sigma$ by induction hypothesis.

  If none of these rules apply, E is solved. So assume $E =_{\text{df}} x_1 \approx t_1, \ldots, x_k \approx t_k$ solved. Then $\sigma =_{\text{df}} [x_1 := t_1, \ldots, x_k := t_k]$ is an mgu of E, and σ is clearly idempotent.

  It remains to show that, for every idempotent mgu σ' of E, $I, \rho \models \sigma'$. By Lemma 3.3, there is a finite sequence of **(Swap)** steps, say n steps, leading from E to $\overline{\sigma'}$. We prove the claim by induction on n. If $n = 0$, we have already proved this. Otherwise, $E = E', x \approx y$, and we can go from $E'[y := x], y \approx x$ to $\overline{\sigma'}$ in $n - 1$ **(Swap)** steps. By assumption σ' is an mgu of E, so by Lemma 3.2, σ' is an mgu of E'; by induction hypothesis, $I, \rho \models \sigma'$. □

  We now show that ($\approx L$) is sound. Assume that we have derived $mgu(s \approx t, \overline{\sigma}); \Gamma \vdash \Delta$. By induction hypothesis: (a) $I, \rho \models (mgu(s \approx t, \overline{\sigma}); \Gamma \vdash \Delta)$. Now assume that: (b) $I, \rho \models \sigma$ and $I, \rho \models F$ for every formula F in $\Gamma, s \approx t$. In particular, $I, \rho \models s \approx t$. Moreover, by Proposition 3.8, $I, \rho \models \overline{\sigma}$, so $I, \rho \models s \approx t, \overline{\sigma}$. By Lemma 3.9, $I, \rho \models mgu(s \approx t, \overline{\sigma})$, whatever the actual definition of our function mgu, provided it returns idempotent most general unifiers. Since by (b) $I, \rho \models F$ for every formula F in Γ, by (a) $I, \rho \models G$ for some formula G in Δ. Discharging assumption (b), we infer that $I, \rho \models (\sigma; \Gamma, s \approx t \vdash \Delta)$.

- ($\approx\uparrow$): let $f \in \phi(P)$, $\sigma \neq \bot$, and assume that $\vec{s}_f\sigma = \vec{t}_f\sigma$, and that $\sigma; \Gamma, P(s_1, \ldots, s_n)$, $P(t_1, \ldots, t_n), \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta$ is derived. By induction hypothesis: (c) if $I, \rho \models \sigma$ and for every F in $\Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n), \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n}$, we have $I, \rho \models F$, then $I, \rho \models G$ for some G in Δ. Now assume: (d) $I, \rho \models \sigma$ and for every F in $\Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n)$, we have $I, \rho \models F$. In particular, $I, \rho \models P(s_1, \ldots, s_n)$ and $I, \rho \models P(t_1, \ldots, t_n)$. Since $I, \rho \models \sigma$ by (d) and $s_i\sigma = t_i\sigma$ for every $i \in f$, by Lemma 3.6, $I[\![s_i]\!]\rho = I[\![t_i]\!]\rho$ for every $i \in f$. Since I respects functionalities (Definition 2.2), we also have $I[\![s_i]\!]\rho = I[\![t_i]\!]\rho$ for every i outside f, i.e., $i \in \overline{f}^n$. That is, $I, \rho \models \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n}$. Together with (d), this implies that the premises of (c) are satisfied, so $I, \rho \models G$ for some formula G in Δ. Discharging (d), $I, \rho \models (\sigma; \Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \Delta)$.

- The cases of the other rules of $\mathrm{LKc}_{\approx}$ are standard. The only difficulties are with the quantifier rules. For reasons of symmetry, we shall only deal with $\forall$. The case of ($\forall L$) is trivial (but depends on the fact that ρ maps every variable x_τ to some element of $I(\tau)$, which uses the implicit fact that $I(\tau)$ is not empty).

  ($\forall R$): By induction hypothesis, for every ρ': (g) if $I, \rho' \models \sigma$ and for every F' in Γ, $I, \rho' \models F'$, then for some G' in $F[x := y], \Delta$, we have $I, \rho' \models G'$. Assume that: (h) $I, \rho \models \sigma$ and for every F' in Γ, $I, \rho \models F'$. Let v be an arbitrary element of $I(\tau)$, where τ is the type of x. Let $\rho[y := v]$ denote the valuation mapping y to v and every other variable z to $\rho(z)$. Since y is not free in $\overline{\sigma}$ and since $I, \rho \models \overline{\sigma}$ by (h) and Proposition 3.8, $I, \rho[y := v] \models \overline{\sigma}$, hence again $I, \rho[y := v] \models \sigma$. Since y is not free in Γ, by (h) again $I, \rho[y := v] \models F'$ for every F' in Γ. By (g) with $\rho' = \rho[x := v]$, for some G' in $F[x := y], \Delta$, we must have $I, \rho[y := v] \models G'$. If $I, \rho[y := v] \models F[x := y]$ for every $v \in I(\tau)$, then by definition $I, \rho \models \forall x_\tau \cdot F$, so $I, \rho \models (\sigma; \Gamma \vdash \forall x_\tau \cdot F, \Delta)$. Otherwise, for some $v \in I(\tau)$, for some G' in Δ, $I, \rho[y := v] \models G'$. Since y is not free in Δ, $I, \rho \models G'$, hence again $I, \rho \models (\sigma; \Gamma \vdash \forall x_\tau \cdot F, \Delta)$. $\square$

3.3 Completeness

We first derive a few easy properties of $\mathrm{LKc}_{\approx}$.

Lemma 3.10 *For every formulas F, G: $\sigma; \Gamma, F \vdash G, \Delta$ is provable in $\mathrm{LKc}_{\approx}$ as soon as $\sigma = \bot$, or $\sigma \neq \bot$ and $F\sigma = G\sigma$.*

Proof. Observe that ($\approx$) cannot be used directly, since the closing formulas A and B are non-equality atoms. We prove the claim by induction on $\partial(F)$, where $\partial(F)$ denotes the size of F when we do not count terms: $\partial(A) =_{\mathrm{df}} \partial(s \approx t) =_{\mathrm{df}} \partial(\mathbf{0}) =_{\mathrm{df}} 1$, $\partial(F \supset G) =_{\mathrm{df}} \partial(F \wedge G) =_{\mathrm{df}} \partial(F \vee G) =_{\mathrm{df}} \partial(F) + \partial(G) + 1$, $\partial(\forall x_\tau \cdot F) =_{\mathrm{df}} \partial(\exists x_\tau \cdot F) =_{\mathrm{df}} \partial(F) + 1$. Note that if $F\sigma = G\sigma$, then $\partial(F) = \partial(G)$.

If $\sigma = \bot$, then this is by rule ($\bot L$). Otherwise, if F is a non-equality atom A, then G is also a non-equality atom since $F\sigma = G\sigma$, then this is by rule ($\approx$). If F is an equality $s \approx t$, then G is also an equality $s' \approx t'$, with $s\sigma = s'\sigma$ and $t\sigma = t'\sigma$; then we produce the following proof:

$$\dfrac{\dfrac{}{mgu(s \approx t, \sigma); \Gamma \vdash s' \approx t'}\;(\approx R) \text{ or } (\bot L)}{\sigma; \Gamma, s \approx t \vdash s' \approx t'}\;(\approx L)$$

Indeed, letting $\sigma' =_{\mathrm{df}} mgu(s \approx t, \sigma)$, either $\sigma' = \bot$ and we use ($\bot L$); or $\sigma' \neq \bot$ and ($\approx R$) applies, because $s'\sigma' = s\sigma'$ (since σ' is an instance of σ, which unifies s and s') $= t\sigma'$ (since σ' unifies s and t by definition) $= t'\sigma'$ (since σ' is an instance of σ, which unifies t and t').

If F is not atomic, then G is not atomic either and has the same topmost logical connective. We deal with the cases of $\supset$ and $\exists$ as illustrations of what happens, and we let the reader deal with the remaining cases, since this is a classical argument. If $F = F_1 \supset F_2$, since $F\sigma = G\sigma$, in particular $G = G_1 \supset G_2$ with $F_1\sigma = G_1\sigma$ and $F_2\sigma = G_2\sigma$; then we produce the proof:

$$\dfrac{\dfrac{\begin{array}{c}(\text{induction on } F_2,\, G_2)\\ \vdots \\ \sigma;\Gamma, F_2, G_1 \vdash G_2, \Delta\end{array} \quad \begin{array}{c}(\text{induction on } G_1,\, F_1)\\ \vdots \\ \sigma;\Gamma, G_1 \vdash F_1, G_2, \Delta\end{array}}{\sigma;\Gamma, F, G_1 \vdash G_2, \Delta}\,(\supset L)}{\sigma;\Gamma, F \vdash G, \Delta}\,(\supset R)$$

If $F = \exists x_\tau \cdot F_1$, then $G = \exists x_\tau \cdot G_1$, since by α-renaming we may assume that the quantified variable is the same in both formulae; moreover, $F_1\sigma = G_1\sigma$, with x_τ not free in $\overline{\sigma}$. Let y be a variable of sort τ that is not free in $\overline{\sigma}, \Gamma, G, \Delta$, then by α-renaming $F = \exists y \cdot F_1[x := y]$ and $G = \exists y \cdot G_1[x := y]$; and we may therefore produce the proof:

$$\dfrac{\dfrac{\begin{array}{c}(\text{induction on } F_1[x := y])\\ \vdots \\ \sigma;\Gamma, F_1[x := y] \vdash G_1[x := y], \Delta\end{array}}{\sigma;\Gamma, F_1[x := y] \vdash G, \Delta}\,(\exists R)}{\sigma;\Gamma, F \vdash G, \Delta}\,(\exists L)$$

The induction indeed applies, since $F_1[x := y]\sigma = F_1\sigma[x := y]$ (since y and x are not free in $\overline{\sigma}$, in particular neither in $\mathrm{dom}\,\sigma$ nor in any $\mathrm{fv}(z\sigma)$, $z \in \mathrm{dom}\,\sigma$) $= G_1\sigma[x := y]$ (since $F_1\sigma = G_1\sigma$) $= G_1[x := y]\sigma$ (by the same argument as for F_1). $\square$

Lemma 3.11 (Weakening) *If $\sigma;\Gamma \vdash \Delta$ is provable in* LKc$_\approx$, *resp.* LKc$_\approx$ *without (Cut), then $\sigma';\Gamma',\Gamma \vdash \Delta,\Delta'$ is provable in the same system, for every $\sigma' \equiv mgu(\overline{\sigma}, E)$, for every system of equations E, for every multisets of formulas Γ' and Δ'.*

Proof. By structural induction on the given proof π of $\sigma;\Gamma \vdash \Delta$. First, if $\sigma' = \bot$, this is trivial: just use $(\bot L)$. So assume that $\sigma' \neq \bot$, in particular that $\sigma \neq \bot$.

We examine the last rule of π:

- $(\bot L)$: impossible since $\sigma \neq \bot$;
- $(\approx)$: since $A\sigma = B\sigma$, $A\sigma' = B\sigma'$ as well, so the result is by $(\approx)$ again;
- $(\approx R)$: since $s\sigma = t\sigma$, $s\sigma' = t\sigma'$ as well, so we use $(\approx R)$ again;
- $(\approx L)$: π ends in:

$$\dfrac{\begin{array}{c}\vdots \\ mgu(s \approx t, \overline{\sigma});\Gamma \vdash \Delta\end{array}}{\sigma;\Gamma, s \approx t \vdash \Delta}\,(\approx L)$$

  so we produce:

$$\dfrac{\begin{array}{c}(\text{induction or } (\bot L))\\ \vdots \\ mgu(s \approx t, \overline{\sigma}');\Gamma',\Gamma \vdash \Delta,\Delta'\end{array}}{\sigma';\Gamma',\Gamma, s \approx t \vdash \Delta,\Delta'}\,(\approx L)$$

  Indeed, $mgu(s \approx t, \overline{\sigma}') \equiv mgu(s \approx t, \overline{\sigma}, E) \equiv mgu(\overline{mgu(s \approx t, \overline{\sigma})}, E)$ (because mgu is the meet of the semi-lattice $\Sigma_\bot$), so the induction hypothesis is applicable.

- $(\approx\uparrow)$: π ends in the following, where $f \in \phi(P)$, and $\vec{s}_f\sigma = \vec{t}_f\sigma$:

$$\dfrac{\begin{array}{c}\vdots \\ \sigma;\Gamma, P(s_1,\ldots,s_n), P(t_1,\ldots,t_n), \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta\end{array}}{\sigma;\Gamma, P(s_1,\ldots,s_n), P(t_1,\ldots,t_n) \vdash \Delta}\,(\approx\uparrow)$$

so we produce:

$$\dfrac{\begin{array}{c}\text{(by induction)}\\ \vdots \\ \sigma';\Gamma',\Gamma,P(s_1,\ldots,s_n),P(t_1,\ldots,t_n),\vec{s}_{\overline{f}^n}\approx\vec{t}_{\overline{f}^n}\vdash\Delta,\Delta'\end{array}}{\sigma';\Gamma',\Gamma,P(s_1,\ldots,s_n),P(t_1,\ldots,t_n)\vdash\Delta,\Delta'}\ (\approx\uparrow)$$

- For all other rules, the argument is a straightforward appeal to the induction hypothesis.

□

Our goal now is to show:

Theorem 3.12 (Completeness) *System* LKc$_\approx$ *is complete: if* $I \models (\sigma;\Gamma\vdash\Delta)$ *for every free equational interpretation* I *that respects functionalities, then* $\sigma;\Gamma\vdash\Delta$ *is provable in* LKc$_\approx$*.*

Proof. Let T be the (infinite) set of formulas of the form:

1. (Reflexivity) $\forall x_\tau \cdot x\approx x$;
2. (Symmetry) $\forall x_\tau\cdot\forall y_\tau\cdot x\approx y\supset y\approx x$;
3. (Transitivity) $\forall x_\tau\cdot\forall y_\tau\cdot\forall z_\tau\cdot x\approx y\wedge y\approx z\supset x\approx z$;
4. (FCongruence) $\forall x_{1\,\tau_1}\cdot\forall y_{1\,\tau_1}\cdot\ldots\cdot\forall x_{n\,\tau_n}\cdot\forall y_{n\,\tau_n}\cdot x_1\approx y_1\supset\ldots\supset x_n\approx y_n\supset c(x_1,\ldots,x_n)\approx c(y_1,\ldots,y_n)$ for every constructor c of arity $\tau_1\times\ldots\times\tau_n\to\tau$; (if $n=0$, we assume that this formula denotes $c\approx c$);
5. (PCongruence) $\forall x_{1\,\tau_1}\cdot\forall y_{1\,\tau_1}\cdot\ldots\cdot\forall x_{n\,\tau_n}\cdot\forall y_{n\,\tau_n}\cdot x_1\approx y_1\supset\ldots\supset x_n\approx y_n\supset P(x_1,\ldots,x_n)\supset P(y_1,\ldots,y_n)$ for every predicate P of arity $\tau_1\times\ldots\times\tau_n\to o$; (if $n=0$, we assume that this formula denotes $P()\supset P()$);
6. (Clash) $\forall x_{1\,\tau_1}\cdot\ldots\cdot\forall x_{n\,\tau_n}\cdot\forall y_{1\,\tau'_1}\cdot\ldots\cdot\forall y_{m\,\tau'_m}\cdot\neg c(x_1,\ldots,x_n)\approx d(y_1,\ldots,y_m)$ for every distinct constructors of c and d, of respective arities $\tau_1\times\ldots\times\tau_n\to\tau$ and $\tau'_1\times\ldots\times\tau'_m\to\tau'$;
7. (Decomp) $\forall x_{1\,\tau_1}\cdot\forall y_{1\,\tau_1}\cdot\ldots\cdot\forall x_{n\,\tau_n}\cdot\forall y_{n\,\tau_n}\cdot c(x_1,\ldots,x_n)\approx c(y_1,\ldots,y_n)\supset x_i\approx y_i$ for every constructor c of arity $\tau_1\times\ldots\times\tau_n\to\tau$, $n\geq 1$, and every i, $1\leq i\leq n$;
8. (Check) $\forall y_1\cdot\ldots\cdot\forall y_n\cdot\neg x\approx t$ for every term t such that $x\neq t$ and x is free in t, where $\{y_1,\ldots,y_n\}=\mathrm{fv}(t)$;
9. (Function) $\forall x_1\cdot\forall y_1\cdot\ldots\cdot\forall x_n\cdot\forall y_n\cdot\bigwedge\vec{x}_f\approx\vec{y}_f\supset P(x_1,\ldots,x_n)\supset P(y_1,\ldots,y_n)\supset x_i\approx y_i$ for every predicate P of arity $\tau_1\times\ldots\times\tau_n\to o$, every $f\in\phi(P)$ and every $i\in\{1,\ldots,n\}$; by $\bigwedge\vec{x}_f\approx\vec{y}_f\supset F$, we denote the formula $x_{i_1}\approx y_{i_1}\supset\ldots\supset x_{i_k}\approx y_{i_k}\supset F$, where $f=\{i_1,\ldots,i_k\}$, $1\leq i_1<\ldots<i_k\leq n$; if $f=\emptyset$, this simply denotes F.

To prove completeness, the idea is to show that LKc$_\approx$ is able to prove all the formulas of T. Since LKc$_\approx$ is so close to usual sequent calculi, this is intuitively enough to conclude that LKc$_\approx$ is complete. The details are a bit more complicated.

Say that $I\models T$ if and only if $I\models F$ for every F in T. On the one hand, we have the following easy fact:

Proposition 3.13 *Every free equational interpretation* I *that respects functionalities is such that* $I\models T$*.*

The following is a kind of converse:

Lemma 3.14 *For every interpretation* I *such that* $I\models T$*, there is an interpretation, called the* quotient interpretation $I/\!\approx$*, which is free, equational and respects functionalities, and such that* $I\models F$ *if and only if* $I/\!\approx\,\models F$*, for every formula* F*.*

Proof. Assume $I \models T$. Then we build the quotient interpretation $I' =_{\mathrm{df}} I/\approx$ as follows. First, by 1.-3., $I(\approx)$ is an equivalence relation on each set $I(\tau)$. Let $I'(\tau)$ be the quotient set $I(\tau)/I(\approx)$, and let $\overline{v}$ be the equivalence class of $v \in I(\tau)$. For each constructor c of arity $\tau_1 \times \ldots \times \tau_n \to \tau$, define $I'(c) : I'(\tau_1) \times \ldots \times I'(\tau_n) \to I'(\tau)$ as the function mapping $\overline{v_1}, \ldots, \overline{v_n}$ to $\overline{I(c)(v_1, \ldots, v_n)}$; this is well-defined, that is, independent of the choices of $v_1 \in \overline{v_1}$, ..., $v_n \in \overline{v_n}$, because I obeys 4. For each predicate symbol P of arity $\tau_1 \times \ldots \times \tau_n \to o$, let $I'(P)$ be the subset of $I'(\tau_1) \times \ldots \times I'(\tau_n)$ consisting of all tuples $(\overline{v_1}, \ldots, \overline{v_n})$ such that $(v_1, \ldots, v_n) \in I(P)$; this is well-defined because I obeys 5. Define also $I'(\approx)$ as the set of all $(\overline{v_1}, \overline{v_2})$ such that $(v_1, v_2) \in I(\approx)$, that is, as the expected diagonal set of all $(\overline{v}, \overline{v})$, $\overline{v} \in \bigcup_\tau I'(\tau)$. Therefore I' is an equational interpretation.

Moreover, it is easy to see that, for every formula F, $I, \rho \models F$ if and only if $I', \overline{\rho} \models F$, where $\overline{\rho}$ maps every variable x to $\overline{\rho(x)}$: indeed, we first show by induction on the term t that $I'[\![t]\!]\overline{\rho} = \overline{I[\![t]\!]\rho}$ for every t; then, we prove the claim by induction on F: the case of atoms is by definition, and the induction case is trivial.

It follows that $I \models F$ if and only if $I' \models F$ for every formula F.

Also, in particular $I' \models T$. Then, by 6.-8., I' is a free interpretation, and by 9. it respects functionalities. $\square$

Lemma 3.15 *For every interpretation I such that $I \models T$, $I \models (\sigma; \Gamma \vdash \Delta)$ if and only if $I/\approx \models (\sigma; \Gamma \vdash \Delta)$.*

Proof. Trivial consequence of Lemma 3.14. $\square$

Now assume that: (a) $I \models (\sigma; \Gamma \vdash \Delta)$ for every free equational interpretation I that respects functionalities. If $\sigma = \bot$, then $\sigma; \Gamma \vdash \Delta$ is proved by $(\bot L)$.

So assume that $\sigma \neq \bot$. We claim that: (b) for every interpretation I, $I \models (T, \overline{\sigma}, \Gamma \vdash \Delta)$, that is, by definition, for every valuation ρ, if $I, \rho \models F$ for every $F \in T, \overline{\sigma}, \Gamma$, then $I, \rho \models G$ for some $G \in \Delta$. Since T is a collection of closed formulas, this is again equivalent to: (c) for every interpretation I, if $I \models T$ then $I \models (\sigma; \Gamma \vdash \Delta)$. Let's therefore show (c). To this end, assume that $I \models T$. $I/\approx \models (\sigma; \Gamma \vdash \Delta)$ since $I/\approx$ is a free equational interpretation I that respects functionalities by Lemma 3.14, and by assumption (a). So by Lemma 3.15 $I \models (\sigma; \Gamma \vdash \Delta)$: therefore, (c), hence (b) follows.

So $T, \overline{\sigma}, \Gamma, \neg\Delta$, where $\neg\Delta$ denotes the set of all formulas $\neg G$, $G \in \Delta$, is first-order unsatisfiable. Since first-order logic is compact [GLM97], there must be a finite subset T_{fin} of T such that $T_{fin}, \overline{\sigma}, \Gamma, \neg\Delta$ is unsatisfiable. Consider the natural deduction system ND of Figure 3, which is complete for first-order logic, in the following sense: for every multiset of formulas Λ, for every formula F, if $I \models (\Lambda \vdash F)$ for every I, then $\Lambda \vdash F$ is provable. It follows that: (d) $T_{fin}, \overline{\sigma}, \Gamma, \neg\Delta \vdash \mathbf{0}$ is provable in ND. We shall translate this ND proof to an $\mathrm{LKc}_\approx$ proof of $\sigma; \Gamma \vdash \Delta$.

Since $\mathrm{LKc}_\approx$ is so close to a traditional sequent system, why don't we use a sequent system instead of ND, then? Indeed, it might seem better to take a proof of $T_{fin}, \overline{\sigma}, \Gamma \vdash \Delta$ in some sequent system close to $\mathrm{LKc}_\approx$, and to translate the latter, recursively, to a proof in $\mathrm{LKc}_\approx$ of $\sigma; \Gamma \vdash \Delta$. However, the given sequent proof of $T_{fin}, \overline{\sigma}, \Gamma \vdash \Delta$ will use left rules which will destroy the structure of the formulas in T_{fin}. So, the left-hand sides of sequents in the given sequent proof will have an almost arbitrary shape. This means that we cannot define such a translation on sequents of the exact form $T_{fin}, \overline{\sigma}, \Gamma \vdash \Delta$ only, and we have to show that we can translate every sequent proof of sequents of the form $\Gamma_1 \vdash \Delta$ satisfying some invariant to be found. This invariant should hold of every sequent of the form $T_{fin}, \overline{\sigma}, \Gamma \vdash \Delta$, and should be preserved by the left rules. Despite our best efforts, we did not find such an invariant.

This is why we choose instead to translate a natural deduction proof of $T_{fin}, \overline{\sigma}, \Gamma, \neg\Delta \vdash \mathbf{0}$ to an $\mathrm{LKc}_\approx$ proof of $\sigma; \Gamma \vdash \Delta$: in natural deduction, there is no left rule, and our problem vanishes.

Lemma 3.16 *Whenever σ is an idempotent substitution, other than $\bot$, and $T_{fin}, \overline{\sigma}, \Lambda \vdash F$ is provable in* ND*, then $\sigma; \Lambda \vdash F$ is provable in* $\mathrm{LKc}_\approx$.

Proof. The claim is proved by structural induction on the given ND proof of $T_{fin}, \overline{\sigma}, \Lambda \vdash F$. The most important case is when the last rule is (Ax), then F is in T_{fin}, $\overline{\sigma}$ or Λ. If F is in Λ,

$$\frac{}{\Lambda, F \vdash F}\,(Ax)$$

$$\frac{\Lambda, F \vdash G}{\Lambda \vdash F \supset G}\,(\supset I) \qquad \frac{\Lambda \vdash F \supset G \quad \Lambda \vdash F}{\Lambda \vdash G}\,(\supset E)$$

$$\frac{\Lambda \vdash \mathbf{0}}{\Lambda \vdash F}\,(\mathbf{0}E) \qquad \frac{\Lambda, \neg F \vdash \mathbf{0}}{\Lambda \vdash F}\,(\neg\neg E)$$

$$\frac{\Lambda \vdash F \quad \Lambda \vdash G}{\Lambda \vdash F \wedge G}\,(\wedge I) \qquad \frac{\Lambda \vdash F_1 \wedge F_2}{\Lambda \vdash F_i}\,(\wedge E_i), i = 1, 2$$

$$\frac{\Lambda \vdash F_i}{\Lambda \vdash F_1 \vee F_2}\,(\vee I_i), i = 1, 2 \qquad \frac{\Lambda \vdash F \vee G \quad \Lambda, F \vdash H \quad \Lambda, G \vdash H}{\Lambda \vdash H}\,(\vee E)$$

$$\frac{\Lambda \vdash F[x := y_\tau]}{\Lambda \vdash \forall x_\tau \cdot F}\,(\forall I) \qquad \frac{\Lambda \vdash \forall x_\tau \cdot F \quad \rhd t : \tau}{\Lambda \vdash F[x := t]}\,(\forall E)$$

$$(y \text{ not free in } \Lambda)$$

$$\frac{\Lambda \vdash F[x_\tau := t] \quad \rhd t : \tau}{\Lambda \vdash \exists x_\tau \cdot F}\,(\exists I) \qquad \frac{\Lambda \vdash \exists x_\tau \cdot F \quad \Lambda \vdash \forall x_\tau \cdot (F \supset G)}{\Lambda \vdash G}$$

$$(x \text{ not free in } \Lambda, G)$$

Figure 3: The natural deduction system ND

then the result follows from Lemma 3.10, since $\sigma \neq \perp$. If F is in $\overline{\sigma}$, then F is of the form $x \approx t$, and the Lemma follows from the following proof:

$$\frac{}{\sigma; \Lambda \vdash x \approx t}\,(\approx R)$$

Indeed, $x\sigma = t$ (since $x \approx t$ is in $\overline{\sigma}$) $= t\sigma$ (since σ is idempotent). If F is in T_{fin}, then F is in T, and we have to consider the following 9 cases:

1. (Reflexivity)

$$\frac{\frac{}{; \vdash x \approx x}\,(\approx R)}{; \vdash \forall x_\tau \cdot x \approx x}\,(\forall R)$$

2. (Symmetry)

$$\frac{\frac{\frac{\frac{}{mgu(x \approx y); \vdash y \approx x}\,(\approx R)}{; x \approx y \vdash y \approx x}\,(\approx L)}{; \vdash x \approx y \supset y \approx x}\,(\supset R)}{; \vdash \forall x_\tau \cdot \forall y_\tau \cdot x \approx y \supset y \approx x}\,(\forall R) \text{ twice}$$

3. (Transitivity)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{mgu(x \approx z, y \approx z); \vdash x \approx z}\,(\approx R)}{mgu(x \approx y); y \approx z \vdash x \approx z}\,(\approx L)}{; x \approx y, y \approx z \vdash x \approx z}\,(\approx L)}{; \vdash x \approx y \wedge y \approx z \supset x \approx z}\,(\wedge L), (\supset R)}{; \vdash \forall x_\tau \cdot \forall y_\tau \cdot \forall z_\tau \cdot x \approx y \wedge y \approx z \supset x \approx z}\,3 \times (\forall R)$$

4. (FCongruence) We go a bit faster now:

$$\dfrac{\dfrac{\dfrac{}{mgu(x_1 \approx y_1, \ldots, x_n \approx y_n); \vdash c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n)}\,(\approx R)}{; x_1 \approx y_1, \ldots, x_n \approx y_n \vdash c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n)}\,n \times (\approx L)}{\forall x_1 \cdot \forall y_1 \cdot \ldots \cdot \forall x_n \cdot \forall y_n \cdot x_1 \approx y_1 \supset \ldots \supset x_n \approx y_n \supset c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n)}$$

where the last line is by $(\supset R)$ n times, then $(\forall R)$ $2n$ times.

5. (PCongruence)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{mgu(x_1 \approx y_1, \ldots, x_n \approx y_n); P(x_1, \ldots, x_n) \vdash P(y_1, \ldots, y_n)}\,(\approx)}{; x_1 \approx y_1, \ldots, x_n \approx y_n, P(x_1, \ldots, x_n) \vdash P(y_1, \ldots, y_n)}\,n \times (\approx L)}{; x_1 \approx y_1, \ldots, x_n \approx y_n \vdash P(x_1, \ldots, x_n) \supset P(y_1, \ldots, y_n)}\,(\supset R)}{\forall x_1 \cdot \forall y_1 \cdot \ldots \cdot \forall x_n \cdot \forall y_n \cdot x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n \supset P(x_1, \ldots, x_n) \supset P(y_1, \ldots, y_n)}$$

where the last line is by $(\supset R)$ n times, and $(\forall R)$ $2n$ times.

6. (Clash) Recall that $\neg F$ abbreviates $F \supset \mathbf{0}$. If c and d are distinct constructors, then $mgu(c(x_1, \ldots, x_n) \approx d(y_1, \ldots, y_m)) = \perp$, therefore:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\perp; \vdash \mathbf{0}}\,(\perp L)}{; c(x_1, \ldots, x_n) \approx d(y_1, \ldots, y_m) \vdash \mathbf{0}}\,(\approx L)}{; \vdash \neg c(x_1, \ldots, x_n) \approx d(y_1, \ldots, y_m)}\,(\supset R)}{; \vdash \forall x_1 \cdot \ldots \cdot \forall x_n \cdot \forall y_1 \cdot \ldots \cdot \forall y_m \cdot \neg c(x_1, \ldots, x_n) \approx d(y_1, \ldots, y_m)}$$

where we use $(\forall R)$ $n + m$ times in the last line.

7. (Decomp) Let $1 \leq i \leq n$, then:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{mgu(x_1 \approx y_1, \ldots, x_n \approx y_n); \vdash x_i \approx y_i}\,(\approx R)}{; c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n) \vdash x_i \approx y_i}\,(\approx L)}{; \vdash c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n) \supset x_i \approx y_i}\,(\supset R)}{; \vdash \forall x_1 \cdot \forall y_1 \cdot \ldots \cdot \forall x_n \cdot \forall y_n \cdot c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n) \supset x_i \approx y_i}$$

where the last line uses $(\forall R)$ $2n$ times.

8. (Check) If $x \neq t$ and x is free in t, then $mgu(x \approx t) = \perp$, so:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\perp; \vdash \mathbf{0}}\,(\perp L)}{; x \approx t \vdash \mathbf{0}}\,(\approx L)}{; \vdash \neg x \approx t}\,(\supset R)}{; \vdash \forall y_1 \cdot \ldots \cdot \forall y_n \cdot \neg x \approx t}\,n \times (\forall R)$$

9. (Function) Let k be the cardinality of f:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{mgu(\vec{x}_{\{1,\ldots,n\}} \approx \vec{y}_{\{1,\ldots,n\}}); P(x_1,\ldots,x_n), P(y_1,\ldots,y_n) \vdash x_i \approx y_i}\,(\approx R)}{; \vec{x}_f \approx \vec{y}_f, P(x_1,\ldots,x_n), P(y_1,\ldots,y_n), \vec{x}_{\overline{f}^n} \approx \vec{y}_{\overline{f}^n} \vdash x_i \approx y_i}\, n \times (\approx L)}{; \vec{x}_f \approx \vec{y}_f, P(x_1,\ldots,x_n), P(y_1,\ldots,y_n) \vdash x_i \approx y_i}\,(\approx\uparrow)}{; \vdash \forall x_1 \cdot \forall y_1 \cdot \ldots \cdot \forall x_n \cdot \forall y_n \cdot \bigwedge \vec{x}_f \approx \vec{y}_f \supset P(x_1,\ldots,x_n) \supset P(y_1,\ldots,y_n) \supset x_i \approx y_i}$$

where the last line uses $(\supset R)$ $k+2$ times, and $(\forall R)$ $2n$ times.

When the last rule in the ND proof of $T_{fin}, \overline{\sigma}, \Lambda \vdash F$ is not (Ax), then we use standard natural-deduction-to-sequent-translation arguments to build an LKc$_{\approx}$ proof of $\sigma; \Lambda \vdash F$. Let (W) denote the admissible rule of weakening (Lemma 3.11), and $(Close')$ denote the admissible rule generalizing $(\approx)$ to non-atomic formulae introduced in Lemma 3.10. We examine each non-(Ax) ND rule in turn:

$(\supset I)$: this is by $(\supset R)$. That is, by induction we have a proof of $\sigma; \Lambda, F \vdash G$, so by $(\supset R)$, we get the desired proof of $\sigma; \Lambda \vdash F \supset G$.

$(\supset E)$:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \sigma; \Lambda \vdash F \supset G\end{matrix}}{\sigma; \Lambda \vdash F \supset G, G}\,(W) \qquad \dfrac{\dfrac{}{\sigma; \Lambda, G \vdash G}\,(Close') \quad \dfrac{\begin{matrix}\vdots\\ \sigma; \Lambda \vdash F\end{matrix}}{\sigma; \Lambda \vdash F, G}\,(W)}{\sigma; \Lambda, F \supset G \vdash G}\,(\supset L)}{\sigma; \Lambda \vdash G}\,(Cut)$$

$(\neg\neg E)$:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\\ \sigma; \Lambda, \neg F \vdash \mathbf{0}\end{matrix} \quad \dfrac{}{\sigma; \Lambda, \neg F, \mathbf{0} \vdash}\,(\mathbf{0}L)}{\sigma; \Lambda, \neg F \vdash}\,(Cut)}{\sigma; \Lambda, \neg F \vdash F}\,(W) \qquad \dfrac{\dfrac{}{\sigma; \Lambda, F \vdash \mathbf{0}, F}\,(Close')}{\sigma; \Lambda \vdash \neg F, F}\,(\supset R)}{\sigma; \Lambda \vdash F}\,(Cut)$$

$(\mathbf{0}E)$:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \sigma; \Lambda \vdash \mathbf{0}\end{matrix}}{\sigma; \Lambda, \neg F \vdash \mathbf{0}}\,(W)}{\sigma; \Lambda \vdash F}\,(\neg\neg E)$$

where we use the admissible $(\neg\neg E)$, as derived above.

$(\wedge I)$:

$$\dfrac{\begin{matrix}\vdots\\ \sigma; \Lambda \vdash F\end{matrix} \quad \begin{matrix}\vdots\\ \sigma; \Lambda \vdash G\end{matrix}}{\sigma; \Lambda \vdash F \wedge G}\,(\wedge R)$$

$(\wedge E_i)$, $i = 1, 2$:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \sigma; \Lambda \vdash F_1 \wedge F_2\end{matrix}}{\sigma; \Lambda \vdash F_1 \wedge F_2, F_i}\,(W) \qquad \dfrac{\dfrac{}{\sigma; \Lambda, F_1, F_2 \vdash F_i}\,(Close')}{\sigma; \Lambda, F_1 \wedge F_2 \vdash F_i}\,(\wedge L)}{\sigma; \Lambda \vdash F_i}\,(Cut)$$

$(\vee I_i)$, $i = 1, 2$:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda \vdash F_i\end{array}}{\sigma; \Lambda \vdash F_1, F_2}\,(W)}{\sigma; \Lambda \vdash F_1 \vee F_2}\,(\vee R)$$

$(\vee E)$:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda \vdash F \vee G\end{array}}{\sigma; \Lambda \vdash F \vee G, H}\,(W) \quad \dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda, F \vdash H\end{array} \quad \begin{array}{c}\vdots\\ \sigma; \Lambda, G \vdash H\end{array}}{\sigma; \Lambda, F \vee G \vdash H}\,(\vee L)}{\sigma; \Lambda \vdash H}\,(Cut)$$

$(\forall I)$: this is exactly $(\forall R)$, assuming that y is not free in $\forall x_\tau \cdot F$ either: we can do this without loss of generality, by replacing y by a fresh variable throughout the ND proof if necessary first.

$(\forall E)$: let t be of type τ, then:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda \vdash \forall x_\tau \cdot F\end{array}}{\sigma; \Lambda \vdash \forall x_\tau \cdot F, F[x := t]}\,(W) \quad \dfrac{\dfrac{}{\sigma; \Lambda, \forall x_\tau \cdot F, F[x := t] \vdash F[x := t]}\,(Close')}{\sigma; \Lambda, \forall x_\tau \cdot F \vdash F[x := t]}\,(\forall L)}{\sigma; \Lambda \vdash F[x := t]}\,(Cut)$$

$(\exists I)$: let t be of type τ, then:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda \vdash F[x := t]\end{array}}{\sigma; \Lambda \vdash F[x := t], \exists x_\tau \cdot F}\,(W)}{\sigma; \Lambda \vdash \exists x_\tau \cdot F}\,(\exists R)$$

$(\exists E)$:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda \vdash \exists x_\tau \cdot F\end{array}}{\sigma; \Lambda \vdash \exists x_\tau \cdot F, G}\,(W) \quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda \vdash \forall x_\tau \cdot F \supset G\end{array}}{\sigma; \Lambda \vdash F \supset G}\,(\forall E)}{\sigma; \Lambda, F \vdash G}\,(\supset U)}{\sigma; \Lambda, \exists x_\tau \cdot F \vdash G}\,(\exists L)}{\sigma; \Lambda \vdash G}\,(Cut)$$

where $(\supset U)$ is the following derived rule:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma; \Lambda \vdash F \supset G\end{array}}{\sigma; \Lambda, F \vdash F \supset G}\,(W) \quad \dfrac{}{\sigma; \Lambda, F \vdash F}\,(Close')}{\sigma; \Lambda, F \vdash G}\,(\supset E)$$

$\square$

By Lemma 3.16 and (d), $\sigma; \Gamma, \neg\Delta \vdash \mathbf{0}$ is therefore provable in LKc$_\approx$. Using (Cut) together with a proof of $\sigma; \Gamma, \neg\Delta, \mathbf{0} \vdash$ obtained by $(\mathbf{0}L)$, we get a proof of $\sigma; \Gamma, \neg\Delta \vdash$. Letting Δ be $G_1, \ldots, G_n$, we claim that there is an LKc$_\approx$ proof of $\sigma; \Gamma \vdash \Delta$. This is by induction on n. If $n = 0$, this is obvious, otherwise, by induction there is an LKc$_\approx$ proof of $\sigma; \Gamma, \neg G_n \vdash G_1, \ldots, G_{n-1}$, and we build the following:

$$\dfrac{\dfrac{\dfrac{\vdots}{\sigma; \Gamma, \neg G_n \vdash G_1, \ldots, G_{n-1}}}{\sigma; \Gamma, \neg G_n \vdash G_1, \ldots, G_{n-1}, G_n}(W) \quad \dfrac{\dfrac{}{\sigma; \Gamma, G_n \vdash G_1, \ldots, G_{n-1}, G_n, \mathbf{0}}(Close')}{\sigma; \Gamma \vdash G_1, \ldots, G_{n-1}, G_n, \neg G_n}(\supset R)}{\sigma; \Gamma \vdash G_1, \ldots, G_n}(Cut)$$

This finishes the proof. □

3.4 Cut Elimination

Theorem 3.12 is nice, but heavily relies on the (Cut) rule. The purpose of this section is to show that we never actually need (Cut). We roughly follow [GLT89], Chapter 13, except that our (Cut) rule is an additive instead of a multiplicative cut. The latter is an inessential difference once we have weakening and contraction. We shall see that the latter are derived rules in LKc$_\approx$, even without (Cut).

Referring to the notations of Figure 1, call a formula occurrence *principal* in a rule if it is explicitly shown in the conclusion of this rule (i.e., if it is not an occurrence inside the Γ or Δ components), and *active* in some premise if it is explicitly shown in this premise (again, this means not in Γ or Δ).

Define the *degree* $\partial(F)$ of a formula F by: $\partial(A) =_{\rm df} \partial(s \approx t) =_{\rm df} \partial(\mathbf{0}) =_{\rm df} 1$, $\partial(F \wedge G) =_{\rm df} \partial(F \vee G) =_{\rm df} \partial(F \supset G) =_{\rm df} \max(\partial(F), \partial(G)) + 1$, $\partial(\forall x_\tau \cdot F) =_{\rm df} \partial(\exists x_\tau \cdot F) =_{\rm df} \partial(F) + 1$.

In an instance of (Cut), the occurrences of the formula F that are active in the premises are called the *cut formulas*. Their degree is the same, and is called the *degree* of the cut rule. The *degree* $\partial(\pi)$ of a proof π is the sup of the degrees of its cut rules, and 0 if π is cut-free (i.e., if π does not use (Cut)). Let $h(\pi)$ denote the *height* of a proof, defined as 1 is it ends in a rule with no premise ($(\bot L)$, $(\approx)$, $(\approx R)$, $(\mathbf{0}L)$), as $\max(h(\pi_1), h(\pi_2))$ if it ends with a cut with premises proved by π_1 and π_2 respectively, otherwise as $\max_i(h(\pi_i)) + 1$, where π_i ranges over all immediate subproofs of π.

We first prove a few technical lemmas.

Lemma 3.17 *Let σ be an idempotent substitution, such that $s\sigma = s'\sigma$ and $t\sigma = t'\sigma$; then $mgu(s \approx t, \overline{\sigma}) \equiv mgu(s' \approx t', \overline{\sigma})$.*

Proof. First, let σ' be $mgu(s \approx t, \overline{\sigma})$. We show that σ' is $\bot$ or unifies both $s' \approx t'$ and $\overline{\sigma}$. If σ' is not $\bot$, then since $\sigma \succeq \sigma'$, σ' unifiers every pair of terms that σ unifies, in particular $\overline{\sigma}$; indeed, σ unifies $\overline{\sigma}$ because σ is idempotent. Moreover, $s'\sigma' = s\sigma'$ (since σ, hence σ', unifies s and s') $= t\sigma'$ (since σ' unifies s and t by construction) $= t'\sigma'$ (since σ, hence σ', unifies t and t'). In short, $mgu(s' \approx t', \overline{\sigma}) \succeq mgu(s \approx t, \sigma)$. Symmetrically, $mgu(s \approx t, \overline{\sigma}) \succeq mgu(s' \approx t', \sigma)$. □

Lemma 3.18 (Substitution Replacement) *Assume $\sigma \equiv \sigma'$. If $\sigma; \Gamma \vdash \Delta$ has a proof of degree d and height h in* LKc$_\approx$*, then so does $\sigma'; \Gamma \vdash \Delta$.*

Proof. Easy structural induction on the given proof of $\sigma; \Gamma \vdash \Delta$. □

Define $\Gamma\sigma = \Gamma'\sigma$ for multisets Γ and Γ' by: $(F_1, \ldots, F_n)\sigma = (G_1, \ldots, G_m)\sigma$ if and only if $n = m$ and for some permutation p of $\{1, \ldots, n\}$, for all i, $1 \leq i \leq n$, $F_i\sigma = G_{p(i)}\sigma$.

Lemma 3.19 (Rewriting) *If $\sigma; \Gamma \vdash \Delta$ has an* LKc$_\approx$ *proof of degree d and height n, $\sigma \neq \bot$, and $\Gamma\sigma = \Gamma'\sigma$, $\Delta\sigma = \Delta'\sigma$, then $\sigma; \Gamma' \vdash \Delta'$ has an* LKc$_\approx$ *proof of degree at most d and height at most n.*

Proof. By structural induction on the given LKc$_\approx$ proof π of $\sigma; \Gamma \vdash \Delta$. Examine the last rule in the proof:

- $(\bot L)$: trivial.

- ($\approx$): π proves $\sigma; \Gamma_1, A \vdash B, \Delta_1$, and $(\Gamma_1, A)\sigma = \Gamma'\sigma$, $(B, \Delta_1)\sigma = \Delta'\sigma$, so Γ' is of the form Γ'_1, A', with $A\sigma = A'\sigma$, and Δ' is of the form B', Δ'_1, with $B\sigma = B'\sigma$. In particular, we produce the desired proof of $\sigma; \Gamma'_1, A' \vdash B', \Delta'_1$ by ($\approx$); its degree is 0, its height is 1, just like π. The ($\approx R$) case is similar.

- ($\approx\uparrow$): π proves $\sigma; \Gamma_1, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \Delta$ from the premise $\sigma; \Gamma_1, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n), \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta$, where $f \in \phi(P)$, and $\vec{s}_f = \vec{t}_f$, and $(\Gamma_1, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n))\sigma = \Gamma'\sigma$, and $\Delta\sigma = \Delta'\sigma$. In particular, Γ' is of the form $\Gamma'_1, P(s'_1, \ldots, s'_n), P(t'_1, \ldots, t'_n)$, where $\Gamma_1\sigma = \Gamma'_1\sigma$, $s_i\sigma = s'_i\sigma$ and $t_i\sigma = t'_i\sigma$ for every i, $1 \leq i \leq n$. By induction hypothesis, there is a proof of depth at most d and height at most $n-1$ of $\sigma; \Gamma'_1, P(s'_1, \ldots, s'_n), P(t'_1, \ldots, t'_n), \vec{s'}_{\overline{f}^n} \approx \vec{t'}_{\overline{f}^n} \vdash \Delta'$. Since $\vec{s'}_f = \vec{t'}_f$, applying ($\approx\uparrow$) once yields the desired proof of depth at most d and height at most n of $\sigma; \Gamma'_1, P(s'_1, \ldots, s'_n), P(t'_1, \ldots, t'_n) \vdash \Delta'$.

- ($\approx L$): π proves $\sigma; \Gamma_1, s \approx t \vdash \Delta$ from the premise $mgu(s \approx t, \overline{\sigma}); \Gamma_1 \vdash \Delta$, and $(\Gamma_1, s \approx t)\sigma = \Gamma'\sigma$, $\Delta\sigma = \Delta'\sigma$. So Γ' is of the form $\Gamma'_1, s' \approx t'$ with $\Gamma_1\sigma = \Gamma'_1\sigma$, $s\sigma = s'\sigma$, and $t\sigma = t'\sigma$. By induction hypothesis, there is a proof of depth at most d and of height at most $n-1$ of $mgu(s \approx t, \overline{\sigma}); \Gamma'_1 \vdash \Delta'$. By Lemma 3.17, $mgu(s \approx t, \overline{\sigma}) \equiv mgu(s' \approx t', \overline{\sigma})$, so by Lemma 3.18, there is a proof of depth at most d and of height at most $n-1$ of $mgu(s' \approx t', \overline{\sigma}); \Gamma'_1 \vdash \Delta'$. Using ($\approx L$), we get a proof of depth d and height at most n of $\sigma; \Gamma'_1, s' \approx t' \vdash \Delta'$.

- For all other rules, this is a straightforward appeal to the induction hypothesis.

$\square$

Lemma 3.20 (Weakening) *If $\sigma; \Gamma \vdash \Delta$ has a proof of degree d and height n in* LKc$_\approx$*, then $\sigma'; \Gamma', \Gamma \vdash \Delta, \Delta'$ has a proof of degree at most d and height at most n in* LKc$_\approx$*, for every $\sigma' \equiv mgu(\overline{\sigma}, E)$, for every system of equations E, for every multisets of formulas Γ' and Δ'.*

Proof. Replay the proof of Lemma 3.11, taking care of degrees and heights. $\square$

The main difficulty in showing cut-elimination occurs when considering cuts between proofs ending on quantifier rules whose principal formulas are the cut formulae. We need to show that whenever we can prove a given sequent in LKc$_\approx$, then we can prove the same sequent with some variable y replaced by some term t of the same type, by a proof of no larger degree and height. This will be established, in a slightly different form, in Lemma 3.22 below.

Given two systems of equations E_1 and E_2, write $E_1 \Vdash E_2$ if and only if E_1 is not unifiable, or $mgu(E_1) \Vdash E_2$. For short, we agree that $\perp \Vdash E$ for every system of equations E (we say that $\perp$ unifies every system of equations), that $\perp \Vdash \perp$, and that $E \Vdash \perp$ if and only if E is not unifiable. With these conventions, $E_1 \Vdash E_2$ if and only if $mgu(E_1) \Vdash E_2$. Notice that an equivalent definition is: $E_1 \Vdash E_2$ if and only if every unifier of E_1 is also a unifier of E_2, where $\perp$ has no unifier by convention.

Lemma 3.21 *The following hold:*

(*i*) *If $E_1 \Vdash E_2$, then $E_1\sigma \Vdash E_2\sigma$ for every substitution σ.*

(*ii*) *If $E_1, E_2 \Vdash E_3$, and E_2 is a collection of reflexivity equations (equations of the form $t \approx t$), then $E_1 \Vdash E_3$.*

(*iii*) *If $E_1\sigma, \overline{\sigma} \Vdash E_2$, then $E_1, \overline{\sigma} \Vdash E_2$.*

(*iv*) *If $E_1, \overline{\sigma} \Vdash E_2\sigma$, then also $E_1, \overline{\sigma} \Vdash E_2$.*

(*v*) *If $E_1 \Vdash E_3$, then $E_1, E_2 \Vdash E_3$.*

(*vi*) *If θ is an idempotent substitution, then $E_1\theta \Vdash E_2\theta$ if and only if $E_1, \overline{\theta} \Vdash E_2$.*

Proof. (i): let σ_1 be $mgu(E_1)$ and σ_1' be $mgu(E_1\sigma)$; if $\sigma_1' = \bot$, the result is clear. Otherwise, σ_1' unifies $E_1\sigma$, that is, $E_1\sigma\sigma_1'$ is a collection of reflexivity equations, so $\sigma\sigma_1'$ unifies E_1. It follows that $\sigma\sigma_1'$ is an instance of $mgu(E_1)$, that is, of σ_1. Since $E_1 \Vdash E_2$, that is, since σ_1 unifies E_2, its instance $\sigma\sigma_1'$ also unifies E_2; it follows that σ_1' unifies $E_2\sigma$.

(ii): let σ be $mgu(E_1)$; it is easy to see that $\sigma \equiv mgu(E_1, E_2)$ as well, so σ unifies E_3.

(iii): let σ' be $mgu(E_1, \overline{\sigma})$. Then σ' unifies every equation $s \approx t$ in E_1, and also s with $s\sigma$ and t with $t\sigma$ (since σ' unifies $\overline{\sigma}$); so it unifies $s\sigma$ with $t\sigma$, and this for every equation $s \approx t$ in E_1. So σ' unifies $E_1\sigma$. Moreover, by definition σ' unifies $\overline{\sigma}$, so σ' unifies $E_1\sigma, \overline{\sigma}$. Since $E_1\sigma, \overline{\sigma} \Vdash E_2$, it follows that σ' unifies E_2.

(iv): let σ' be $mgu(E_1, \overline{\sigma})$. By assumption, σ' unifies $E_2\sigma$, so σ' unifies every equation $s\sigma \approx t\sigma$, for every $s \approx t$ in E_2. Since σ' unifies $\overline{\sigma}$, σ' unifies s with $s\sigma$ and t with $t\sigma$, so σ' unifies $s \approx t$, for every $s \approx t$ in E_2. Therefore, σ' unifies E_2.

(v): let σ be $mgu(E_1, E_2)$. Then σ unifies E_1, in particular, so it is an instance of $mgu(E_1)$. Since $E_1 \Vdash E_3$, it follows that σ also unifies E_3.

(vi), if direction: since $E_1, \overline{\theta} \Vdash E_2$, it follows that $E_1\theta, \overline{\theta}\theta \Vdash E_2\theta$ by (i). But $\overline{\theta}\theta$ is by definition the set of all equations $x\theta \approx x\theta\theta$, $x \in \operatorname{dom}\theta$. Since θ is idempotent, $x\theta\theta = x\theta$, so $\overline{\theta}\theta$ is a set of reflexivity equations, therefore by (ii), $E_1\theta \Vdash E_2\theta$.

(vi), only if direction: since $E_1\theta \Vdash E_2\theta$, by (v) $E_1\theta, \overline{\theta} \Vdash E_2\theta$, so by (iii) and (iv), $E_1, \overline{\theta} \Vdash E_2$. $\square$

Lemma 3.22 *For every idempotent substitution θ, if $\sigma; \Gamma \vdash \Delta$ has a proof of degree d and height n in* LKc$_\approx$*, then $mgu(\overline{\sigma}\theta); \Gamma\theta \vdash \Delta\theta$ has a proof of degree at most d and height at most n.*

Proof. That is, we can apply substitutions θ to whole proofs. We prove the result by structural induction on the given proof of $\sigma; \Gamma \vdash \Delta$. We examine the last rule: if this is $(\bot L)$, the result is trivial. Otherwise, if $mgu(\overline{\sigma}\theta) = \bot$, then $mgu(\overline{\sigma}\theta); \Gamma\theta \vdash \Delta\theta$ has a proof by $(\bot L)$, with degree 0 and height 1, so the result is clear. So assume that $\sigma \neq \bot$, $mgu(\overline{\sigma}\theta) \neq \bot$, and examine each rule in turn:

- $(\approx R)$: we have derived $\sigma; \Gamma \vdash s \approx t, \Delta$, using $s\sigma = t\sigma$. The latter means $\overline{\sigma} \Vdash s \approx t$. By Lemma 3.21 (v), $\overline{\sigma}, \overline{\theta} \Vdash s \approx t$. By Lemma 3.21 (i), $\overline{\sigma}\theta, \overline{\theta}\theta \Vdash s\theta \approx t\theta$. Since θ is idempotent, $\overline{\theta}\theta$ is a collection of reflexivity equations, so by Lemma 3.21 (ii), $\overline{\sigma}\theta \Vdash s\theta \approx t\theta$. This means precisely that $\overline{\sigma}\theta$ unifies $s\theta \approx t\theta$, so $mgu(\overline{\sigma}\theta); \Gamma\theta \vdash \Delta\theta, s\theta \approx t\theta$ is provable by $(\approx R)$ again. The case of $(\approx)$ is similar.

- $(\approx L)$: we have derived $\sigma; \Gamma, s \approx t \vdash \Delta$ from a proof of $mgu(s \approx t, \overline{\sigma}); \Gamma \vdash \Delta$ of depth d and height $n-1$. By induction hypothesis: $(*)$ there is a proof of $mgu(\overline{mgu(s \approx t, \overline{\sigma})}\theta); \Gamma\theta \vdash \Delta\theta$ of depth d at most and height $n-1$ at most.

  We claim that $mgu(\overline{\sigma}\theta, s\theta \approx t\theta)$ is an instance of $mgu(\overline{mgu(s \approx t, \overline{\sigma})}\theta)$. Indeed, $\overline{\sigma}, s \approx t \Vdash \overline{mgu(s \approx t, \overline{\sigma})}$ trivially, so by Lemma 3.21 (i) $\overline{\sigma}\theta, s\theta \approx t\theta \Vdash \overline{mgu(s \approx t, \overline{\sigma})}\theta$. By definition, this means that $mgu(\overline{\sigma}\theta, s\theta \approx t\theta)$ unifies $\overline{mgu(s \approx t, \overline{\sigma})}\theta$. Therefore, $mgu(\overline{\sigma}\theta, s\theta \approx t\theta)$ is indeed an instance of $mgu(\overline{mgu(s \approx t, \overline{\sigma})}\theta)$.

  In particular, $mgu(\overline{\sigma}\theta, s\theta \approx t\theta)$ is equivalent w.r.t. $\equiv$ to the mgu of $\overline{mgu(\overline{mgu(s \approx t, \overline{\sigma})}\theta)}$ and some system of equations E: just take $E =_{\mathrm{df}} \overline{mgu(\overline{\sigma}\theta, s\theta \approx t\theta)}$ itself. It follows from $(*)$, Lemma 3.20 and Lemma 3.18 that we can build a proof of $mgu(\overline{\sigma}\theta, s\theta \approx t\theta); \Gamma\theta \vdash \Delta\theta$ of depth d at most and height $n-1$ at most. By rule $(\approx L)$, we infer a proof of $mgu(\overline{\sigma}\theta); \Gamma\theta, s\theta \approx t\theta \vdash \Delta\theta$ of depth d at most and height n at most, as desired.

- $(\approx\uparrow)$: we have derived $\sigma; \Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \Delta$ from a proof of the sequent $\sigma; \Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n), \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta$ of depth d and height $n-1$, where $f \in \phi(P)$, $\sigma \neq \bot$, and $\vec{s}_f\sigma = \vec{t}_f\sigma$. In other words, $\overline{\sigma} \Vdash \vec{s}_f \approx \vec{t}_f$, so $\overline{\sigma}\theta \Vdash \vec{s}_f\theta \approx \vec{t}_f\theta$ by Lemma 3.21 (i), therefore $mgu(\overline{\sigma}\theta)$ unifies $\vec{s}_f\theta \approx \vec{t}_f\theta$. We conclude by the induction hypothesis and rule $(\approx\uparrow)$ again.

- $(\forall R)$: we have derived $\sigma; \Gamma \vdash \forall x_\tau \cdot F, \Delta$ from a proof of $\sigma; \Gamma \vdash F[x := y_\tau], \Delta$ of depth d and height $n-1$, where y is a variable that does not occur free in $\overline{\sigma}, \Gamma, \forall x_\tau \cdot F, \Delta$. Without loss of generality, we may assume that $\mathrm{dom}\,\theta$ is included in the set of free variables of $\overline{\sigma}, \Gamma, \forall x_\tau \cdot F, \Delta$, so $y \notin \mathrm{dom}\,\theta$. By induction hypothesis, there is a proof of $mgu(\overline{\sigma}\theta); \Gamma\theta \vdash F[x := y_\tau]\theta, \Delta\theta$ of depth at most d and height at most $n-1$. Since $y \notin \mathrm{dom}\,\theta$, and since by α-renaming we may assume that x does not occur free in any term $z\theta$, $z \in \mathrm{dom}\,\theta$, $F[x := y_\tau]\theta = F\theta[x := y_\tau]$. We conclude by rule $(\forall R)$. The case of rule $(\exists L)$ is similar.

The cases of all other rules are straightforward. $\square$

Lemma 3.23 (Reflexivity Elimination) *Whenever $s\sigma = t\sigma$, if $\sigma; \Gamma, s \approx t \vdash \Delta$ has a proof of degree d and height n in* LKc$_\approx$*, then $\sigma; \Gamma \vdash \Delta$ has a proof of degree at most d and height at most n.*

Proof. By structural induction on the proof π of $\sigma; \Gamma, s \approx t \vdash \Delta$ of degree d and height n, we show the result under the slightly generalized assumption that $\sigma = \bot$ or $s\sigma = t\sigma$. When the last rule of π is $(\approx L)$ and $s \approx t$ is the principal formula, we have deduced $\sigma; \Gamma, s \approx t \vdash \Delta$ from a proof π' of $mgu(s \approx t, \overline{\sigma}); \Gamma \vdash \Delta$ of degree d and height $n-1$. Then $mgu(s \approx t, \overline{\sigma}) \equiv \sigma$, since σ unifies $s \approx t$, so by Lemma 3.18 there is a proof of $\sigma; \Gamma \vdash \Delta$ of degree d and height $n-1$, and the claim is proved. All other cases are direct appeals to the induction hypothesis. $\square$

Lemma 3.24 *Given any* LKc$_\approx$ *proof of the form:*

$$\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \sigma; \Gamma \vdash F_0, \Delta\end{array} \quad \begin{array}{c}\vdots\,\pi_2\\ \sigma; \Gamma, F_0 \vdash \Delta\end{array}}{\sigma; \Gamma \vdash \Delta}\,(Cut)$$

where $\partial(F_0) = d$, $\partial(\pi_1) < d$ and $\partial(\pi_2) < d$, then we can build effectively an LKc$_\approx$ *proof π of $\sigma; \Gamma \vdash \Delta$ with $\partial(\pi) < d$.*

Proof. By induction on $h(\pi_1) + h(\pi_2)$. If F_0 is not principal in the last rule R_1 of π_1 or F_0 is not principal in the last rule R_2 of π_2, then this is by induction hypothesis (in case the rule has no premise, this terminates the proof). So deal with the cases where F_0 is principal in R_1 and in R_2. We have the following cases:

- $R_1 = (\approx)$, then F_0 is a non-equality atom, hence R_2 must be one of the following rules, since these are the only ones where a non-equality atom may be principal on the left of the $\vdash$ sign:
- $R_2 = (\approx)$:

$$\dfrac{\dfrac{\sigma \neq \bot,\ A\sigma = F_0\sigma}{\sigma; \Gamma, A \vdash F_0, B, \Delta}\,(\approx) \quad \dfrac{\sigma \neq \bot,\ F_0\sigma = B\sigma}{\sigma; \Gamma, A, F_0 \vdash B, \Delta}\,(\approx)}{\sigma; \Gamma, A \vdash B, \Delta}\,(Cut) \rightsquigarrow \dfrac{}{\sigma; \Gamma, A \vdash B, \Delta}\,(\approx)$$

since indeed $A\sigma = B\sigma$.

- $R_2 = (\approx\uparrow)$, then F_0 is of the form $P(s_1, \ldots, s_n)$; let $f \in \phi(P)$, with $\sigma \neq \bot$, $\vec{s}_f\sigma = \vec{t}_f\sigma$. The left premise of the cut is $\ldots, A \vdash P(s_1, \ldots, s_n), \ldots$ with $A\sigma = P(s_1, \ldots, s_n), \sigma$, and the right premise is $\ldots, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \ldots$, and we have two cases, according to whether A and $P(t_1, \ldots, t_n)$ are equal or not.

In the first case, when A and $P(t_1, \ldots, t_n)$ are not equal, the cut looks like:

$$\dfrac{\dfrac{A\sigma = P(s_1, \ldots, s_n)\sigma}{\begin{array}{c}\sigma; \Gamma, P(t_1, \ldots, t_n), A \vdash\\ P(s_1, \ldots, s_n), \Delta\end{array}}\,(\approx) \quad \dfrac{\begin{array}{c}\vdots\,\pi_2'\\ \sigma; \Gamma, A, P(s_1, \ldots, s_n),\\ P(t_1, \ldots, t_n), \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta\end{array}}{\begin{array}{c}\sigma; \Gamma, A, P(s_1, \ldots, s_n),\\ P(t_1, \ldots, t_n) \vdash \Delta\end{array}}\,(\approx\uparrow)}{\sigma; \Gamma, A, P(t_1, \ldots, t_n) \vdash \Delta}\,(Cut)$$

and we transform it as follows. First, $A\sigma = P(s_1,\dots,s_n)\sigma$, so A is of the form $P(u_1,\dots,u_n)$, with $u_i\sigma = s_i\sigma$ for every i, so $u_i\sigma = t_i\sigma$ for every $i \in f$ and we produce:

$$\dfrac{\dfrac{\dfrac{}{\begin{array}{c}\sigma;\Gamma,A,P(t_1,\dots,t_n),\\ \vec{u}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash A,\Delta\end{array}}(\approx) \qquad \begin{array}{c}\vdots\,\pi_2''\\ \sigma;\Gamma,A,A,P(t_1,\dots,t_n),\\ \vec{u}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta\end{array}}{\sigma;\Gamma,A,P(t_1,\dots,t_n),\vec{u}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta}(*)}{\sigma;\Gamma,A,P(t_1,\dots,t_n) \vdash \Delta}(\approx\uparrow)$$

where π_2'' is obtained from π_2' by Lemma 3.19, which allows us to replace $P(t_1,\dots,t_n)$ by A, and $(*)$ is by induction hypothesis.

In the second case, when A and $P(t_1,\dots,t_n)$ are equal, the cut looks like:

$$\dfrac{\dfrac{P(t_1,\dots,t_n)\sigma = P(s_1,\dots,s_n)\sigma}{\sigma;\Gamma,P(t_1,\dots,t_n) \vdash P(s_1,\dots,s_n),\Delta}(\approx) \qquad \dfrac{\begin{array}{c}\vdots\,\pi_2'\\ \sigma;\Gamma,P(s_1,\dots,s_n),\\ P(t_1,\dots,t_n),\vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n} \vdash \Delta\end{array}}{\begin{array}{c}\sigma;\Gamma,P(s_1,\dots,s_n),\\ P(t_1,\dots,t_n) \vdash \Delta\end{array}}(\approx\uparrow)}{\sigma;\Gamma,P(t_1,\dots,t_n) \vdash \Delta}(Cut)$$

and we transform it as follows. Notice that, since $s_i\sigma = t_i\sigma$ for every i, the equations $s_i \approx t_i$, $i \in \overline{f}^n$, in the conclusion of π_2' are all such that $s_i\sigma = t_i\sigma$. So by Lemma 3.23, there is a proof π_2'' of $\sigma;\Gamma,P(s_1,\dots,s_n),P(t_1,\dots,t_n) \vdash \Delta$ of depth at most d and height at most $n-1$. This allows us to produce:

$$\dfrac{\dfrac{}{\sigma;\Gamma,P(t_1,\dots,t_n) \vdash P(s_1,\dots,s_n),\Delta}(\approx) \qquad \begin{array}{c}\vdots\,\pi_2''\\ \sigma;\Gamma,P(s_1,\dots,s_n),P(t_1,\dots,t_n) \vdash \Delta\end{array}}{\sigma;\Gamma,P(t_1,\dots,t_n) \vdash \Delta}(*)$$

- $R_1 = (\approx R)$, and F_0 is an equality $s \approx t$, then since F_0 is principal in R_2 as well, R_2 must be $(\approx L)$, and we transform:

$$\dfrac{\dfrac{\sigma \neq \bot,\ s\sigma = t\sigma}{\sigma;\Gamma \vdash s \approx t,\Delta} \qquad \dfrac{\begin{array}{c}\vdots\,\pi_2'\\ mgu(s \approx t,\overline{\sigma});\Gamma \vdash \Delta\end{array}}{\sigma;\Gamma,s \approx t \vdash \Delta}}{\sigma;\Gamma \vdash \Delta}(Cut)$$

into the proof of $\sigma;\Gamma \vdash \Delta$ obtained from π_2' by Lemma 3.18, noticing that, since $s\sigma = t\sigma$, $mgu(s \approx t,\overline{\sigma}) \equiv mgu(\overline{\sigma}) \equiv \sigma$.

- $R_1 = (\supset R)$, so $R_2 = (\supset L)$ (in particular R_2 is not $(\approx)$, since the principal formula in $(\approx)$ cannot be an implication); we transform:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\pi_1'\\ \sigma;\Gamma,F \vdash G,\Delta\end{array}}{\sigma;\Gamma \vdash F \supset G,\Delta}(\supset R) \qquad \dfrac{\begin{array}{c}\vdots\,\pi_2'\\ \sigma;\Gamma,G \vdash \Delta\end{array} \quad \begin{array}{c}\vdots\,\pi_2''\\ \sigma;\Gamma \vdash F,\Delta\end{array}}{\sigma;\Gamma,F \supset G \vdash \Delta}(\supset L)}{\sigma;\Gamma \vdash \Delta}(Cut)$$

where π_1', π_2', π_2'' are proofs of degrees $< d$, and of heights at most $h(\pi_1)-1$, $h(\pi_2)-1$,

$h(\pi_2) - 1$ respectively, into the following proof π:

$$\dfrac{\begin{matrix}\vdots\ \pi_2''\\ \sigma;\Gamma\vdash F,\Delta\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\ \pi_1'\\ \sigma;\Gamma,F\vdash G,\Delta\end{matrix}\quad \begin{matrix}\vdots\ \pi_2'\\ \sigma;\Gamma,G\vdash\Delta\end{matrix}}{\sigma;\Gamma,F\vdash\Delta}\,(*)}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

where $(*)$ is obtained by first weakening π_2' by Lemma 3.20 to a proof π_2''' of $\sigma;\Gamma,G\vdash F,\Delta$, of degree $< d$ and of height at most $h(\pi_2)-1$, then by using the induction hypothesis, since $h(\pi_1') + h(\pi_2''') \leq h(\pi_1) + h(\pi_2) - 2 < h(\pi_1) + h(\pi_2)$; this yields a proof π' of $\sigma;\Gamma,F\vdash\Delta$ of degree $< d$. Since $\partial(F) < d$, it follows that π is again of degree $< d$.

- $R_1 = (\wedge R)$, so $R_2 = (\wedge L)$; we transform:

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \pi_1'\\ \sigma;\Gamma\vdash F,\Delta\end{matrix}\quad \begin{matrix}\vdots\ \pi_1''\\ \sigma;\Gamma\vdash G,\Delta\end{matrix}}{\sigma;\Gamma\vdash F\wedge G,\Delta}\,(\wedge R) \qquad \dfrac{\begin{matrix}\vdots\ \pi_2'\\ \sigma;\Gamma,F,G\vdash\Delta\end{matrix}}{\sigma;\Gamma,F\wedge G\vdash\Delta}\,(\wedge L)}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

into:

$$\dfrac{\begin{matrix}\vdots\ \pi_1''\\ \sigma;\Gamma\vdash G,\Delta\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\ \pi_1'\\ \sigma;\Gamma\vdash F,\Delta\end{matrix}\quad \begin{matrix}\vdots\ \pi_2'\\ \sigma;\Gamma,F,G\vdash\Delta\end{matrix}}{\sigma;\Gamma,G\vdash\Delta}\,(*)}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

where $(*)$ is by first weakening π_1' to a proof of $\sigma;\Gamma,G\vdash F,\Delta$, then applying the induction hypothesis. As above, the resulting proof has degree $< d$.

- $R_1 = (\vee R)$, $R_2 = (\vee L)$; we transform:

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \pi_1'\\ \sigma;\Gamma\vdash F,G,\Delta\end{matrix}}{\sigma;\Gamma\vdash F\vee G,\Delta}\,(\vee R) \qquad \dfrac{\begin{matrix}\vdots\ \pi_2'\\ \sigma;\Gamma,F\vdash\Delta\end{matrix}\quad \begin{matrix}\vdots\ \pi_2''\\ \sigma;\Gamma,G\vdash\Delta\end{matrix}}{\sigma;\Gamma,F\vee G\vdash\Delta}\,(\vee L)}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

into:

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \pi_1'\\ \sigma;\Gamma\vdash F,G,\Delta\end{matrix}\quad \begin{matrix}\vdots\ \pi_2'\\ \sigma;\Gamma,F\vdash\Delta\end{matrix}}{\sigma;\Gamma\vdash G,\Delta}\,(*) \qquad \begin{matrix}\vdots\ \pi_2''\\ \sigma;\Gamma,G\vdash\Delta\end{matrix}}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

where $(*)$ is by first weakening π_2' to a proof of $\sigma;\Gamma,F\vdash G,\Delta$, then applying the induction hypothesis. As above, the resulting proof has degree $< d$.

- $R_1 = (\forall R)$, $R_2 = (\forall L)$; let t be a term of type τ, and assume that y_τ is not free in $\overline{\sigma},\Gamma,\Delta$. We transform:

$$\dfrac{\dfrac{\begin{matrix}\vdots\ \pi_1'\\ \sigma;\Gamma\vdash F[x:=y_\tau],\Delta\end{matrix}}{\sigma;\Gamma\vdash\forall x_\tau\cdot F,\Delta}\,(\forall R) \qquad \dfrac{\begin{matrix}\vdots\ \pi_2'\\ \sigma;\Gamma,\forall x_\tau\cdot F,F[x:=t]\vdash\Delta\end{matrix}}{\sigma;\Gamma,\forall x_\tau\cdot F\vdash\Delta}\,(\forall L)}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

into:

$$\dfrac{\begin{array}{c}\vdots\,\pi_1'[y:=t]\\ \sigma;\Gamma\vdash F[x:=t],\Delta\end{array}\quad\dfrac{\begin{array}{cc}\vdots\,\pi_1 & \vdots\,\pi_2'\\ \sigma;\Gamma\vdash\forall x_\tau\cdot F,\Delta & \sigma;\Gamma,\forall x_\tau\cdot F,F[x:=t]\vdash\Delta\end{array}}{\sigma;\Gamma,F[x:=t]\vdash\Delta}\,(*)}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

which we now explain and justify. Notice that we may assume without loss of generality that $[y := t]$ is an idempotent substitution. Otherwise, first replace y by a fresh variable z_τ throughout π_1': we can do this, by Lemma 3.22, noticing that $[y := z]$ is idempotent. So assume that $[y := t]$ is an idempotent substitution: then by Lemma 3.22 there is a proof of $mgu(\overline{\sigma}[y := t]); \Gamma[y := t] \vdash F[x := y][y := t], \Delta[y := t]$, that is, since y is not free in $\overline{\sigma}, \Gamma, \forall x_\tau \cdot F, \Delta$, of $mgu(\overline{\sigma}); \Gamma \vdash F[x := t], \Delta$. Since $mgu(\overline{\sigma}) \equiv \sigma$, by Lemma 3.18, there is also a proof of $\sigma; \Gamma \vdash F[x := t], \Delta$: this is the proof named $\pi_1'[y := t]$ above. Notice that this proof has degree $< d$ and height at most $n - 1$. On the other hand, step $(*)$ is obtained by induction hypothesis after weakening on π_1. The induction hypothesis applies since $h(\pi_1) + h(\pi_2') = h(\pi_1) + h(\pi_2) - 1 < h(\pi_1) + h(\pi_2)$. Finally, the last instance of (Cut) has degree $\partial(F[x := t]) = \partial(F) < d$.

- The argument is similar when $R_1 = (\exists R)$ and $R_2 = (\exists L)$.

Finally, it is clear that the processes described above are all effective. □

Theorem 3.25 (Cut Elimination) *Every proof of $\sigma; \Gamma \vdash \Delta$ in* LKc$_\approx$ *can be effectively transformed into one of the same sequent that does not use (Cut).*

Proof. We first claim that: every proof of $\sigma; \Gamma \vdash \Delta$ of degree $d \geq 1$ can be effectively transformed into one of degree at most $d - 1$. This is by structural induction on the proof: if it ends in a non-(Cut) rule, or a (Cut) rule of degree $< d$, then this is by induction hypothesis, otherwise this is by induction hypothesis and Lemma 3.24.

The Theorem follows from the claim, by induction on d. □

4 Proof Search

As is usual, we obtain a free-variable tableau method by turning sequents upside-down, looking for a cut-free proof in LKc$_\approx$. The search for terms t in the case of rules $(\forall L)$ and $(\exists R)$ is done by representing the unknown value of t by a so-called free variable X. (We shall instead call such variables *existential variables*, to avoid the risk of confusion with the notion of free variables of terms. By contrast, the variables in $\mathcal{V}$ that we have considered until then will be called *universal*.) The application of some of the rules will then impose some unification constraints on terms built on free variables, thus determining the possible values of t. Traditionally, the only rule that generates unification constraints is the axiom:

$$\dfrac{}{;\Gamma, F \vdash F, \Delta}\,(Ax)$$

where some formula on the left of the sequent has to be unified with some formula on the right, yielding the common formula F.

In LKc$_\approx$, the corresponding rule is $(\approx)$. Observe that the unification problem is more complicated, as $(\approx)$ itself is more complex than (Ax). Moreover, $(\bot L)$, $(\approx R)$ and $(\approx\uparrow)$ will also impose some unification constraints. To deal with this situation, we separate the problem of generating the constraints from that of solving them.

4.1 Constrained Tableaux

First, we make the following observation:

Lemma 4.1 *Consider the following restrictions of* $(\forall L)$ *and* $(\exists R)$*:*

$$\frac{\sigma;\Gamma,\forall x_\tau\cdot F, F[x:=t]\vdash\Delta}{\sigma;\Gamma,\forall x_\tau\cdot F\vdash\Delta}\,(\forall L^-)\qquad \frac{\sigma;\Gamma\vdash F[x:=t],\exists x_\tau\cdot F,\Delta}{\sigma;\Gamma\vdash\exists x_\tau\cdot F,\Delta}\,(\exists R^-)$$

where $\rhd t:\tau$ *and* $\mathrm{fv}(t)\subseteq\mathrm{fv}(\overline{\sigma},\Gamma,\forall x\cdot F,\Delta)$.

Let $\mathrm{LKc}^-_\approx$ *denote the restriction of* $\mathrm{LKc}_\approx$ *where the only allowed instances of* $(\forall L)$*, resp.* $(\exists R)$*, are instances of* $(\forall L^-)$*, resp.* $(\exists R^-)$*. Then, any sequent that has a cut-free proof in* $\mathrm{LKc}_\approx$ *has a cut-free proof in* $\mathrm{LKc}^-_\approx$*.*

Proof. By induction on the height of the given cut-free proof in $\mathrm{LKc}_\approx$. This is clear if the last rule is anything except $(\forall L)$ or $(\exists R)$. In case the proof ends in an instance of $(\exists R)$, it is of the form:

$$\frac{\begin{array}{c}\vdots\,\pi_1\\ \sigma;\Gamma\vdash F[x:=t],\exists x\cdot F,\Delta\end{array}}{\sigma;\Gamma\vdash\exists x\cdot F,\Delta}\,(\exists R)$$

where π_1 is cut-free. Let $y_{1_{\tau_1}},\ldots,y_{k_{\tau_k}}$ be the free variables of t that are not free in $\overline{\sigma},\Gamma,\exists x\cdot F,\Delta$. Recall that there exists a ground term of type τ, for each sort τ (Definition 1.1): for each i, $1\le i\le k$, let t_i be some ground term of type τ_i. The substitution $\theta =_{\mathrm{df}} [y_1:=t_1,\ldots,y_k:=t_k]$ is ground, hence idempotent. By Lemma 3.22 with $d=0$ and $n=h(\pi_1)$, there is a proof of degree at most 0 (hence a cut-free proof) and of height at most n of $mgu(\overline{\sigma}\theta);\Gamma\theta\vdash F[x:=t]\theta,(\exists x\cdot F)\theta,\Delta\theta$. Since $\mathrm{dom}\,\theta$ does not intersect $\mathrm{fv}(\overline{\sigma},\Gamma,\forall x\cdot F,\Delta)$, the latter sequent is exactly $mgu(\overline{\sigma});\Gamma\vdash F[x:=t\theta],\exists x\cdot F,\Delta$. Since $mgu(\overline{\sigma})\equiv\sigma$, by Lemma 3.18, there is also a proof of degree at most 0 (a cut-free proof, again) and height at most $h(\pi_1)$ of $\sigma;\Gamma\vdash F[x:=t\theta],\exists x\cdot F,\Delta$. By induction hypothesis, we can derive the latter by some cut-free $\mathrm{LKc}^-_\approx$ proof π'_1. By construction, the free variables of $t\theta$ are all among $\mathrm{fv}(\overline{\sigma},\Gamma,\forall x\cdot F,\Delta)$, so we can derive:

$$\frac{\begin{array}{c}\vdots\,\pi'_1\\ \sigma;\Gamma\vdash F[x:=t\theta],\exists x\cdot F,\Delta\end{array}}{\sigma;\Gamma\vdash\exists x\cdot F,\Delta}\,(\exists R^-)$$

The case of $(\forall L)$ is entirely similar. □

The *raison d'être* of this Lemma is the following. When we solve unification constraints (see Section 4.2), it will be apparent that we shall benefit from knowing the fact that an existential variable X should denote a term t whose set of free (universal) variables we already know, and is finite. Lemma 4.1 gives us a way of estimating the set of free universal variables in the unknown term t.

Each existential variable X will then be equipped with a given set of universal variables $\{x_1,\ldots,x_n\}$ that it may depend on. We shall sometimes represent this dependency explicitly by writing X as $X^{x_1,\ldots,x_n}$.

This looks like skolemization, or its dual, herbrandization [GLM97]. However, herbrandization (the one of the latter mechanisms that is sound for provability) is a modification of rules $(\forall R)$ and $(\exists L)$, not $(\forall L)$ and $(\exists R)$. E.g., herbrandization allows us, when looking for a proof of $\Gamma\vdash\forall x\cdot F,\Delta$, to search for one of $\Gamma\vdash F[x:=f(y_1,\ldots,y_n)],\Delta$, where f is some fresh function symbol, and y_1, …, y_n are the free variables in $\forall x\cdot F$. (This variant is also known as the δ^+ rule, and can be justified without any recourse to semantics [GLM97]; there are other variants, see [BHS93].)

In $\mathrm{LKc}_\approx$, it is not sound to herbrandize. Indeed, introducing fresh function symbols in our framework means introducing new constructors, which is far removed from the intent of herbrandization. Concretely, herbrandization allows us to derive $\forall x_{\mathtt{nat}}\cdot\neg x\approx\mathtt{0}$ for example, from $\neg f()\approx\mathtt{0}$, where the latter is provable by $(\approx L)$, because the system erroneously considers f as a constructor; but this is absurd.

The standard solution to this conundrum is to manage dependencies among free variables (see [Koh95] for example). However, we have not found a way to solve unification constraints in the presence of dependencies.

Another solution would be to do some *predicative herbrandization*: whereas we cannot introduce fresh function symbols, we can introduce fresh predicate symbols. Predicative herbrandization then means deriving $\Gamma \vdash \forall x \cdot F, \Delta$ from some instance of $\Gamma \vdash \exists x \cdot P(y_1, \ldots, y_n, x) \wedge F, \Delta$, where P is a fresh predicate with functionality $\{1, \ldots, n\}$. In this form, predicative herbrandization is incomplete: we cannot even derive $\forall x \cdot A \supset A$, because we cannot prove that $\exists x \cdot P(y_1, \ldots, y_n, x)$ holds. Adding the latter to the left-hand side Γ of the sequent is a dead end: to use it, we have to resort to predicative herbrandization again, and nothing ever comes out of it.

Definition 4.2 (Constraints) *Let $\mathcal{X}$ be a set of so-called* existential variables $X_\tau^{x_1 \tau_1, \ldots, x_n \tau_n}$, *or X for short, where $x_1, \ldots, x_n \in \mathcal{V}$ are the universal variables which X* depends on. *We assume that, for every list $x_1, \ldots, x_n$ of universal variables, for every sort τ, there are infinitely many existential variables $X_\tau^{x_1 \tau_1, \ldots, x_n \tau_n}$, that any two existential variables with the same name X have the same list of variables that they depend on, and that $\mathcal{X}$ and $\mathcal{V}$ are disjoint. We drop the τ subscript from X when τ is clear from the context.*

The extended pre-terms *are given by the following grammar:*

$$t ::= x_\tau \mid c(t, \ldots, t) \mid X_\tau^{x, \ldots, x}$$

where x ranges over $\mathcal{V}$, $X(x, \ldots, x)$ over $\mathcal{X}$ and c over $\mathcal{C}$. The extended terms *are those that are typable, where the $\triangleright$ relation is extended by:*

$$\frac{}{\triangleright X_\tau^{x_1 \tau_1, \ldots, x_n \tau_n} : \tau}$$

We extend the notions of equations, systems, formulas, and so on, to use extended terms instead of terms.

A constraint *is an expression of the form $E_1 \overset{?}{\vdash} E_2$, where E_1 and E_2 are extended systems of equations or the symbol $\bot$. A* constraint set *K is a finite multiset of constraints.*

A substitution ϑ is admissible *if and only if $\operatorname{dom} \vartheta \subseteq \mathcal{X}$ and $\operatorname{fv}(X^{x_1, \ldots, x_n} \vartheta) \subseteq \{x_1, \ldots, x_n\}$ for every $X^{x_1, \ldots, x_n} \in \operatorname{dom} \vartheta$. We say that ϑ satisfies $E_1 \overset{?}{\vdash} E_2$, written $\vartheta \models E_1 \vdash E_2$, if and only if $E_1 \vartheta \vdash E_2 \vartheta$, and that $\vartheta \models K$ if and only if $\vartheta \models E_1 \vdash E_2$ for every $E_1 \overset{?}{\vdash} E_2$ in K.*

Note that every admissible substitution ϑ is idempotent by definition, and that no universal variable x is in its domain. It follows that $s\vartheta$ is an extended term whenever s is, and that in general all definitions above make sense.

α	α_1	α_2
$+(F \wedge G)$	$+F$	$+G$
$-(F \vee G)$	$-F$	$-G$
$-(F \supset G)$	$+F$	$-G$

β	β_1	β_2
$-(F \wedge G)$	$-F$	$-G$
$+(F \vee G)$	$+F$	$+G$
$+(F \supset G)$	$-F$	$+G$

γ_τ	$\gamma_0(t)$
$+\forall x_\tau \cdot F$	$+F[x := t]$
$-\exists x_\tau \cdot F$	$-F[x := t]$

δ_τ	$\delta_0(y)$
$-\forall x_\tau \cdot F$	$-F[x := y]$
$+\exists x_\tau \cdot F$	$+F[x := y]$

Figure 4: Smullyan-type classification

We classify formulas in the four categories α, β, γ and δ, as usual in tableaux: see Figure 4. An *open tableau branch* is a 4-tuple $\vec{x}\ ; K\ ; E\ ; S$, where $\vec{x}$ is a list of universal variables (in $\mathcal{X}$), K is a constraint set, E is an extended system of equations, and S is a finite multiset of signed formulas. A *signed formula* $\pm F$ is either a *positive formula* $+F$ or a *negative formula* $-F$. The list $\vec{x}$ represents all the universal variables in the current scope, K is the set of constraints that have been accumulated on this branch until now, E corresponds to the σ component of sequents, and S to the $\Gamma \vdash \Delta$ component: formulas F of Γ appear as $+F$ in S, formulas G of Δ appear as $-G$.

A *closed tableau branch* is just a constraint set K. The rules of Figure 5 derive sets of tableau branches (in conclusion) from open tableau branches (in premise). The notation $\vec{x} \cdot y$ denotes the list $\vec{x}$ with y added at the end; the notation $X_\tau^{\vec{x}}$ denotes $X_\tau^{x_1,\ldots,x_n}$, assuming that $\vec{x} = x_1 \cdot \ldots \cdot x_n$. The rules $(+\bot)$, (Cl), $(Refl)$ and $(+\mathbf{0})$ produce closed branches, and are therefore called *closing rules.*

Definition 4.3 (Tableaux) *A* tableau *T is a multiset $B_1, \ldots, B_k$ of tableau branches. T is* closed *if and only if B_1, …, B_k are all closed branches, i.e., constraint sets. The constraint set $K(T)$ is the union of all its closed branches.*

Tableau expansion *is the rewrite rule $\Longrightarrow$ on tableaux defined by $T, B \Longrightarrow T, B_1, \ldots, B_k$, for every tableau rule:*

$$\frac{B}{B_1 \mid \ \ldots \ \mid B_k}$$

Closed tableaux are therefore just the $\Longrightarrow$-normal forms. A $\Longrightarrow$-normal form of some tableau of T is called a *closed tableau for T*.

$$\frac{\vec{x}\ ; K\ ; E\ ; S}{K, (E \Vdash^? \bot)}\,(+\bot) \qquad \frac{\vec{x}\ ; K\ ; E\ ; S, +P(s_1, \ldots, s_n), -P(t_1, \ldots, t_n)}{K, (E \Vdash^? s_1 \approx t_1, \ldots, s_n \approx t_n)}\,(Cl)$$

$$\frac{\vec{x}\ ; K\ ; E\ ; S, +s \approx t}{\vec{x}\ ; K\ ; E, s \approx t\ ; S}\,(EqL) \qquad \frac{\vec{x}\ ; K\ ; E\ ; S, -s \approx t}{K, (E \Vdash^? s \approx t)}\,(Refl)$$

$$\frac{\vec{x}\ ; K\ ; E\ ; S, +P(s_1, \ldots, s_n), +P(t_1, \ldots, t_n)}{\vec{x}\ ; K, (\Vdash^? s_i \approx t_i)_{i \in f}\ ; E, \vec{s}_{\overline{f}^n} \approx \vec{t}_{\overline{f}^n}\ ; S, +P(s_1, \ldots, s_n), +P(t_1, \ldots, t_n)}\,(Func)$$
$$\text{(provided } f \in \phi(P)\text{, and some } s_j \approx t_j \text{ is not in } E,\ j \in \overline{f}^n\text{)}$$

$$\frac{\vec{x}\ ; K\ ; E\ ; S, +\mathbf{0}}{K}\,(+\mathbf{0})$$

$$\frac{\vec{x}\ ; K\ ; E\ ; S, \alpha}{\vec{x}\ ; K\ ; E\ ; S, \alpha_1, \alpha_2}\,(\alpha) \qquad \frac{\vec{x}\ ; K\ ; E\ ; S, \beta}{\vec{x}\ ; K\ ; E\ ; S, \beta_1 \mid \vec{x}\ ; K\ ; E\ ; S, \beta_2}\,(\beta)$$

$$\frac{\vec{x}\ ; K\ ; E\ ; S, \gamma_\tau}{\vec{x}\ ; K\ ; E\ ; S, \gamma_\tau, \gamma_0(X_\tau^{\vec{x}})}\,(\gamma) \quad (X_\tau^{\vec{x}} \in \mathcal{X} \text{ fresh}) \qquad \frac{\vec{x}\ ; K\ ; E\ ; S, \delta_\tau}{\vec{x} \cdot y_\tau\ ; K\ ; E\ ; S, \delta_0(y_\tau)}\,(\delta) \quad (y_\tau \in \mathcal{V} \text{ fresh})$$

Figure 5: Constrained Tableaux

As a side note, we have not defined "fresh" in the γ and δ rules. This can be made formal as follows: with each open branch, associate an infinite set SX of existential variables, and an infinite set SY of universal variables. The γ and δ rules then become:

$$\frac{\vec{x}\ ; K\ ; E\ ; S, \delta_\tau\ ; SX\ ; SY}{\vec{x} \cdot y_\tau\ ; K\ ; E\ ; S, \delta_0(y_\tau)\ ; SX\ ; SY \setminus \{y_\tau\}}\,(\delta)\ (y_\tau \in SY)$$

$$\frac{\vec{x}\ ; K\ ; E\ ; S, \gamma_\tau\ ; SX\ ; SY}{\vec{x}\ ; K\ ; E\ ; S, \gamma_\tau, \gamma_0(X_\tau^{\vec{x}})\ ; SX \setminus \{X_\tau^{\vec{x}}\}\ ; SY}\,(\gamma)\ (X_\tau^{\vec{x}} \in SX)$$

and the (β) rules become:

$$\frac{\vec{x}\ ; K\ ; E\ ; S, \beta\ ; SX\ ; SY}{\vec{x}\ ; K\ ; E\ ; S, \beta_1\ ; SX_1\ ; SY_1 \mid \vec{x}\ ; K\ ; E\ ; S, \beta_2\ ; SX_2\ ; SY_2}\,(\beta)$$

where SX_1, SX_2 forms a partition of SX into two infinite subsets, and similarly for SY_1, SY_2 and SY. The SX and SY components remain unchanged from premise to conclusion in the α rules, in $(Func)$ and in (EqL). Although this is what we understand by "fresh", we won't use this formalization, as it is heavy and does not help our understanding much.

Given an admissible substitution ϑ, an extended system of equations E, and a finite multiset $S =_{\mathrm{df}} +F_1, \ldots, +F_m, -G_1, \ldots, -G_n$ of signed formulas, let $(S)^{\vdash}\vartheta$ denote $F_1\vartheta, \ldots, F_m\vartheta \vdash G_1\vartheta, \ldots, G_n\vartheta$, and $(E; S)^{\vdash}\vartheta$ be the sequent $mgu(E\vartheta); (S)^{\vdash}\vartheta$, that is, $mgu(E\vartheta); F_1\vartheta, \ldots, F_m\vartheta \vdash G_1\vartheta, \ldots, G_n\vartheta$. Notice that this is an actual sequent, provided that every existential variable $X_\tau^{x_1,\ldots,x_n}$ free in E, S is in $\operatorname{dom}\vartheta$.

Lemma 4.4 (Soundness) *Let T be a closed tableau for $;;; B$. For every admissible substitution ϑ such that $\vartheta \models K(T)$ and $\operatorname{dom}\vartheta \supseteq \operatorname{fv}(K(T)) \cap \mathcal{X}$, $;(B)^{\vdash}\vartheta$ is provable in* $\mathrm{LKc}_{\approx}$.

Proof. Define $\vartheta \models B$, when B is a tableau branch, as: $\vartheta \models K$ when B is a closed branch (a constraint set) K, otherwise as $\vartheta \models K$ and $(E; S)^{\vdash}\vartheta$ is provable in $\mathrm{LKc}_{\approx}$, when B is the open branch $\vec{x}\,; K\,; E\,; S$. Define $\vartheta \models T$, where T is a tableau, by $\vartheta \models B$ for every B in T.

We show the more general claim that: if $T_1 \Longrightarrow^* T_2$, $\operatorname{dom}\vartheta \supseteq \operatorname{fv}(T_2) \cap \mathcal{X}$ and $\vartheta \models T_2$, then $\operatorname{dom}\vartheta \supseteq \operatorname{fv}(T_1) \cap \mathcal{X}$ and $\vartheta \models T_1$. It is enough to prove this when $T_1 \Longrightarrow T_2$, and the claim will follow by induction on reduction length. Let therefore T_1 be T_0, B, and T_2 be $T_0, B_1, \ldots, B_k$, where:

$$\frac{B}{B_1 \mid \quad \ldots \quad \mid B_k}$$

is some tableau rule R. By assumption, $\operatorname{dom}\vartheta \supseteq \operatorname{fv}(B_i) \cap \mathcal{X}$ and $\vartheta \models B_i$ for every i, $1 \leq i \leq k$. In any case, B is an open branch $\vec{x}\,; K\,; E\,; S$, but the B_i's may be closed or open. The fact that $\operatorname{dom}\vartheta \supseteq \operatorname{fv}(T_1)$ is by an easy case analysis on the rule R. To show that $\vartheta \models T_1$, we examine each tableau rule in turn:

- $(+\bot)$: $k = 1$, $B_1 = K, (E \mathrel{\boldsymbol{\vdash}^{?}} \bot)$. By assumption, $\vartheta \models B_1$, so $E\vartheta$ is not unifiable. So $(E; B)^{\vdash}\vartheta$ is an instance of the $(\bot L)$ rule of $\mathrm{LKc}_{\approx}$. Moreover, $\vartheta \models K$, and $\vartheta \models T_0$, so $\vartheta \models T_1$.

- (Cl): $k = 1$, $B_1 = K, (E \mathrel{\boldsymbol{\vdash}^{?}} s_1 \approx t_1, \ldots, s_n \approx t_n)$. Since $\vartheta \models B_1$, it follows that $E\vartheta \mathrel{\boldsymbol{\vdash}} s_1\vartheta \approx t_1\vartheta, \ldots, s_n\vartheta \approx t_n\vartheta$, so either $E\vartheta$ is not unifiable, and $(E; B)^{\vdash}\vartheta$ is an instance of the $(\bot L)$ rule of $\mathrm{LKc}_{\approx}$, or it is unifiable, and $(E; B)^{\vdash}\vartheta$ is an instance of the $(\approx)$ rule of $\mathrm{LKc}_{\approx}$. Moreover, in any case $\vartheta \models K$, and $\vartheta \models T_0$, so $\vartheta \models T_1$. The $(Refl)$ case is similar, and maps to $(\approx R)$.

- (EqL): $B = \vec{x}\,; K\,; E\,; S, +s \approx t$, $k = 1$, $B_1 = \vec{x}\,; K\,; E, s \approx t\,; S$. Let $(S)^{\vdash}\vartheta$ be of the form $\Gamma \vdash \Delta$. Since $\vartheta \models B_1$, $mgu(E\vartheta, s\vartheta \approx t\vartheta); \Gamma \vdash \Delta$ has a proof in $\mathrm{LKc}_{\approx}$. By rule $(\approx L)$ of $\mathrm{LKc}_{\approx}$, we get a proof of $mgu(E\vartheta); \Gamma, s\vartheta \approx t\vartheta \vdash \Delta$, that is, of $(E; S, +s \approx t)^{\vdash}\vartheta$. On the other hand, $\vartheta \models K$, and $\vartheta \models T_0$, so $\vartheta \models T_1$. The arguments are similar for $(Func)$, and for the α and β-rules.

- $(+\mathbf{0})$: $B = \vec{x}\,; K\,; E\,; S, +\mathbf{0}$, $k = 1$, $B_1 = K$. Since $\vartheta \models B_1$, $\vartheta \models K$; on the other hand, $(E; S, +\mathbf{0})^{\vdash}\vartheta = mgu(E\vartheta); (S, +\mathbf{0})^{\vdash}\vartheta$ is provable by the $(\mathbf{0}L)$ rule of $\mathrm{LKc}_{\approx}$.

- (γ): $B = \vec{x}\,; K\,; E\,; S, \gamma_\tau$, $k = 1$, $B_1 = \vec{x}\,; K\,; E\,; S, \gamma_\tau, \gamma_0(X_\tau^{\vec{x}})$, where X_τ is a fresh existential variable name. Let $(S)^{\vdash}\vartheta$ be $\Gamma \vdash \Delta$, and assume that the γ_τ formula is $+\forall x_\tau \cdot F$; the $-\exists x_\tau \cdot F$ case is similar. Since $\vartheta \models B_1$, $mgu(E\vartheta); \Gamma, \forall x_\tau \cdot F\vartheta, F[x := X_\tau^{\vec{x}}]\vartheta \vdash \Delta$ has a proof in $\mathrm{LKc}_{\approx}$.

  We now have two cases. Case 1: x is free in F. In particular, $X_\tau^{\vec{x}}$ is free in B_1. By assumption, $\operatorname{dom}\vartheta \supseteq \operatorname{fv}(B_1) \cap \mathcal{X}$, so $X_\tau^{\vec{x}}$ is in $\operatorname{dom}\vartheta$; since ϑ is an admissible substitution, $X_\tau^{\vec{x}}\vartheta$ is a term t (not an extended term), so $F[x := X_\tau^{\vec{x}}]\vartheta = F\vartheta[x := t]$, hence we have a proof of $mgu(E\vartheta); \Gamma, \forall x_\tau \cdot F\vartheta, F\vartheta[x := t] \vdash \Delta$.

  Case 2: x is not free in F. Then choose any ground term t of type τ, so that $mgu(E\vartheta); \Gamma, \forall x_\tau \cdot F\vartheta, F\vartheta[x := t] \vdash \Delta$ has a proof in $\mathrm{LKc}_{\approx}$ again (since $F\vartheta[x := t] = F\vartheta = F[x := X_\tau^{\vec{x}}]\vartheta$).

In both cases, by the $(\forall L)$ rule of $\mathrm{LKc}_{\approx}$, we get a proof of $mgu(E\vartheta); \Gamma, \forall x_\tau \cdot F\vartheta \vdash \Delta$, that is, of $(E; S, \gamma_\tau)^{\vdash}\vartheta$.

- (δ): $B = \vec{x}\,; K\,; E\,; S, \delta_\tau$, $k = 1$, $B_1 = \vec{x} \cdot y_\tau\,; K\,; E\,; S, \delta_0(y_\tau)$, where y_τ is a fresh universal variable. Let $(S)^{\vdash}\vartheta$ be $\Gamma \vdash \Delta$, and assume that the δ_τ formula is $-\forall x_\tau \cdot F$; the $+\exists x_\tau \cdot F$ case is similar. Since $\vartheta \models B_1$, $mgu(E\vartheta); \Gamma \vdash F[x := y_\tau]\vartheta, \Delta$ has a proof. Since $y \in \mathcal{V}$ and ϑ is admissible, y is not in $\mathrm{dom}\,\vartheta$. Moreover, by α-renaming, we may assume that x is not free in $X^{\vec{x}}\vartheta$, so $F[x := y_\tau]\vartheta = F\vartheta[x := y_\tau]$. As y_τ is fresh, we may apply rule $(\forall R)$ of $\mathrm{LKc}_{\approx}$ and infer $\vartheta \models B_1$, $mgu(E\vartheta); \Gamma \vdash \forall x_\tau \cdot \vartheta, \Delta$ in $\mathrm{LKc}_{\approx}$, that is, $(E\,; S, \delta_\tau)^{\vdash}\vartheta$.

□

For every sequent $\sigma; \Gamma \vdash \Delta$, let $+\Gamma$ denote the multiset of all formulas $+F$, $F \in \Gamma$, $-\Delta$ denote the multiset of all formulas $-G$, $G \in \Delta$. If $\sigma = \bot$, let $\overline{\sigma}$ be $\bot$ as well.

Lemma 4.5 (Completeness) *If $\sigma; \Gamma \vdash \Delta$ is provable in $\mathrm{LKc}_{\approx}$, then there is a closed tableau T such that $\vec{x}_0\,;\ ; \overline{\sigma}\,; +\Gamma, -\Delta \Longrightarrow^* T$ and an admissible substitution ϑ such that $\vartheta \models K(T)$, where $\vec{x}_0$ contains all the free variables in $\overline{\sigma}, \Gamma, \Delta$.*

Proof. Recall that $\mathrm{yld}\,\vartheta = \bigcup_{x \in \mathrm{dom}\,\vartheta} \mathrm{fv}(x\vartheta)$. Say that a branch is *well-formed* if and only if it is closed, or it is an open branch $\vec{x}\,; K\,; E\,; S$ and all the free variables in E, S are in the list $\vec{x}$ (by abuse, we shall write $\mathrm{fv}(E, S) \subseteq \{\vec{x}\}$). Say that a tableau is *well-formed* if and only if all its branches are well-formed. Notice that if T_1 is well-formed and $T_1 \Longrightarrow T_2$, then T_2 is well-formed.

We claim that: $(*)$ if ϑ is an admissible substitution, $\vec{x}\,; K\,; E\,; S$ is well-formed, $\vartheta \models K$, $mgu(E\vartheta); (S)^{\vdash}\vartheta$ has a cut-free $\mathrm{LKc}_{\approx}^{-}$ proof π, and $\mathrm{yld}\,\vartheta \subseteq \{\vec{x}\}$, then $\vec{x}\,; K\,; E\,; S \Longrightarrow^* T$ for some closed tableau T, and $\vartheta \cup \vartheta' \models T$ for some admissible substitution ϑ' such that $\mathrm{dom}\,\vartheta \cap \mathrm{dom}\,\vartheta' = \emptyset$. This is by induction on the height of π. We examine each rule of $\mathrm{LKc}_{\approx}^{-}$ in turn:

- $(\bot L)$: so $E\vartheta$ is not unifiable: take $\vartheta' =_{\mathrm{df}} []$, and applying the tableau rule $(+\bot)$, we build the closed tableau $K, (E \Vdash^{?} \bot)$. Since $\vartheta \models K$ and $E\vartheta$ is not unifiable, $\vartheta \models K, (E \Vdash^{?} \bot)$.

- $(\approx)$: $E\vartheta$ is unifiable, and letting σ be its mgu, $(S)^{\vdash}\vartheta$ is of the form $\Gamma, A \vdash B, \Delta$, where $A\sigma = B\sigma$. So S can be written $S', +A', -B'$, where $(S')^{\vdash}\vartheta$ is $\Gamma \vdash \Delta$, and $A'\vartheta = A$ and $B'\vartheta = B$. In particular, $A'\vartheta\sigma = B'\vartheta\sigma$. A' and B' must then have the same topmost predicate symbols, so write A' as $P(s_1, \ldots, s_n)$, and B' as $P(t_1, \ldots, t_n)$ for some terms s_1, $\ldots$, s_n, t_1, $\ldots$, t_n such that $s_i\vartheta\sigma = t_i\vartheta\sigma$ for every i, $1 \le i \le n$. Since $\sigma = mgu(E\vartheta)$, it follows: (i) $E\vartheta \Vdash s_1\vartheta \approx t_1\vartheta, \ldots, s_n\vartheta \approx t_n\vartheta$. Apply the tableau rule (Cl): we get the closed tableau $K, (E \Vdash^{?} s_1 \approx t_1, \ldots, s_n \approx t_n)$. By (i) and since $\vartheta \models K$ by assumption, $(*)$ is proved with $\vartheta' =_{\mathrm{df}} []$. The argument is similar with $(\approx R)$.

- $(\approx L)$: $E\vartheta$ is unifiable, $(S)^{\vdash}\vartheta$ is of the form $\Gamma, s \approx t \vdash \Delta$, and the sequent $mgu(E\vartheta); (S)^{\vdash}\vartheta$ is derived from a shorter proof π' of $mgu(E\vartheta, s \approx t); \Gamma \vdash \Delta$. So S has the form $S', +s' \approx t'$, with $(S')^{\vdash}\vartheta = \Gamma \vdash \Delta$, $s'\vartheta = s$ and $t'\vartheta = t$. Apply the tableau rule (EqL), getting $\vec{x}\,; K\,; E, s' \approx t'\,; S'$. By assumption, $\vartheta \models K$ and π' is a proof of $mgu((E, s' \approx t')\vartheta); (S')^{\vdash}\vartheta$, moreover $\mathrm{yld}\,\vartheta \subseteq \{\vec{x}\}$. Since π' is shorter than π, we apply the induction hypothesis: $(*)$ follows.

- $(\approx\uparrow)$: $E\vartheta$ is unifiable, $(S)^{\vdash}\vartheta$ is of the form $\Gamma, P(s_1, \ldots, s_n), P(t_1, \ldots, t_n) \vdash \Delta$, and, letting σ be $mgu(E\vartheta)$, we have $\vec{s}_f\sigma = \vec{t}_f\sigma$. So S is of the form $S', P(s'_1, \ldots, s'_n), P(t'_1, \ldots, t'_n)$, with $(S')^{\vdash} = \Gamma \vdash \Delta$, and $s'_i\vartheta = s_i$, $t'_i\vartheta = t_i$ for every i, $1 \le i \le n$. The argument is essentially the same as in previous cases, noticing that in addition $\vartheta \models (\Vdash^{?} s'_i \approx t'_i)$ for every i, $i \in f$.

- $(\mathbf{0}L)$: trivial.

- $(\wedge L)$: easy recourse to the induction hypothesis, using the corresponding α-rule. Similar for $(\vee R)$, $(\supset R)$.

- $(\wedge R)$: S must be of the form $S', -(F \wedge G)$. Let $(S')^{\vdash}\vartheta$ be $\Gamma \vdash \Delta$, so that $(S)^{\vdash}\vartheta$ is $\Gamma \vdash F\vartheta \wedge G\vartheta, \Delta$. We have shorter proofs π_1 of $mgu(E\vartheta); \Gamma \vdash F\vartheta, \Delta$ and π_2 of $mgu(E\vartheta); \Gamma \vdash G\vartheta, \Delta$. By induction, there are admissible substitutions ϑ'_1 and ϑ'_2 such that $\mathrm{dom}\,\vartheta \cap \mathrm{dom}\,\vartheta'_1 = \emptyset$, $\mathrm{dom}\,\vartheta \cap \mathrm{dom}\,\vartheta'_2 = \emptyset$, $\vec{x}\,; K\,; E\,; S', -F \Longrightarrow^* T_1$ with $\vartheta'_1 \models K(T_1)$, and $\vec{x}\,; K\,; E\,; S', -G \Longrightarrow^* T_2$ with $\vartheta'_2 \models K(T_2)$. Now observe that we may assume without loss of generality that $\mathrm{dom}\,\vartheta \cup \mathrm{dom}\,\vartheta'_1 \subseteq \mathrm{fv}(T_1)$ and that $\mathrm{dom}\,\vartheta \cup \mathrm{dom}\,\vartheta'_2 \subseteq \mathrm{fv}(T_2)$. So $\mathrm{dom}\,\vartheta'_1$ contains only existential variables created during the rewriting from $\vec{x}\,; K\,; E\,; S', -F$ to T_1 by the γ rules, and similarly for $\mathrm{dom}\,\vartheta'_2$. (We let the reader write the full proof, using the sets SX and SY introduced above.) By the freshness condition on existential variables in the γ rules, $\mathrm{dom}\,\vartheta'_1 \cap \mathrm{dom}\,\vartheta'_2 = \emptyset$. In particular, the notation $\vartheta'_1 \cup \vartheta'_2$ makes sense and denotes an admissible substitution: call it ϑ'. Since $\mathrm{dom}\,\vartheta \cap \mathrm{dom}\,\vartheta'_1 = \emptyset$ and $\mathrm{dom}\,\vartheta \cap \mathrm{dom}\,\vartheta'_2 = \emptyset$, it follows that $\mathrm{dom}\,\vartheta \cap \mathrm{dom}\,\vartheta' = \emptyset$. Moreover, $\vartheta \cup \vartheta' \models T_1, T_2$. Claim $(*)$ then follows by using the β-rule of tableaux, $\vec{x}\,; K\,; E\,; S', -(F \wedge G) \Longrightarrow (\vec{x}\,; K\,; E\,; S', -F), (\vec{x}\,; K\,; E\,; S', -G) \Longrightarrow^* T_1, T_2$. The argument is similar for $(\vee L)$, $(\supset L)$.

- $(\forall L^-)$: S must be of the form $S', +\forall x_\tau \cdot F$, and letting $(S')^{\vdash}\vartheta$ be $\Gamma'\vartheta \vdash \Delta'\vartheta$, so that $(S)^{\vdash}\vartheta$ is $\Gamma'\vartheta, \forall x_\tau \cdot F\vartheta \vdash \Delta'\vartheta$, we have derived $mgu(E\vartheta); \Gamma'\vartheta, F\vartheta[x := t] \vdash \Delta'\vartheta$ by a shorter proof π'. (Notice that this is the case provided $x \notin \mathrm{dom}\,\vartheta$, which we can always ensure by a suitable α-renaming.) Alternatively, we have derived $mgu(E\vartheta); \Gamma'\vartheta, F(\vartheta \cup [x := t]) \vdash \Delta'\vartheta$ by π'. Since X_τ is fresh, moreover, $X_\tau^{\vec{x}}$ is not free in E, so $E\vartheta = E(\vartheta \cup \{X_\tau^{\vec{x}} := t\})$; similarly, $\Gamma'\vartheta = \Gamma'(\vartheta \cup \{X_\tau^{\vec{x}} := t\})$ $\Delta'\vartheta = \Delta'(\vartheta \cup \{X_\tau^{\vec{x}} := t\})$, so: (ii) π' is an $\mathrm{LKc}^-_{\approx}$ proof of $mgu(E(\vartheta \cup \{X_\tau^{\vec{x}} := t\})); \Gamma'(\vartheta \cup \{X_\tau^{\vec{x}} := t\}), F[x := X_\tau^{\vec{x}}](\vartheta \cup \{X_\tau^{\vec{x}} := t\}) \vdash \Delta'(\vartheta \cup \{X_\tau^{\vec{x}} := t\})$.

  Furthermore, because we have used rule $(\forall L^-)$, we also have $\mathrm{fv}(t) \subseteq \mathrm{fv}(E\vartheta, \Gamma'\vartheta, \forall x \cdot F\vartheta, \Delta'\vartheta)$. By assumption, $\mathrm{yld}\,\vartheta \subseteq \{\vec{x}\}$, so $\mathrm{fv}(t) \subseteq \mathrm{fv}(E, \Gamma', \forall x \cdot F, \Delta') \cup \{\vec{x}\} = \mathrm{fv}(E, S) \cup \{\vec{x}\}$. Since $\vec{x}\,; K\,; E\,; S$ is well-formed, $\mathrm{fv}(E, S) \subseteq \{\vec{x}\}$, so: (iii) $\mathrm{fv}(t) \subseteq \{\vec{x}\}$. So $\vartheta \cup \{X_\tau^{\vec{x}} := t\}$ is an admissible substitution. Clearly, $\vec{x}\,; K\,; E\,; S', +\forall x_\tau \cdot F, F[x := X_\tau^{\vec{x}}]$ is well-formed, $\vartheta \cup \{X_\tau^{\vec{x}} := t\} \models K$ (since $\vartheta \models K$). Using (ii) above, and since $\mathrm{yld}(\vartheta \cup \{X_\tau^{\vec{x}} := t\}) \subseteq \mathrm{yld}\,\vartheta \cup \mathrm{fv}(t) \subseteq \{\vec{x}\}$ (since $\mathrm{yld}\,\vartheta \subseteq \{\vec{x}\}$ by assumption, and by (iii)), the induction hypothesis applies, so that $\vec{x}\,; K\,; E\,; S', +\forall x_\tau \cdot F, F[x := X_\tau^{\vec{x}}] \Longrightarrow^* T$ for some closed tableau and some admissible substitution ϑ'' such that $(\vartheta \cup \{X_\tau^{\vec{x}} := t\}) \cup \vartheta'' \models T$ and $\mathrm{dom}(\vartheta \cup \{X_\tau^{\vec{x}} := t\}) \cap \mathrm{dom}\,\vartheta'' = \emptyset$. Taking $\vartheta' =_{\mathrm{df}} \{X_\tau^{\vec{x}} := t\} \cup \vartheta''$ then proves $(*)$.

- $(\forall R)$: S must be of the form $S', -\forall x_\tau \cdot F$. Let $(S')^{\vdash}\vartheta$ be $\Gamma \vdash \Delta$, so that $(S)^{\vdash}\vartheta$ is $\Gamma \vdash \forall x_\tau \cdot F\vartheta, \Delta$. We have a shorter proof π' of $mgu(E\vartheta); \Gamma \vdash F\vartheta[x := y_\tau], \Delta$, where y_τ is not free in the conclusion of the $(\forall R)$ rule. By Lemma 3.22, we may assume that y_τ is the fresh variable chosen by the corresponding δ rule. However, the δ rule derives $\vec{x}{\cdot}y_\tau\,; K\,; E\,; S', -F[x := y_\tau]$, and $(S', -F[x := y_\tau])^{\vdash}\vartheta$ is $\Gamma \vdash F[x := y_\tau]\vartheta, \Delta$, not $\Gamma \vdash F\vartheta[x := y_\tau], \Delta$. Fortunately, the latter two are the same, since y_τ is fresh and $\mathrm{dom}\,\vartheta \cap \mathcal{X} = \emptyset$. The induction hypothesis applies, proving the claim. The case is similar for $(\exists L)$.

The Lemma is then proved by applying $(*)$ with ϑ the empty substitution, $\vec{x} =_{\mathrm{df}} \vec{x}_0$, K the empty constraint set, $E =_{\mathrm{df}} \overline{\sigma}$, $S =_{\mathrm{df}} +\Gamma, -\Delta$. Indeed, $\vec{x}\,; K\,; E\,; S$ is well-formed by construction, and trivially $\vartheta \models K$. Moreover, $\sigma; \Gamma \vdash \Delta$ is provable in $\mathrm{LKc}_{\approx}$, so it has in fact a cut-free $\mathrm{LKc}^-_{\approx}$ proof, by Theorem 3.25 and Lemma 4.1. And $\mathrm{yld}\,\vartheta \subseteq \{\vec{x}\}$ (trivially), so $(*)$ applies: there must be a closed tableau T and an admissible substitution ϑ' such that $\vartheta' \models T$. $\square$

Say that a constraint set K is *satisfiable* if and only if there is an admissible substitution ϑ such that $\vartheta \models K$. Combining most of the results of this paper, we get:

Theorem 4.6 *Given any closed sequent* $;\Gamma \vdash \Delta$*, the following are equivalent:*

1. $I \models (;\Gamma \vdash \Delta)$ *for every free equational interpretation I that respects functionalities;*
2. $;\Gamma \vdash \Delta$ *has a proof in* $\mathrm{LKc}_{\approx}$*;*
3. $;\Gamma \vdash \Delta$ *has a proof in* $\mathrm{LKc}_{\approx}$ *without* (Cut)*;*

4. $;\Gamma \vdash \Delta$ *has a proof in* $\mathrm{LKc}^{-}_{\approx}$*;*

5. $;\Gamma \vdash \Delta$ *has a proof in* $\mathrm{LKc}^{-}_{\approx}$ *without* (Cut)*;*

6. $;\ ;\ ;+\Gamma, -\Delta$ *has a closed tableau* T *such that* $K(T)$ *is satisfiable.*

4.2 Solving Constraints

Unfortunately, we have the following result:

Theorem 4.7 (Undecidability) *The problem of determining whether a constraint set* K *given as input is satisfiable, is undecidable.*

Proof. By reduction from MPU, the monadic predicate unification problem [Ami90]. This problem is as follows. Let $Cons$ be a set of function symbols c, d, ..., given with their arities; the *terms* $s, t, \ldots$, are inductively defined as the first-order variables $X, Y, Z, \ldots$, and the applications $c(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and $c \in Cons$ is of arity n. The atomic *formulas* A, B, are expressions of the form $F(t)$, where t is a term and F is a second-order variable taken from a set $(F_i)_i \in I$, or the predicate constant P. *Substitutions* θ map first-order variables X to terms and second-order variables F to abstractions $\lambda w \cdot B$, where B is a formula. Given A, $A\theta$ is defined as: $(F(t))\theta =_{\mathrm{df}} B[w := t\theta]$ provided that $F \in \mathrm{dom}\,\theta$ and $\theta(F) = \lambda w \cdot B$, otherwise $(F(t))\theta =_{\mathrm{df}} F(t\theta)$ and $(P(t))\theta =_{\mathrm{df}} P(t\theta)$, where $t\theta$ is first-order substitution, defined as usual. A MPU problem S is a finite set of equations $A \approx B$ between formulas; θ solves S if and only if $A\theta = B\theta$ for every $A \approx B$ in S. This problem is shown to be undecidable as soon as $Cons$ contains a function symbol c of arity at least 2 in [Ami90].

Without loss of generality, we may assume that the equations $A \approx B$ that we consider are not of the form $F(t_1) \approx F(t_2)$, with the same second-order variable F on both sides; otherwise, replace it by the two equations $F(t_1) \approx G(a), G(a) \approx F(t_2)$, where G is a fresh second-order variable and a is any fixed constant. We may also assume that the language of terms contains at least one constant, and therefore look only for unifiers θ that are *ground* in the sense that for each term t or predicate $\lambda w \cdot A$ in the range of θ, $\mathrm{fv}(t) = \emptyset$, $\mathrm{fv}(A) \subseteq \{w\}$. Finally, we may assume that unifiers θ of S are *grounding* in that every free variable of S is in $\mathrm{dom}\,\theta$.

Now let $\mathtt{T}$ be a sort, and assume that there is a constant a of type $\mathtt{T}$, a constructor P of arity $\mathtt{T} \to \mathtt{T}$, and a constructor c of arity $\mathtt{T} \times \mathtt{T} \to \mathtt{T}$. Equate the first-order variables X of MPU with the existential variables $X^{\emptyset}_{\mathtt{T}}$, and the predicate symbol P with the constructor P; for every second-order variable F_i, $i \in I$, let x_i be a distinct universal variable of sort $\mathtt{T}$, not occurring in any MPU problem, and let $X_i^{x_i}$ be a distinct existential variable of sort $\mathtt{T}$. With each second-order equation $A \approx B$, associate a constraint $\underline{A \approx B}$ defined as follows:

A	B	$\underline{A \approx B}$
$P(t_1)$	$P(t_2)$	$\stackrel{?}{\vdash} P(t_1) \approx P(t_2)$
$F_{i_1}(t_1)$	$F_{i_2}(t_2)$	$x_{i_1} \approx t_1, x_{i_2} \approx t_2 \ \stackrel{?}{\vdash} X_{i_1} \approx X_{i_2}$
$F_{i_1}(t_1)$	$P(t_2)$	$x_{i_1} \approx t_1 \ \stackrel{?}{\vdash} X_{i_1} \approx P(t_2)$
$P(t_1)$	$F_{i_2}(t_2)$	$x_{i_2} \approx t_2 \ \stackrel{?}{\vdash} P(t_1) \approx X_{i_2}$

For each MPU problem S, let then $\underline{S}$ be the set of all constraints $\underline{A \approx B}$, where $A \approx B$ is in S.

Given a ground MPU substitution θ, let $\underline{\theta}$ denote the substitution mapping X to t whenever θ maps X to t, and mapping $X_i^{x_i}$ to $B[w := x_i]$ whenever θ maps F_i to $\lambda w \cdot B$.

We first claim that: (i) if θ is a ground MPU substitution, then $\underline{\theta}$ is admissible. This is clear: the domain of $\underline{\theta}$ consists of existential variables, which are mapped to terms that are either ground or where only some x_i is free.

Then: (ii) if the ground MPU substitution θ unifies $A \approx B$ and is grounding for $A \approx B$, then $\underline{\theta}$ satisfies the constraint $\underline{A \approx B}$. If $A = P(t_1)$ and $B = P(t_2)$, then this is clear.

If A is of the form $F_{i_1}(t_1)$ and B is of the form $F_{i_2}(t_2)$, with $F_{i_1} \neq F_{i_2}$, then since θ is grounding, $F_{i_1}, F_{i_2} \in \mathrm{dom}\,\theta$. Let $\theta(F_{i_1})$ be $\lambda w \cdot B_1$ and $\theta(F_{i_2})$ be $\lambda w \cdot B_2$. Since θ unifies $A \approx B$, by definition $B_1[w := t_1\theta] = B_2[w := t_2\theta]$. Since θ and $\underline{\theta}$ agree on first-order (a.k.a., universal) variables, then: $(*)$ $B_1[w := t_1\underline{\theta}] = B_2[w := t_2\underline{\theta}]$. Since $\underline{\theta}$ is admissible by (i), $(x_1 \approx t_1, x_2 \approx t_2)\underline{\theta} = (x_1 \approx t_1\underline{\theta}, x_2 \approx t_2\underline{\theta})$. If this is not unifiable, then clearly $(x_1 \approx t_1, x_2 \approx t_2)\underline{\theta} \Vdash X_{i_1}\underline{\theta} \approx X_{i_2}\underline{\theta}$, proving (ii). Otherwise, an mgu of $(x_1 \approx t_1, x_2 \approx t_2)\underline{\theta}$ is $\sigma =_{\mathrm{df}} [x_1 := t_1\underline{\theta}, x_2 := t_2\underline{\theta}]$; then $X_{i_1}\underline{\theta}\sigma = B_1[w := x_{i_1}]\sigma = B_1[w := t_1\underline{\theta}]$ (because θ is ground, hence no other variable than x_{i_1} is free in $B_1[w := x_{i_1}]$), and similarly $X_{i_2}\underline{\theta}\sigma = B_2[w := t_2\underline{\theta}]$. By $(*)$, it follows that σ unifies $X_{i_1}\underline{\theta}$ and $X_{i_2}\underline{\theta}$, that is, that $(x_1 \approx t_1, x_2 \approx t_2)\underline{\theta} \Vdash X_{i_1}\underline{\theta} \approx X_{i_2}\underline{\theta}$. In other words, $\underline{\theta}$ satisfies $\underline{A \approx B}$.

If $A = F_{i_1}(t_1)$ and $B = P(t_2)$, then let $\theta(F_{i_1})$ be $\lambda x \cdot B_1$, so that $B_1[w := t_1\theta] = P(t_2\theta)$, hence: $(**)$ $B_1[w := t_1\underline{\theta}] = P(t_2\underline{\theta})$. If $(x_{i_1} \approx t_1)\underline{\theta}$ has no unifier, then the claim is obvious. Otherwise, an mgu is $\sigma =_{\mathrm{df}} [x_{i_1} := t_1\underline{\theta}]$, and as above, $X_{i_1}\underline{\theta}\sigma = B_1[w := t_1\underline{\theta}]$, so by $(**)$ $X_{i_1}\underline{\theta}\sigma = P(t_2\underline{\theta})\sigma$, so that $\underline{\theta}$ satisfies $\underline{A \approx B}$. The remaining, symmetrical case is similar.

Finally: (iii) if θ is a ground MPU substitution that is grounding for $A \approx B$, and $\underline{\theta}$ satisfies the constraint $\underline{A \approx B}$, then θ unifies $A \approx B$. If $A = P(t_1)$ and $B = P(t_2)$, then $\underline{\theta}$ unifies $P(t_1)$ with $P(t_2)$, so θ as well, since $\underline{\theta}$ and θ agree on first-order variables, proving (iii).

If A is of the form $F_{i_1}(t_1)$ and B is of the form $F_{i_2}(t_2)$, with $F_{i_1} \neq F_{i_2}$, then since θ is grounding, $F_{i_1}, F_{i_2} \in \operatorname{dom}\theta$. Let $\theta(F_{i_1})$ be $\lambda w \cdot B_1$ and $\theta(F_{i_2})$ be $\lambda w \cdot B_2$. Since $\underline{\theta}$ satisfies $\underline{A \approx B}$, we have $(x_{i_1} \approx t_1, x_{i_2} \approx t_2)\underline{\theta} \Vdash X_{i_1}\underline{\theta} \approx X_{i_2}\underline{\theta}$. Since $\underline{\theta}$ is admissible by (i): $(\dagger)$ $x_{i_1} \approx t_1\underline{\theta}, x_{i_2} \approx t_2\underline{\theta} \Vdash X_{i_1}\underline{\theta} \approx X_{i_2}\underline{\theta}$. Now since θ is grounding, $t_1\underline{\theta}$ and $t_2\underline{\theta}$ are ground terms, so the left-hand side is unifiable: let $\sigma =_{\mathrm{df}} [x_{i_1} := t_1\underline{\theta}, x_{i_2} := t_2\underline{\theta}]$ be one of its mgus. By $(\dagger)$, σ unifies $X_{i_1}\underline{\theta}$ and $X_{i_2}\underline{\theta}$. But, as in case (ii), $X_{i_1}\underline{\theta}\sigma = B_1[w := x_{i_1}]\sigma = B_1[w := t_1\underline{\theta}]$, and similarly $X_{i_2}\underline{\theta}\sigma = B_2[w := t_2\underline{\theta}]$. So $B_1[w := t_1\underline{\theta}] = B_2[w := t_2\underline{\theta}]$. Since $\underline{\theta}$ and θ agree on first-order variables, it follows that $B_1[w := t_1\theta] = B_2[w := t_2\theta]$, therefore θ unifies $A \approx B$. The remaining cases are similar.

By (i)–(iii), the existence of a unifier of S and the satisfiability problem for $\underline{S}$ are equivalent. The reduction from S to $\underline{S}$ is clearly recursive, so constraint satisfiability is undecidable. $\square$

An interesting lesson that this result teaches us is how close solving constraints is to second-order unification. In particular, universal variables are λ-bound variables and existential variables are really first or second-order variables. The latter part is clearer if we understand the existential variables $X^{x_1,\ldots,x_n}$ as analogous to the second-order term $X(x_1, \ldots, x_n)$. Then $\ldots, x_1 \approx t_1, \ldots, x_n \approx t_n \vdash X(x_1, \ldots, x_n) \approx \ldots$ can be transformed into $\ldots, x_1 \approx t_1, \ldots, x_n \approx t_n \vdash X(t_1, \ldots, t_n) \approx \ldots$ without changing the sets of solutions.

We now propose a complete set of rules to solve constraints. Our first move is to simplify constraints.

Lemma 4.8 *For every admissible substitution ϑ:*

(i) $E_1\vartheta \Vdash E_2\vartheta$ *if and only if* $E_1, \overline{\theta} \Vdash E_2$*;*

(ii) $E_1, \overline{\vartheta} \Vdash E_2$ *if and only if* $\overline{\sigma_1}, \overline{\vartheta} \Vdash \overline{\sigma_2}$*, where* $\sigma_1 =_{\mathrm{df}} mgu(E_1)$ *and* $\sigma_2 =_{\mathrm{df}} mgu(E_2\sigma_1)$*.*

where $\perp, \overline{\vartheta}$ *denotes* $\perp$*.*

Proof. Claim (i) is as in Lemma 3.21 (vi). Notice also that: $(\dagger)$ if $E, \overline{\sigma} \Vdash E'\sigma$, then also $E, \overline{\sigma} \Vdash E'$. This is as in Lemma 3.21 (iv).

Conversely: $(\ddagger)$ if $E, \overline{\sigma} \Vdash E'$, then also $E, \overline{\sigma} \Vdash E'\sigma$, whenever σ is idempotent. Indeed, let σ' be $mgu(\overline{\sigma}, E)$. By assumption, σ' unifies E', that is, every equation $s \approx t$ in E'; in other words, $s\sigma' = t\sigma'$ for every $s \approx t$ in E'. But σ' is also an instance of σ, since σ' unifies $\overline{\sigma}$, so we may write σ' as $\sigma\sigma''$ for some substitution σ'', and then $s\sigma\sigma'' = t\sigma\sigma''$ for every $s \approx t$ in E'. Since σ is idempotent, this means that $s\sigma\sigma\sigma'' = t\sigma\sigma\sigma''$, i.e., $s\sigma\sigma' = t\sigma\sigma'$. So σ' unifies every equation $s\sigma \approx t\sigma$ in $E'\sigma$. That is, σ' unifies $E'\sigma$.

Then, notice that whenever $E \Vdash E'$ and E' and E'' have the same set of unifiers, then $E \Vdash E''$. In particular: $(*)$ $E \Vdash E'$ if and only if $E \Vdash \overline{mgu(E')}$. Similarly: $(**)$ $E, E' \Vdash E''$ if and only if $E, \overline{mgu(E')} \Vdash E''$.

Let's prove Claim (ii).

(If) Since $\overline{\sigma_1}, \overline{\vartheta} \Vdash \overline{\sigma_2}$, by $(*)$ and the definition of σ_2, $\overline{\sigma_1}, \overline{\vartheta} \Vdash E_2\sigma_1$. By $(\dagger)$, $\overline{\sigma_1}, \overline{\vartheta} \Vdash E_2$. By $(**)$ and the definition of σ_1, $E_1, \overline{\vartheta} \Vdash E_2$.

(Only if) Since $E_1, \overline{\vartheta} \Vdash E_2$, by $(**)$ and the definition of σ_1, $\overline{\sigma_1}, \overline{\vartheta} \Vdash E_2$. By $(\ddagger)$, $\overline{\sigma_1}, \overline{\vartheta} \Vdash E_2\sigma_1$. By $(*)$ and the definition of σ_2, $\overline{\sigma_1}, \overline{\vartheta} \Vdash \overline{\sigma_2}$. $\square$

Definition 4.9 *Define the following normalization function* N*:* $N(E_1 \Vdash^? E_2) =_{\mathrm{df}} (\overline{\sigma_1} \Vdash^? \overline{\sigma_2})$*, where* $\sigma_1 =_{\mathrm{df}} mgu(E_1)$ *and* $\sigma_2 =_{\mathrm{df}} mgu(E_2\sigma_1)$*. On constraint sets,* $N(K)$ *is defined to be the set of all* $N(E_1 \Vdash^? E_2)$*, for* $E_1 \Vdash^? E_2$ *in* K*.*

Lemma 4.8 then states that K and $N(K)$ are satisfied by exactly the same admissible substitutions. $N(K)$ then consists of constraints that have the following form and are called normalized:

Definition 4.10 *A* normalized constraint *is one of the form:*

$$\zeta_1 \approx s_1, \ldots, \zeta_m \approx s_m \Vdash^? \zeta_{m+1} \approx s_{m+1}, \ldots, \zeta_{m+n} \approx s_{m+n}$$

where the ζ_i*'s are either universal or existential variables, and:*

(i) *the* ζ_i*'s,* $1 \leq i \leq m+n$*, are pairwise distinct;*

(ii) $\mathrm{fv}(s_i) \cap \{\zeta_1, \ldots, \zeta_m\} = \emptyset$*,* $1 \leq i \leq m$*;*

(iii) $\mathrm{fv}(s_i) \cap \{\zeta_1, \ldots, \zeta_{m+n}\} = \emptyset$*,* $m+1 \leq i \leq m+n$*.*

(iv) $\zeta_i \neq s_i$*,* $1 \leq i \leq m+n$*.*

A constraint set is called normalized *if and only if all its constraints are normalized.*

Let's say that a sort τ is *infinite* if and only if there are infinitely many ground terms of type τ. Because we have assumed that there were no empty sorts, checking whether τ is infinite can be tested effectively by building the bottom-up tree automaton whose states are sorts and whose transitions are constructors, and checking whether there is a loop through state τ. A sort that is not infinite is *finite*.

Definition 4.11 *A constraint set* $E_1 \Vdash^? E_2$ *is* pre-solved *in* K *if and only if one of the following holds:*

(i) $E_1 \Vdash^? E_2$ *is* right-pre-solved*:* $E_2 \neq \bot$ *consists entirely of equations of the form* $X_\tau \approx X'_\tau$*;*

(ii) $E_1 \Vdash^? E_2$ *is* left-pre-solved *in* K*:* E_1 *contains an equation* $X_\tau \approx X'_{\tau'}$*, or* $X'_{\tau'} \approx X_\tau$*, where* τ *is an infinite type,* X_τ *does not occur in any right-hand side* E'_2 *of any constraint* $E'_1 \Vdash^? E'_2$ *in* K*.*

A constraint set K *is* pre-solved*, resp.* right-pre-solved*, resp.* left-pre-solved*, if and only if every constraint in* K *is pre-solved, resp. right-pre-solved, resp. left-pre-solved in* K*.*

Lemma 4.12 *Every pre-solved constraint set is satisfiable.*

Proof. For every sort τ, let a^0_τ be a fixed ground term of type τ. For every infinite sort τ, let a^1_τ, $\ldots$, a^k_τ, $\ldots$, be an infinite sequence of ground terms of type τ, all pairwise distinct and distinct from a^0_τ; in this case, let also $X_{\tau,1}$, $\ldots$, X_{τ,k_τ} be the pairwise distinct existential variables that occur as X_τ of case (ii). Let then ϑ be the admissible substitution mapping $X_{\tau,i}$ to a^i_τ, for every infinite sort τ, $1 \leq i \leq k_\tau$, and mapping all other existential variables X_τ free in K to a^0_τ.

For every constraint $E_1 \Vdash^? E_2$ in K, either it obeys (i) or it obeys (ii). In the first case, E_2 consists entirely of equations $X_\tau \approx X'_\tau$. Neither X_τ nor X'_τ can be of the form $X_{\tau,i}$, $1 \leq i \leq k_\tau$, so $X_\tau\vartheta = a^0_\tau = X'_\tau\vartheta$: so ϑ unifies E_2, therefore $\vartheta \models E_1 \Vdash^? E_2$. In the second case, E_1 contains an equation $X_\tau \approx X'_\tau$ or $X'_\tau \approx X_\tau$, with X_τ and X'_τ distinct variables, τ an infinite sort, and X_τ is some $X_{\tau,i}$, $1 \leq i \leq k_\tau$. Then X'_τ is either some $X_{\tau,j}$, $1 \leq j \leq k_\tau$, $i \neq j$ (since X_τ and X'_τ are distinct), so $X'_\tau\vartheta = a^j_\tau$ is not unifiable with $X_\tau\vartheta = a^i_\tau$ (they are distinct ground terms), or X'_τ is some variable of case (i), so that $X'_\tau\vartheta = a^0_\tau$ is again not unifiable with $X_\tau\vartheta$. In any case, $E_1\vartheta$ is not unifiable, so $\vartheta \models E_1 \Vdash^? E_2$. $\square$

We can refine this result as follows. When τ is finite, and there are $k \geq 1$ ground terms of type τ, we may replace condition (ii) by saying that X_τ is left-pre-solved in K provided the conditions of (ii) hold and there are no more than $k-1$ existential variables of type τ free in K. But our unification procedure will be able to enumerate all terms of type τ in finite time in this case, so this does not make it terminate on more problems.

Call a substitution φ from existential variables to extended terms of the same type *pre-admissible* if and only if, for every $X_\tau^S \in \operatorname{dom}\varphi$, the free universal variables in $X_\tau^S\varphi$ are in S, and the free existential variables in $X_\tau^S\varphi$ are of the form $X'^{S'}_{\tau'}$ with $S' \subseteq S$ and $X'^{S'}_{\tau'} \notin \operatorname{dom}\varphi$. Notice that admissible substitution are pre-admissible substitutions mapping variables to extended terms having no free existential variable. Observe also that all pre-admissible substitutions are idempotent.

We define a set of transformation rules on pairs K/φ consisting of a constraint set K and a pre-admissible substitution φ: see Figure 6. The imitation and projection rules **(Imit)** and **(Proj)** are clearly inspired from Huet's higher-order unification procedure [Hue75]. Even more so is the notion of pre-solved system.

$$
\begin{array}{rrcll}
(\bot\Vdash) & K, (\bot \Vdash^{?} E)/\varphi & \to & K/\varphi & \\
(\Vdash\epsilon) & K, (E \Vdash^{?})/\varphi & \to & K/\varphi & \\
(\mathbf{Sub}) & K/\varphi & \to & N(K\psi)/\varphi\psi & (\Vdash^{?} X_\tau^S \approx X'^{S'}_\tau, E) \in K, \\
 & & & & \psi =_{\mathrm{df}} [X_\tau^S := X''^{S\cap S'}_\tau, X'^{S'}_\tau := X''^{S\cap S'}_\tau] \\
 & & & & X''^{S\cap S'}_\tau \text{ fresh} \\
(\mathbf{Imit}) & K/\varphi & \to & N(K\psi)/\varphi\psi & \psi =_{\mathrm{df}} [X_\tau^S := c(X^S_{1\,\tau_1}, \ldots, X^S_{n\,\tau_n})], \\
 & & & & (E_1 \Vdash^{?} E_2) \text{ not right-pre-solved}, \\
 & & & & E_1 \neq \bot,\ X_\tau^S \approx c(t_1, \ldots, t_n) \in E_2, \\
 & & & & \alpha(c) = \tau_1 \times \ldots \tau_n \to \tau,\ X^S_{1\,\tau_1}, \ldots, X^S_{n\,\tau_n} \text{ fresh} \\
(\mathbf{Proj}) & K/\varphi & \to & N(K\psi)/\varphi\psi & X_\tau^S \text{ free in } K,\ x_\tau \in S,\ \psi =_{\mathrm{df}} [X_\tau^S := x_\tau] \\
(\mathbf{Hyp}) & K/\varphi & \to & N(K\psi)/\varphi\psi & \psi =_{\mathrm{df}} [X_\tau^S := c(X^S_{1\,\tau_1}, \ldots, X^S_{n\,\tau_n})], \\
 & & & & (E_1 \Vdash^{?} E_2) \text{ in } K, \\
 & & & & X_\tau^S \in \mathrm{fv}(E_1), \\
 & & & & \alpha(c) = \tau_1 \times \ldots \tau_n \to \tau,\ X^S_{1\,\tau_1}, \ldots, X^S_{n\,\tau_n} \text{ fresh}
\end{array}
$$

Figure 6: Constraint solving rules

Theorem 4.13 (Soundness) *If there is a sequence of transformation steps from $K/[]$ to K_0/φ_0, where K_0 is pre-solved and φ_0 is pre-admissible, then there is an admissible instance ϑ of φ_0 such that $\vartheta \models K$.*

Proof. First, a straightforward induction on the sequence of transformation steps from K/φ to K_0/φ_0 shows that φ_0 is an instance, say $\varphi\psi_0$, of φ, with ψ_0 pre-admissible.

We show that, if there is a sequence of transformation steps from K/φ to $K_0/\varphi\psi_0$, where K_0 is pre-solved and φ and ψ_0 are pre-admissible, then there is an admissible instance ϑ of ψ_0 such that $\vartheta \models K$. This is by induction on the number k of transformation steps from K/φ to K_0/φ_0. If $k = 0$, then $\psi_0 = []$ and $\vartheta \models K_0$ by Lemma 4.12. Otherwise, $k \geq 1$, and examine the first transformation step. The case of $(\bot\Vdash)$ and $(\Vdash\epsilon)$ is trivial, since every substitution satisfies $\bot \Vdash^{?} E$, as well as $E \Vdash^{?}$. In all other cases, K/φ transforms in one step to some $N(K\psi)/\varphi\psi$, where φ_0 is an instance of $\varphi\psi$, say $\varphi\psi\psi_0'$. In particular, $\psi_0 = \psi\psi_0'$. By induction hypothesis, there is an admissible instance ϑ' of ψ_0' such that $\vartheta' \models N(K\psi)$. By Lemma 4.8, $\vartheta' \models K\psi$, so $\vartheta =_{\mathrm{df}} \psi\vartheta'$ is such that $\vartheta \models K$. Moreover, ϑ' is an instance of ψ_0', so ϑ is an instance of $\psi\psi_0' = \psi_0$. $\square$

Theorem 4.14 (Completeness) *Let K be a satisfiable constraint set. Then, there is a sequence of transformation steps from $K/[]$, where $[]$ is the empty substitution, to some K_0/φ_0, where K_0 is pre-solved and φ_0 is pre-admissible.*

Proof. We shall show more generally that we may insist on K_0 being not only pre-solved but right-pre-solved.

Let the size $|\varphi|$ of a substitution be defined as $\sum_{X \in \mathrm{dom}\,\varphi} |X\varphi|$, where $|x| =_{\mathrm{df}} |X| =_{\mathrm{df}} 1$, $c(t_1, \ldots, t_n) =_{\mathrm{df}} 1 + |t_1| + \ldots + |t_n|$. Similarly, let the size be defined on equations, systems, constraints and constraint sets by: $|s \approx t| =_{\mathrm{df}} |s| + |t|$, $|E| =_{\mathrm{df}} \sum_{s \approx t \in E} |s \approx t|$ if $E \neq \bot$, $|\bot| = 0$, $|E_1 \vdash^? E_2| =_{\mathrm{df}} |E_1| + |E_2| + 1$, $|K| =_{\mathrm{df}} \sum_{E_1 \vdash^? E_2 \in K} |E_1 \vdash^? E_2|$.

We first claim that: (i) If K_1 and K_2 are two constraint sets such that K_2 is obtained from K_1 by replacing some constraint $E_1 \vdash^? E_2$ in K_1 by $E_1 \vdash^? \bot$, and if there is a sequence of transformation steps from K_2/φ_1 to some K_0/φ_0, where K_0 is right-pre-solved, then there is another sequence of transformation steps from K_1/φ_1 to K_0/φ_0. This is by induction on the number k of transformations from K_2/φ_1 to K_0/φ_0. If $k = 0$, this is vacuously true, since $E_1 \vdash^? \bot$ cannot be right-pre-solved. Otherwise, $k \geq 1$. Write K_1 as $K, (E_1 \vdash^? E_2)$, so that $K_2 = K, (E_1 \vdash^? \bot)$. The cases are as follows:

- If $E_1 = \bot$ and the first transformation is by $(\bot\vdash)$ applied to $E_1 \vdash^? \bot$, then K_2/φ_1 transforms to K/φ_1, and K_1/φ_1 also transforms to K/φ_1 by the same rule. So both K_2/φ_1 and K_1/φ_1 transform to K_0/φ_0, as claimed.

- If the first transformation is by **(Proj)** on $E_1 \vdash^? \bot$, transforming K_2/φ_1 into $N(K_2\psi)/\varphi_1\psi$, with $\psi =_{\mathrm{df}} [X^S_\tau := x_\tau]$, then X^S_τ is free in E_1, so X^S_τ is also free in $E_1 \vdash^? E_2$, and therefore we can transform K_1/φ_1 into $N(K_1\psi)/\varphi_1\psi$ by **(Proj)** again. Now notice that $N(K_2\psi)$ is $N(K_1\psi)$ with $N((E_1 \vdash^? E_2)\psi)$ replaced by $N((E_1 \vdash^? \bot)\psi)$. Moreover, letting $N((E_1 \vdash^? E_2)\psi)$ be written as $E'_1 \vdash^? E'_2$, $N((E_1 \vdash^? \bot)\psi)$ is $E'_1 \vdash^? \bot$; indeed, $E'_1 = mgu(E_1\psi)$ by definition, so $N((E_1 \vdash^? \bot)\psi) = (mgu(E_1\psi) \vdash^? mgu(\bot\psi)) = E'_1 \vdash^? \bot$. Therefore, the induction hypothesis applies, proving the claim.

- If the first transformation is by **(Hyp)** on $E_1 \vdash^? \bot$, then the argument is similar.

- Otherwise, the first transformation must act on some other constraint, and we conclude by the induction hypothesis directly again.

Let ϑ be some fixed admissible substitution such that $\vartheta \models K$. We now claim that: (ii) for every K_1/φ_1 such that K_1 is normalized, φ_1 is pre-admissible, $\vartheta = \varphi_1\vartheta_1$ for some admissible substitution ϑ_1, and $\vartheta_1 \models K_1$, then we can transform K_1/φ_1 into some K_0/φ_0, where K_0 is right-pre-solved and φ_0 is pre-admissible. This is by induction on $(|\vartheta_1|, |K_1|)$ ordered lexicographically.

If K_1 is right-pre-solved, then we are done. So assume that K_1 is not right-pre-solved. Without loss of generality, we may assume that $\mathrm{dom}\,\vartheta_1 \subseteq \mathrm{fv}(K_1)$. Otherwise, we replace ϑ_1 by some smaller substitution having $\mathrm{dom}\,\vartheta_1 \cap \mathrm{fv}(K_1)$ as domain.

Since K_1 is not right-pre-solved, there is a constraint $E_1 \vdash^? E_2$ in K_1 that is not right-pre-solved in K_1. Write E_2 as $E_3, X_1 \approx X'_1, \ldots, X_p \approx X'_p$, where $p \geq 0$ and E_3 does not contain any equation of the form $X \approx X'$.

If $\mathrm{dom}\,\vartheta_1 \cap \mathrm{fv}(E_1 \vdash^? E_3)$ is empty, then $\vartheta_1 \models E_1 \vdash^? E_2$ implies that $E_1 \vdash E_3, X_1\vartheta \approx X'_1\vartheta, \ldots, X_p\vartheta \approx X'_p\vartheta$, in particular $E_1 \vdash E_3$. Since $E_1 \vdash^? E_2$ is normalized, it is of the form given in Definition 4.10, namely:

$$\zeta_1 \approx s_1, \ldots, \zeta_m \approx s_m \vdash^? \zeta_{m+1} \approx s_{m+1}, \ldots, \zeta_{m+n} \approx s_{m+n}$$

Without loss of generality, we may assume that $\zeta_{m+1} = X_1$, $s_{m+1} = X'_1$, $\ldots$, $\zeta_{m+p} = X_p$, $s_{m+p} = X'_p$, and $p \leq n$. It follows that $E_1 \vdash^? E_3$ equals:

$$\zeta_1 \approx s_1, \ldots, \zeta_m \approx s_m \vdash^? \zeta_{m+p+1} \approx s_{m+p+1}, \ldots, \zeta_{m+n} \approx s_{m+n}$$

With the notations of Definition 4.10, then, $E_1 \vdash E_3$ means that $\zeta_{m+i}\sigma = s_{m+i}\sigma$, for every i, $p + 1 \leq i \leq n$, where $\sigma =_{\mathrm{df}} [\zeta_1 := s_1, \ldots, \zeta_m := s_m]$. By Definition 4.10 (i), $\zeta_{m+i}\sigma = \zeta_{m+i}$, and by Definition 4.10 (iii), $s_{m+i}\sigma = s_{m+i}$. So $\zeta_{m+i} = s_{m+i}$ for every i, $1 \leq i \leq n$. But by Definition 4.10 (iv), $\zeta_{m+i} \neq s_{m+i}$ for every i, $1 \leq i \leq n$. It follows that $n = p$, contradicting the fact that $E_1 \vdash^? E_2$ is not right-pre-solved.

So assume on the other hand that there is a variable X^S_τ in $\operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_1 \overset{?}{\Vdash} E_3)$ (in the "non right-pre-solved part" of the constraint). Without loss of generality, we may choose X^S_τ so that $X^S_\tau\vartheta_1$ is a universal variable x if such an X^S_τ exists; otherwise, we choose X^S_τ so that $X^S_\tau \in \operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_1)$, provided that $\operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_1) \neq \emptyset$; and in all other cases, we pick any X^S_τ in $\operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_3)$. Examine each case in turn:

- Case 1: Let X^S_τ be such that $X^S_\tau\vartheta_1$ be a universal variable x. Then $x \in S$, and x is of type τ, since ϑ_1 is admissible. So rule **(Proj)** applies: let ψ be $[X^S_\tau := x]$, and ϑ_2 be the restriction of ϑ_1 to $\operatorname{dom}\vartheta_1 \setminus \{X^S_\tau\}$. Then $\psi\vartheta_2 = \vartheta_1$, and $|\vartheta_2| < |\vartheta_1|$. Let K_2 be $N(K_1\psi)$. By construction, K_2 is normalized, $\varphi_2 =_{\mathrm{df}} \varphi_1\psi$ is pre-admissible, $\vartheta = \varphi_2\vartheta_2$ (since $\vartheta = \varphi_1\vartheta_1 = \varphi_1\psi\vartheta_2$). Moreover, $\vartheta_1 \models K_1$ implies $\vartheta_2 \models K_1\psi$ by definition, so that $\vartheta_2 \models K_2$, since $K_2 = N(K_1\psi)$ and N preserves the set of satisfying admissible substitutions by Lemma 4.8. Since $|\vartheta_2| < |\vartheta_1|$, the induction hypothesis applies to K_2/φ_2 and ϑ_2, and we are done.

- Case 2: no X^S_τ is such that $X^S_\tau\vartheta_1$ is a universal variable x, but there is an X^S_τ such that $X^S_\tau \in \operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_1)$. In particular, since ϑ_1 is admissible, $X^S_\tau\vartheta_1$ must be of the form $c(t_1, \ldots, t_n)$ for some constructor c of arity $\tau_1 \times \ldots \times \tau_n \to \tau$, and for some terms t_i of respective types τ_i, $1 \leq i \leq n$. Then apply rule **(Hyp)**, getting $K_2 =_{\mathrm{df}} N(K_1\psi)$, $\varphi_2 =_{\mathrm{df}} \varphi_1\psi$: letting ϑ_2 be ϑ_1 restricted to the variables other than X^S_τ, union $[X^S_{1\,\tau_1} := t_1, \ldots, X^S_{n\,\tau_n} := t_n]$, we check that K_2 is normalized, φ_2 is pre-admissible (because $X^S_{1\,\tau_1}, \ldots, X^S_{n\,\tau_n}$ are fresh), $\vartheta = \varphi_2\vartheta_2$ and $\vartheta_2 \models K_2$ (indeed, $\vartheta_1 \models K_1$, and $\vartheta_1 = \psi\vartheta_2$, so $\vartheta_2 \models K_1\psi$, and we use Lemma 4.8). Since $|\vartheta_2| < |\vartheta_1|$, the induction hypothesis applies to K_2/φ_2 and ϑ_2, and we are done.

- Case 3: no X^S_τ is such that $X^S_\tau\vartheta_1$ is a universal variable x, and $\operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_1) = \emptyset$. Then pick any X^S_τ in $\operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_1 \overset{?}{\Vdash} E_3)$, i.e. in $\operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_3)$. Since ϑ_1 is admissible, $X^S_\tau\vartheta_1$ must be of the form $c(t_1, \ldots, t_n)$ for some constructor c of arity $\tau_1 \times \ldots \times \tau_n \to \tau$, and for some terms t_i of respective types τ_i, $1 \leq i \leq n$. Now examine the places where X^S_τ occurs free in E_3:

  - If there is an equation $X^S_\tau \approx d(s_1, \ldots, s_m)$ in E_3, then either $c = d$ and $n = m$, or not. If $c = d$ and $n = m$, then we may apply **(Imit)**, getting $K_2 =_{\mathrm{df}} N(K_1\psi)$, $\varphi_2 =_{\mathrm{df}} \varphi_1\psi$: letting ϑ_2 be ϑ_1 restricted to the variables other than X^S_τ, union $[X^S_{1\,\tau_1} := t_1, \ldots, X^S_{n\,\tau_n} := t_n]$, we check that K_2 is normalized, φ_2 is pre-admissible (because $X^S_{1\,\tau_1}, \ldots, X^S_{n\,\tau_n}$ are fresh), $\vartheta = \varphi_2\vartheta_2$ and $\vartheta_2 \models K_2$ (indeed, $\vartheta_1 \models K_1$, and $\vartheta_1 = \psi\vartheta_2$, so $\vartheta_2 \models K_1\psi$, and we use Lemma 4.8). Since $|\vartheta_2| < |\vartheta_1|$, the induction hypothesis applies to K_2/φ_2 and ϑ_2, and we are done.
    If $c \neq d$, then no substitution may unify $X^S_\tau\vartheta_1 \approx d(s_1, \ldots, s_m)\vartheta_1$, so necessarily $\vartheta_1 \models E_1 \overset{?}{\Vdash} \perp$. Let K_2 be K_1 with $E_1 \overset{?}{\Vdash} E_2$ replaced by $E_1 \overset{?}{\Vdash} \perp$. Since there is at least an equation in E_2, $|E_1 \overset{?}{\Vdash} \perp| < |E_1 \overset{?}{\Vdash} E_2|$, so we may apply the induction hypothesis to conclude that there is a sequence of transformation steps from K_2/φ_1 to K_0/φ_0, K_0 right-pre-solved and φ_0 pre-admissible. By Claim (i), it follows that there is also a similar sequence of transformation steps from K_1/φ_1 to K_0/φ_0.
  - If there is an equation $X^S_\tau \approx x_\tau$ or $x_\tau \approx X^S_\tau$ in E_3, with x_τ a universal variable, then let σ be $mgu(E_1)$. Recall that $E_1 \overset{?}{\Vdash} E_3$ equals:

    $$\zeta_1 \approx s_1, \ldots, \zeta_m \approx s_m \overset{?}{\Vdash} \zeta_{m+p+1} \approx s_{m+p+1}, \ldots, \zeta_{m+n} \approx s_{m+n}$$

    with the properties of Definition 4.10, and $\operatorname{dom}\vartheta_1 \cap \mathrm{fv}(E_1) = \emptyset$. So $E_1\vartheta_1$ equals E_1, and therefore $\sigma =_{\mathrm{df}} mgu(E_1\vartheta_1)$ is exactly $[\zeta_1 := s_1, \ldots, \zeta_m := s_m]$. Since $\vartheta_1 \models E_1 \overset{?}{\Vdash} E_3$, in particular we must have $X^S_\tau\vartheta_1\sigma = x_\tau\vartheta_1\sigma$. But $X^S_\tau\vartheta_1\sigma = c(t_1, \ldots, t_n)\sigma$, while $x_\tau\vartheta_1\sigma = x_\tau\sigma$ (since $\operatorname{dom}\vartheta_1$ contains only existential variables) $= x_\tau$ (by Definition 4.10 (ii) or (i), depending whether $X^S_\tau \approx x_\tau$ or $x_\tau \approx X^S_\tau$ is in E_3). So this case cannot happen.
  - The only remaining case is when X^S_τ is free in E_3 but does not occur as the left or right-hand side of an equation in E_3. That is, X^S_τ only occurs as strict subterm of

t in at least one equation of the form $x \approx t$ in E_3. But as above $\sigma =_{\text{df}} mgu(E_1\vartheta_1)$ is exactly $[\zeta_1 := s_1, \ldots, \zeta_m := s_m]$, and $x\vartheta_1\sigma = t\vartheta_1\sigma$ since $\vartheta_1 \models E_1 \overset{?}{\Vdash} E_3$. This is impossible since $x\vartheta_1\sigma = x\sigma = x$, while $t\vartheta_1\sigma$, having X_τ^S as a strict subterm, must be of the form $c'(t'_1, \ldots, t'_{n'})$.

The Theorem then follows from Claim (ii), with $K_1 =_{\text{df}} N(K)$ (which is normalized by construction), $\varphi_1 =_{\text{df}} []$ (which is clearly pre-admissible), $\vartheta_1 =_{\text{df}} \vartheta$ (so that $\vartheta = \varphi_1\vartheta_1$, ϑ_1 is admissible and $\vartheta_1 \models K_1$). ☐

We now make the following remarks:

1. We don't need rules $(\Vdash\epsilon)$ or (**Sub**) for completeness. Using them just helps simplify constraint sets. Note that $(\Vdash\epsilon)$, (**Sub**) and $(\bot\Vdash)$ can always be applied eagerly without losing completeness.

2. Call a strategy *opportunistic* if it tries to apply deterministic rules (from a given set) first, i.e. rules for which there will never be any need for backtracking. Apart from applying $(\Vdash\epsilon)$, **(Sub)** and $(\bot\Vdash)$ eagerly, we have the following set of deterministic rules.

   First, if the constraint set K contains a constraint $E_1 \overset{?}{\Vdash} E_2$ with E_1 empty, and such that $E_1 \overset{?}{\Vdash} E_2$ is not right-pre-solved, then E_2 contains an equation $\zeta \approx s$, where not both ζ and s are existential variables (and $\zeta \notin \text{fv}(s)$): necessarily, any solution ϑ must be an instance of $[\zeta := s]$. We then have the following cases:

   - $\zeta \approx s$ is of the form $X_\tau^S \approx x_\tau$ or $x_\tau \approx X_\tau^S$, with $x \in S$. Then let ψ be the pre-admissible substitution $[X_\tau^S := x_\tau]$, and deterministically transform K/φ into $N(K\psi)/\varphi\psi$.
   - $\zeta \approx s$ is of the form $X_\tau^S \approx x_\tau$ or $x_\tau \approx X_\tau^S$, with $x \notin S$. Then fail: K is unsatisfiable.
   - $\zeta \approx s$ is of the form $X_\tau^S \approx c(t_1, \ldots, t_n)$: then apply **(Imit)** deterministically. More efficiently, let s be the term obtained from $c(t_1, \ldots, t_n)$ by replacing each free existential variable $X'^{S'}_{\tau'}$ such that $S' \not\subseteq S$ in it by a fresh variable $X''^{S \cap S'}_{\tau'}$. Let ψ be $[X_\tau^S := s]$ (which is pre-admissible by construction), and transform K/φ into $N(K\psi)/\varphi\psi$. This amounts to doing several **(Imit)** steps in a row followed by (**Sub**) steps.
   - $\zeta \approx s$ is of the form $x_\tau \approx c(t_1, \ldots, t_n)$. Then fail: K is unsatisfiable.

   Other opportunistic strategies include always applying **(Imit)** eagerly on variables X_τ^S such that τ has only one constructor c. This is deterministic, and must terminate, since looping behavior would imply that τ was an empty sort, something that we have excluded. This does not preserve the set of solutions, since we might have had a solution ϑ mapping X_τ^S to some $x_\tau \in S$. But then $\vartheta[x_\tau := c(y)]$ is another solution, so satisfiability is preserved.

3. We can stop applying transformation rules to K/φ as soon as K is pre-solved, and we don't need to wait until we have managed to produce a right-pre-solved constraint set. This is useful, as this helps limit the use of rule **(Hyp)**. The latter is indeed uncontrollable, i.e. there is no way to limit its use, once some existential variable occurs on the left-hand side of a constraint in K.

4. There is no provision in the transformation rules for stopping on recognizing unsatisfiable constraint sets. We have given a few cases where this can be done in Remark 2, but we can do a bit more. We can enforce a sufficient halting condition by the following simple test: K is unsatisfiable as soon as K contains a normalized constraint of the form $E_1 \overset{?}{\Vdash} \bot$, where $E_1 =_{\text{df}} x_1 \approx t_1, \ldots, x_n \approx t_n$, and the free existential variables X_τ^S in E_1 are such that $S \cap \{x_1, \ldots, x_n\} = \emptyset$. Indeed, any admissible solution ϑ must instantiate these X_τ^S by terms where $x_1, \ldots, x_n$ are not free, so that $x_1 \approx t_1\vartheta, \ldots, x_n \approx t_n\vartheta$ is a solved form: but then $[x_1 := t_1\vartheta, \ldots, x_n := t_n\vartheta]$ does not unify $\bot$. So such a constraint cannot be satisfied.

5. A more general unsatisfiability test is provided by declaring unsatisfiable any normalized constraint of the form $E_1 \overset{?}{\Vdash} E_2$, where E_1 is as above, but either $E_2 = \bot$ or E_2 contains

an equation of the form $x \approx t$ with x universal and t a term that is either a universal variable, an application $c(t_1, \ldots, t_n)$ or an existential variable X_τ^S with $x \notin S$. Indeed, letting $\sigma =_{\mathrm{df}} [x_1 := t_1\vartheta, \ldots, x_n := t_n\vartheta]$ (the mgu of $E_1\vartheta$), $x\vartheta\sigma = x$ by Definition 4.10 (i), while $t\vartheta\sigma$ cannot be x in any of the three possible cases for t.

6. A strategy that tends to avoid using **(Hyp)** is the following *optimistic strategy*: favor using the deterministic rules of Remarks 1 and 2, then the rules that act on the right of constraints, namely **(Imit)**, and **(Proj)** in the case that $X_\tau \approx x$ or $x \approx X_\tau$ is on the right of some constraint. The idea of this strategy is to try solving constraints $E_1 \Vdash^? E_2$ by looking for solutions ϑ such that $\sigma =_{\mathrm{df}} mgu(E_1\vartheta)$ is not $\perp$, and σ unifies $E_2\vartheta$. This may then find substitutions ϑ that satisfy some other constraint $E_3 \Vdash^? E_4$ by making $E_3\vartheta$ non-unifiable, but we shall dispose of the latter by $(\perp L)$, which is a deterministic rule, rather than by blind applications of **(Hyp)** to $E_3 \Vdash^? E_4$. It is doubtful that we can get rid of **(Hyp)** entirely without losing completeness, so that **(Hyp)** should still be tried from time to time.

7. Although **(Hyp)** is uncontrollable, i.e., there is no real restriction on its application, some heuristics apply: first, it is always good to try applying **(Hyp)** when there is an equation $X_\tau^S \approx c(t_1, \ldots, t_n)$ on the left of some constraint $E_1 \Vdash^? E_2$ in K, and τ has at least two constructors c and d: instantiating X_τ^S by $\psi =_{\mathrm{df}} [X_\tau^S := d(X_1, \ldots, X_m)]$ will then offer us the opportunity to get rid of at least one constraint in $N(K\psi)$, namely $N(E_1\psi \Vdash^? E_2\psi)$, by $(\perp\Vdash)$. Indeed, $E_1\psi$ contains the equation $d(X_1, \ldots, X_m) \approx c(t_1, \ldots, t_n)$, which cannot be unified.

   Similarly, if some constraint $E_1 \Vdash^? E_2$ in K contains equations of the form $x_{1\,\tau_1} \approx t_1(X^{S_1}_{1\,\tau_2})$, $x_{2\,\tau_2} \approx t_2(X^{S_2}_{2\,\tau_3}), \ldots, x_{k\,\tau_k} \approx t_k(X^{S_k}_{k\,\tau_1})$, $k \geq 1$, where $t(X)$ denotes any term t with X free in t, if $t_i(X^{S_i}_{i\,\tau_i}) \neq X^{S_i}_{i\,\tau_i}$ for at least one i, $1 \leq i \leq k$, and if $X_1, X_2, \ldots, X_k$ are pairwise distinct, $x_2 \in S_1, \ldots, x_k \in S_{k-1}$, $x_1 \in S_k$, then we may apply **(Proj)** k times and instantiate by $\psi =_{\mathrm{df}} [X_1^{S_1} := x_2, \ldots, X_{k-1}^{S_{k-1}} := x_k, X_k^{S_k} := x_1]$: this makes $E_1\psi$ non-unifiable because of the occurs-check, and allows us to use $(\perp\Vdash)$ to get rid of at least one constraint.

   Finally, we can also use **(Hyp)** on variables X_τ^S where τ is a finite type, i.e. when there are only finitely many ground terms of type τ. Indeed, we shall be able to apply **(Hyp)** in this way only finitely many times. But this is still very non-deterministic.

8. Although constraint solving is in general undecidable, there are several decidable particular cases.

   First, in a language with no constructors, neither **(Imit)** nor **(Hyp)** ever applies. It is easy to see that constraint solving is then in NP, where the non-determinism arises from the application of **(Proj)** on variables X_τ^S with S of cardinality at least 2. Although having no constructors may seem to defeat the purpose of this paper, notice that LKc$_\approx$ without constructors is a sound and complete deduction system for first-order logic with equality, where functions are coded as functional predicates: we therefore get a sound and complete tableau calculus for first-order logic with equality, with a decidable unification problem. As far as I know, this is the first such system (recall that E-unification, as well as simultaneous rigid E-unification are undecidable [DV96]).

   Well, this remark is void, strictly speaking, since we have been assuming that every datatype contained at least one ground term. But this remark can be extended to the case where all datatypes contain just one constant, or more generally, to the case where all datatypes are finite. In this case, **(Imit)** or **(Hyp)** can only be applied finitely many times, and the unification procedure terminates again.

   Second, if all constraints that the tableau procedure produces have an empty left-hand side, i.e. if they are all of the form $\Vdash^? E$, then the arguments of Remark 2, plus a straightforward induction on $|K|$, show that constraint solving in this case is polynomial-time decidable. This is the case in particular when we try to prove formulas F where equations only appear at positive occurrences (i.e., under an even number of negations, where negations are left-hand sides of $\supset$), and $\phi(P) = \emptyset$ for every predicate P occurring in F. Notably, this is the case for Horn formulas with equality implemented by unification, as in Prolog, where equality denotes (sound) unification. Recall that, in Prolog, you may write clauses with an equation in the body of clauses (where they are positive, since clauses are negative), but you cannot write a clause with an equation as head.

9. We can also deal with constraint solving by approximation; this is similar to the idea of abstract interpretation [CC77] used in static program analysis. The idea is to replace constraint solving by decidable problems which are either upper or lower approximations of constraint solving.

   Lower approximations of constraint solving are predicates $P^\flat$ such that whenever $P^\flat(K)$ is true, K is satisfiable. One such predicate is the pre-solved predicate (Definition 4.11, Lemma 4.12). A better one is that which does all first-order unifications as in Remark 2, and returns true if the resulting system is pre-solved, and false otherwise. Another one—not the most clever—uses depth bounds on the terms that may be assigned to each existential variable, and returns true if and only if the algorithm terminates within this bound; this is easy to code: assign integer levels to second-order variables, refuse to apply **(Imit)**, **(Proj)**, **(Hyp)** if the level n of X^S_τ is zero, otherwise apply them so that all fresh variables receive level $n-1$. (Compare also with [GL97].)

   Lower approximations are useful: replacing general constraint solving by the computation of $P^\flat(K)$ produces a terminating, albeit incomplete, constraint solving algorithm. Provided $P^\flat(K)$ is smart enough, we may decide that any sequent leading only to closed tableaux T such that $P^\flat(K(T))$ is false is too complex to prove, although the sequent might be provable. In software verification, this may indicate that the reasons why the piece of software under consideration is correct are too intricate, and that it might be worth to consider simplifying its design. In general, if completeness is desirable, it is still interesting to compute $P^\flat(K)$ before we launch the full satisfiability procedure of Figure 6; if $P^\flat(K)$ is true, then we can stop right there without launching the full satisfiability procedure. Of course, we can also compute $P^\flat(K)$ from time to time during the full satisfiability procedure of Figure 6, to stop it whenever it is obvious that we have reached a satisfiable constraint.

   Upper approximations of constraint solving are predicates $P^\sharp$ such that whenever K is satisfiable, then $P^\sharp(K)$ is true, but the converse may not hold. If $P^\sharp$ is computable, then it can be used during tableau expansion to enforce that only open branches of the form $\vec{x}\,;K\,;E\,;S$ and closed branches K with $P^\sharp(K)$ true are ever generated. This prunes the search space by disposing of branches that are *obviously* not satisfiable—i.e., such that $P^\sharp(K)$ is false, hence K is unsatisfiable. When a closed tableau T is obtained, it just remains to check whether $K(T)$ is satisfiable, by the complete algorithm of Figure 6 for example, or using lower approximations $P^\flat$.

   As an example of upper approximation, we may use the unsatisfiability tests of Remarks 4 and 5 as (negations of) such predicates $P^\sharp$. We can however imagine more clever approximations. For example, we can turn K into a *linear constraint* $\ell(K)$, that is, one where every existential variable occurs at most once, by renaming each distinct occurrence of any existential variable X^S_τ to a fresh existential variable $X^S_{i\,\tau}$, $i \geq 0$. Then define $P^\sharp(K)$ as true if and only if $\ell(K)$ is satisfiable. This is clearly an upper approximation. Moreover, we conjecture that $P^\sharp$ is a decidable predicate, much like linear second-order unification [Dow93].

5 Structural Induction

We now wish to consider some sorts as inductive. For example, we wish to consider `nat` as the *smallest* set of ground terms containing `0` and such that $\mathtt{S}(t) \in$ `nat` whenever $t \in$ `nat`.

To this end, we first extend Definition 1.1 so as to mark all sorts that we wish to consider inductive:

Definition 5.1 *An* inductive language *is a tuple* $(\mathcal{C}, \mathcal{P}, \alpha, \phi, \mathcal{I})$*, where the tuple* $(\mathcal{C}, \mathcal{P}, \alpha, \phi)$ *is a language and* $\mathcal{I} \subseteq \mathcal{S}$ *is a finite set of sorts, called the* inductive sorts.

The purpose of the set $\mathcal{I}$ is to identify sorts that we wish to consider as inductive. Semantically, the interpretations I we shall be interested in, from now on, are constrained to *respect inductive sorts*, in that the Cartesian product of all $I(\tau)$'s, τ inductive, should be minimal for the subset ordering when the values of $I(\tau)$, τ not inductive, is left fixed. Formally, this condition is defined by a generalized induction principle:

Definition 5.2 *Let* $\tau_1, \ldots, \tau_q$ *denote the inductive sorts (in* $\mathcal{I}$*), and* $\tau_{q+1}, \ldots, \tau_{q'}$ *denote the non-inductive sorts,* $0 \leq q \leq q'$*. An interpretation* I respects inductive sorts *if and only if, for every sets* $T_1 \subseteq I(\tau_1), \ldots, T_q \subseteq I(\tau_q)$, $T_{q+1} =_{\mathrm{df}} I(\tau_{q+1}), \ldots, T_{q'} =_{\mathrm{df}} I(\tau_{q'})$ *such that:*

- *for every constructor* c*, of arity* $\tau_{i_1} \times \ldots \times \tau_{i_n} \to \tau_{i_0}$*, for every* $v_1 \in T_{i_1}, \ldots, v_n \in T_{i_n}$*,* $I(c)(v_1, \ldots, v_n) \in T_{i_0}$*,*

then $T_1 = I(\tau_1), \ldots, T_q = I(\tau_q)$*.*

We still assume that there is at least one ground term of each sort. Contrarily to the non-inductive case, this restricts the generality of our approach: we cannot add constant constructors to inductive sorts that have no ground term (*empty sorts*), since this would change the semantics. However, empty inductive sorts are a chore to deal with; and it is easy to detect which inductive sorts are empty (sorts are states of a deterministic bottom-up tree automaton whose transitions are constructors, and this problem is the polynomial-time-decidable automaton emptiness problem). As dealing with empty sorts would distract us from the main points of this paper, we simply assume that there are none. They can be dealt with, for example, by forbidding the use of the usual quantifier rules $(\forall R)$, $(\forall L)$, $(\exists L)$ and $(\exists R)$ when τ is an empty sort, and replacing them with the following rules:

$$\frac{}{\sigma; \Gamma, \exists x_\tau \cdot F \vdash \Delta} \qquad \frac{}{\sigma; \Gamma \vdash \forall x_\tau \cdot F, \Delta}$$

when τ is an inductive empty sort. These rules are in fact particular cases of the induction schemes that we shall describe next.

We deal with inductive sorts in the deduction system by adding standard *structural induction rules* $(Ind_\tau R)$ and $(Ind_\tau L)$ to $\mathrm{LKc}_\approx$, for every inductive sort τ, yielding a new sequent system $\mathrm{LKc}_\approx^{ind}$. These rules resemble the ones of Coq [BBC+97], notably.

Say that a constructor c is a *constructor of* τ if and only if $\alpha(c)$ is of the form $\tau_1 \times \ldots \times \tau_n \to \tau'$ with $\tau' = \tau$. Say that a sort τ *has constructors* $c_1, \ldots, c_p$ if and only if the set of constructors of τ is exactly $\{c_1, \ldots, c_p\}$. Let τ be an inductive sort, having constructors $c_1, \ldots, c_p$; let $\alpha(c_i)$ be $\tau_{i1} \times \ldots \times \tau_{in_i} \to \tau$. Write $\bigwedge_{\tau_{ij}=\tau} F_j \supset G$ for the formula $F_{j_1} \supset \ldots \supset F_{j_k} \supset G$, where $j_1 < \ldots < j_k$ is the sequence of indices such that $\tau_{ij} = \tau$. Similarly, let $\bigwedge_{\tau_{ij}=\tau} F_j \wedge G$ the formula $F_{j_1} \wedge \ldots \wedge F_{j_k} \wedge G$. The standard structural induction rule for τ is then:

$$\frac{\begin{array}{c}\sigma; \Gamma \vdash \forall x^1_{\tau_{11}}, \ldots, x^{n_1}_{\tau_{1n_1}} \cdot \bigwedge\limits_{\tau_{1j}=\tau} F[x_\tau := x^j_{\tau_{1j}}] \supset F[x_\tau := c_1(x^1, \ldots, x^{n_1})], \Delta \\ \vdots \\ \sigma; \Gamma \vdash \forall x^1_{\tau_{p1}}, \ldots, x^{n_p}_{\tau_{pn_p}} \cdot \bigwedge\limits_{\tau_{pj}=\tau} F[x_\tau := x^j_{\tau_{pj}}] \supset F[x_\tau := c_p(x^1, \ldots, x^{n_p})], \Delta\end{array}}{\sigma; \Gamma \vdash \forall x_\tau \cdot F, \Delta} \; (Ind_\tau R)$$

where the variables $x^j_{\tau_i j}$, $1 \leq i \leq p$, $1 \leq j \leq n_i$, are fresh, i.e., not free in F. The interested reader may check that the induction principles given for `nat` and `natlist` in the introduction follow this pattern.

Dually, we consider the following left induction rule:

$$\dfrac{\begin{array}{c}\sigma; \Gamma, \exists x^1_{\tau_{11}}, \ldots, x^{n_1}_{\tau_{1n_1}} \cdot \bigwedge_{\tau_{1j}=\tau} \neg F[x_\tau := x^j_{\tau_{1j}}] \wedge F[x_\tau := c_1(x^1, \ldots, x^{n_1})] \vdash \Delta \\ \vdots \\ \sigma; \Gamma, \exists x^1_{\tau_{p1}}, \ldots, x^{n_p}_{\tau_{pn_p}} \cdot \bigwedge_{\tau_{pj}=\tau} \neg F[x_\tau := x^j_{\tau_{pj}}] \wedge F[x_\tau := c_p(x^1, \ldots, x^{n_p})] \vdash \Delta\end{array}}{\sigma; \Gamma, \exists x_\tau \cdot F \vdash \Delta}\ (Ind_\tau L)$$

which can be obtained from $(Ind_\tau R)$ by using De Morgan rules.

In general, other, more expressive or less expressive induction principles are possible, of course. The main defect of the induction principles above are that they do not handle mutually inductive sorts gracefully: e.g., two inductive sorts $\mathtt{A}$ and $\mathtt{B}$ with constructors $a : \mathtt{B} \to \mathtt{A}$, $a_0 : \mathtt{A}$ and $b : \mathtt{A} \to \mathtt{B}$ will produce the following right induction principles:

$$\dfrac{\sigma; \Gamma \vdash F[x_{\mathtt{A}} := a_0], \Delta \quad \sigma; \Gamma \vdash \forall y_{\mathtt{B}} \cdot F[x_{\mathtt{A}} := a(y_{\mathtt{B}})], \Delta}{\sigma; \Gamma \vdash \forall x_{\mathtt{A}} \cdot F, \Delta}\ (Ind_{\mathtt{A}} R)$$

$$\dfrac{\sigma; \Gamma \vdash \forall y_{\mathtt{A}} \cdot F[x_{\mathtt{B}} := b(y_{\mathtt{A}})], \Delta}{\sigma; \Gamma \vdash \forall x_{\mathtt{B}} \cdot F, \Delta}\ (Ind_{\mathtt{B}} R)$$

from which it is impossible to derive the more general induction principle:

$$\dfrac{\sigma; \Gamma \vdash F[x_{\mathtt{A}} := a_0], \Delta \quad \sigma; \Gamma \vdash \forall z_{\mathtt{A}} \cdot F[x_{\mathtt{A}} := z_{\mathtt{A}}] \supset F[x_{\mathtt{A}} := a(b(z_{\mathtt{A}}))], \Delta}{\sigma; \Gamma \vdash \forall x_{\mathtt{A}} \cdot F, \Delta}$$

Such general mutual structural induction principles are however too complicated to describe here; the interested reader should be able to infer the general pattern from the example above. Therefore, we shall only study the $(Ind_\tau R)$ and $(Ind_\tau L)$ rules here.

Theorem 5.3 *If $\sigma; \Gamma \vdash \Delta$ is provable in* $\mathrm{LKc}^{ind}_{\approx}$*, then for every free equational interpretation I that respects inductive sorts and functionalities, for every valuation ρ, $I, \rho \models (\sigma; \Gamma \vdash \Delta)$.*

Proof. As for Theorem 3.4. There are only two additional cases, when the last rule used is $(Ind_\tau R)$ or $(Ind_\tau L)$. Consider $(Ind_\tau R)$, the other case is similar (or use De Morgan's laws to reduce the latter to the former): let τ be an inductive sort, having constructors $c_1, \ldots, c_p$; let $\alpha(c_i)$ be $\tau_{i1} \times \ldots \times \tau_{in_i} \to \tau$. By induction hypothesis, (i) if $I, \rho \models \sigma$ and $I, \rho \models F'$ for every F' in Γ, then: (ii) for all i, $1 \leq i \leq p$, for some formula G_i in $\forall x^1_{\tau_{i1}}, \ldots, x^{n_i}_{\tau_{in_i}} \cdot \bigwedge_{\tau_{ij}=\tau} F[x_\tau := x^j_{\tau_{ij}}] \supset F[x_\tau := c_i(x^1_{\tau_{i1}}, \ldots, x^{n_i}_{\tau_{in_i}})], \Delta$, we have $I, \rho \models G_i$. Assume also that: (iii) $I, \rho \models \sigma$ and $I, \rho \models F'$ for every F' in Γ. So (ii) holds. Note that in the case that, for some i, $1 \leq i \leq p$, G_i is in Δ, then the claim is trivially true, by discharging (iii).

So assume that: (iv) for every i such that $1 \leq i \leq p$, $I, \rho \models \forall x^1_{\tau_{i1}}, \ldots, x^{n_i}_{\tau_{in_i}} \cdot \bigwedge_{\tau_{ij}=\tau} F[x_\tau := x^j_{\tau_{ij}}] \supset F[x_\tau := c_i(x^1_{\tau_{i1}}, \ldots, x^{n_i}_{\tau_{in_i}})]$. We exploit the fact that I respects inductive sorts, and use the notations of Definition 5.2. Without loss of generality, let τ be τ_1. We define the sets $T_1, \ldots, T_{q'}$ as follows: for every j, $2 \leq k \leq q'$, let T_j be $I(\tau_j)$, and T_1 is the set of $v \in I(\tau)$ such that $I, \rho[x := v] \models F$. For every constructor c of arity $\tau_{i_1} \times \ldots \times \tau_{i_n} \to \tau_{i_0}$, we claim that: (v) for every $v_1 \in T_{i_1}, \ldots, v_n \in T_{i_n}$, $I(c)(v_1, \ldots, v_n) \in T_{i_0}$. This is obvious if $i_0 \neq 1$. If $i_0 = 1$, then $c = c_i$ for some i, $1 \leq i \leq p$, since the c_i's are the constructors of $\tau = \tau_1$. So by (iv), if $I, \rho[x^1_{\tau_{i_1}} := v_{i_1}, \ldots, x^{n_i}_{\tau_{in_i}} := v_{in_i}] \models F[x_\tau := x^j_{\tau_{i_j}}]$ for all j such that $\tau_{i_j} = \tau$, then $I, \rho[x^1_{\tau_{i_1}} := v_{i_1}, \ldots, x^{n_i}_{\tau_{in_i}} := v_{in_i}] \models F[x_\tau := c_i(x^1_{\tau_{i_1}}, \ldots, x^{n_i}_{\tau_{in_i}})]$. But for all j's such that $\tau_{i_j} = \tau$, $v_j \in T_{i_j} = T_1$ means precisely that $I, \rho[x^1_{\tau_{i_1}} := v_{i_1}, \ldots, x^{n_i}_{\tau_{in_i}} := v_{in_i}] \models F[x_\tau := x^j_{\tau_{i_j}}]$ (since $x^1_{\tau_{i_1}}$,

$\ldots, x^{n_i}_{\tau_{i_{n_i}}}$ are fresh variables), so $I, \rho[x^1_{\tau_{i_1}} := v_{i_1}, \ldots, x^{n_i}_{\tau_{i_{n_i}}} := v_{i_{n_i}}] \models F[x_\tau := c_i(x^1_{\tau_{i_1}}, \ldots, x^{n_i}_{\tau_{i_{n_i}}})]$; by definition, this means that $I(c)(v_1, \ldots, v_n)$ is in T_1. This proves (v). In turn, (v) and the fact that I respects inductive sorts implies that $T_1 = I(\tau)$, so $I, \rho \models \forall x_\tau \cdot F$, and the claim is proved by discharging (iii). □

We cannot hope to extend the Completeness Theorem 3.12 in the case of infinite inductive sorts: this is because of Gödel's first Incompleteness Theorem [Fef84]. It is interesting to point out what fails in the technique we used in Theorem 3.12: we cannot describe fully the minimality conditions of Definition 5.2 by any set, even infinite, of axioms on our given language. The problem is that the minimality conditions of Definition 5.2 must at least be stated for every possible language extension. However, the technique of Theorem 3.12 immediately shows that $\mathrm{LKc}^{ind}_{\approx}$ on the language whose only inductive sort is `nat` is complete for first-order arithmetic $\mathbf{PA}_1$ (as defined in [Joh92], without addition or multiplication); i.e., every $\mathbf{PA}_1$-provable formula is provable in $\mathrm{LKc}^{ind}_{\approx}$. This naturally extends to $\mathbf{PA}_1$ with addition and multiplication (add defining axioms and totality axioms), or with other primitive recursive functions (same argument), or to other inductive theories, provided that we consider not only standard but also non-standard models.

Cut elimination does not work either. The problem is that we cannot eliminate, say, cuts of the following form (on `nat` here):

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\pi_1\\ \sigma;\Gamma\vdash F[x:=\mathtt{0}],\Delta\end{array}\quad \begin{array}{c}\vdots\,\pi_2\\ \sigma;\Gamma\vdash \forall y_{\mathtt{nat}}\cdot F[x:=y]\\ \supset F[x:=\mathtt{S}(y)],\Delta\end{array}}{\sigma;\Gamma\vdash\forall x_{\mathtt{nat}}\cdot F,\Delta}(Ind_{\mathtt{nat}}R)\quad \dfrac{\begin{array}{c}\vdots\,\pi_3\\ \sigma;\Gamma,\forall x_{\mathtt{nat}}\cdot F, F[x:=t]\vdash\Delta\end{array}}{\sigma;\Gamma,\forall x_{\mathtt{nat}}\cdot F\vdash\Delta}(\forall L)}{\sigma;\Gamma\vdash\Delta}(Cut)$$

when t contains free variables of sort `nat`. There is no problem when t does not contain any free variable of sort `nat`, i.e. when t is of the form $\mathtt{S}(\ldots(\mathtt{S}(\mathtt{0}))\ldots)$, with n occurrences of $\mathtt{S}$, for some $n \in \mathbb{N}$: we can derive $\sigma;\Gamma \vdash F[x := \mathtt{0}], \Delta$ by π_1, then $\sigma;\Gamma \vdash F[x := \mathtt{0}] \supset F[x := \mathtt{S}(\mathtt{0})], \Delta$ (by π_2 and $(\forall E)$; recall that the latter natural deduction rule can be simulated by $(\forall L)$, $(Close)$ and (Cut)), then $\sigma;\Gamma \vdash F[x := \mathtt{S}(\mathtt{0})] \supset F[x := \mathtt{S}(\mathtt{S}(\mathtt{0}))], \Delta$, etc. Cut them all together (using $(\supset E)$, which can be simulated by the cut rule), then with the conclusion of π_3 with cut-formula $F[x := t]$, then again with the left premise of the (Cut) above with cut-formula $\forall x_{\mathtt{nat}} \cdot F$.

When t contains some free variables of sort `nat`, we use Lemma 3.22 (or rather, its analog for $\mathrm{LKc}^{ind}_{\approx}$) to instantiate them to some closed term, say $\mathtt{0}$. (This uses the fact that `nat` is not empty in an essential way.) Let θ be the grounding substitution mapping the free variables of sort `nat` to $\mathtt{0}$. Then, the informal argument above shows how to convert the cut between $(Ind_{\mathtt{nat}}R)$ and $(\forall L)$ into a sequence of cuts. However, the transformed proof no longer proves $\sigma;\Gamma \vdash \Delta$ in general, but $mgu(\overline{\sigma}\theta);\Gamma\theta \vdash \Delta\theta$. This makes any instance of $(\forall R)$ or $(\exists L)$ under the transformed (Cut) illegal, as soon as the generalized variable z is in $\mathrm{dom}\,\theta$. For example, take σ empty, Γ empty, $\Delta =_{\mathrm{df}} P(y_{\mathtt{nat}})$, and $t =_{\mathrm{df}} z_{\mathtt{nat}}$, then we may be faced with the following situation:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\,\pi_1\\ ;\vdash F[x:=\mathtt{0}],P(z_{\mathtt{nat}})\end{array}\quad \begin{array}{c}\vdots\,\pi_2\\ ;\vdash \forall y_{\mathtt{nat}}\cdot F[x:=y]\supset\\ F[x:=\mathtt{S}(y)],P(z_{\mathtt{nat}})\end{array}}{;\vdash\forall x_{\mathtt{nat}}\cdot F,P(z_{\mathtt{nat}})}(Ind_{\mathtt{nat}}R)\quad \dfrac{\begin{array}{c}\vdots\,\pi_3\\ ;\forall x_{\mathtt{nat}}\cdot F, F[x:=z_{\mathtt{nat}}]\vdash P(z_{\mathtt{nat}})\end{array}}{;\forall x_{\mathtt{nat}}\cdot F\vdash P(z_{\mathtt{nat}})}(\forall L)}{;\vdash P(z_{\mathtt{nat}})}(Cut)}{;\vdash\forall x_{\mathtt{nat}}\cdot P(x)}(\forall R)$$

Taking $\theta =_{\mathrm{df}} [z := \mathtt{0}]$, the (Cut) is transformed into a proof of $;\vdash P(\mathtt{0})$, from which we cannot deduce $;\vdash \forall x_{\mathtt{nat}} \cdot P(x)$ any longer.

In fact, this is a well-known conundrum [Gir92]: cut elimination does not work in the presence of induction rules of the form above, and it is necessary to cheat, namely to hide some cut inside the induction rules to get cut elimination. We may do this, for example, by adapting McDowell and Miller's induction rule [MM98]: let $\mathrm{LKc}_{\approx}^{ind'}$ be $\mathrm{LKc}_{\approx}$ plus the following rules:

$$\dfrac{\begin{array}{c}\sigma;\Gamma\vdash\forall x^1_{\tau_{11}},\ldots,x^{n_1}_{\tau_{1n_1}}\cdot\bigwedge_{\tau_{1j}=\tau}F[x_\tau:=x^j_{\tau_{1j}}]\supset F[x_\tau:=c_1(x^1,\ldots,x^{n_1})],\Delta\\ \vdots\\ \sigma;\Gamma\vdash\forall x^1_{\tau_{p1}},\ldots,x^{n_p}_{\tau_{pn_p}}\cdot\bigwedge_{\tau_{pj}=\tau}F[x_\tau:=x^j_{\tau_{pj}}]\supset F[x_\tau:=c_p(x^1,\ldots,x^{n_p})],\Delta\\ \sigma;\Gamma,F[x:=t]\vdash\Delta\qquad\rhd t:\tau\end{array}}{\sigma;\Gamma\vdash\Delta}\,(Ind'_\tau R)$$

where the variables $x^j_{\tau_{ij}}$, $1\leq i\leq p$, $1\leq j\leq n_i$, are fresh, i.e., not free in F; and:

$$\dfrac{\begin{array}{c}\sigma;\Gamma,\exists x^1_{\tau_{11}},\ldots,x^{n_1}_{\tau_{1n_1}}\cdot\bigwedge_{\tau_{1j}=\tau}\neg F[x_\tau:=x^j_{\tau_{1j}}]\wedge F[x_\tau:=c_1(x^1,\ldots,x^{n_1})]\vdash\Delta\\ \vdots\\ \sigma;\Gamma,\exists x^1_{\tau_{p1}},\ldots,x^{n_p}_{\tau_{pn_p}}\cdot\bigwedge_{\tau_{pj}=\tau}\neg F[x_\tau:=x^j_{\tau_{pj}}]\wedge F[x_\tau:=c_p(x^1,\ldots,x^{n_p})]\vdash\Delta\\ \sigma;\Gamma\vdash F[x:=t],\Delta\qquad\rhd t:\tau\end{array}}{\sigma;\Gamma\vdash\Delta}\,(Ind'_\tau L)$$

with similar side-conditions.

Lemma 5.4 *$\mathrm{LKc}_{\approx}^{ind}$ and $\mathrm{LKc}_{\approx}^{ind'}$ prove exactly the same sequents.*

Proof. On the one hand, we may translate the $(Ind'_\tau R)$ rule into the following $\mathrm{LKc}_{\approx}^{ind}$ proof fragment:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma;\Gamma\vdash\forall x^1_{\tau_{i1}},\ldots,x^{n_i}_{\tau_{in_i}}\cdot\bigwedge_{\tau_{ij}=\tau}F[x_\tau:=x^j_{\tau_{ij}}]\\ \supset F[x_\tau:=c_i(x^1,\ldots,x^{n_i})],\Delta\\ (1\leq i\leq p)\end{array}}{\sigma;\Gamma\vdash\forall x_\tau\cdot F,\Delta}\,(Ind_\tau R)\quad\dfrac{\begin{array}{c}\vdots\\ \sigma;\Gamma,F[x:=t]\vdash\Delta\end{array}}{\sigma;\Gamma,\forall x_\tau\cdot F\vdash\Delta}\,(\forall L)}{\sigma;\Gamma\vdash\Delta}\,(Cut)$$

and similarly for $(Ind'_\tau L)$, using $(Ind_\tau L)$, $(\exists R)$ and (Cut).

On the other hand, we may translate $(Ind_\tau R)$ into the following $\mathrm{LKc}_{\approx}^{ind'}$ proof fragment:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \sigma;\Gamma\vdash\forall x^1_{\tau_{i1}},\ldots,x^{n_i}_{\tau_{in_i}}\cdot\bigwedge_{\tau_{ij}=\tau}F[x_\tau:=x^j_{\tau_{ij}}]\\ \supset F[x_\tau:=c_i(x^1,\ldots,x^{n_i})],F[x:=y_\tau],\Delta\\ (1\leq i\leq p)\end{array}\quad\dfrac{}{\sigma;\Gamma,F[x:=y_\tau]\vdash F[x:=y_\tau],\Delta}\,(Close')}{\sigma;\Gamma\vdash F[x:=y_\tau],\Delta}\,(Ind'_\tau R)}{\sigma;\Gamma\vdash\forall x_\tau\cdot F,\Delta}\,(\forall R)$$

where some weakenings have been used. (Check that weakening is admissible in $\mathrm{LKc}_{\approx}^{ind'}$, i.e. Lemma 3.11 works again. Check also that rule $(Close')$ is still admissible in $\mathrm{LKc}_{\approx}^{ind'}$.) Similarly for $(Ind_\tau L)$, which we translate using $(Ind'_\tau L)$, $(\exists L)$ and weakenings. □

Theorem 5.5 *A sequent has a proof in $\mathrm{LKc}_{\approx}^{ind'}$ if and only if it has a proof in $\mathrm{LKc}_{\approx}^{ind'}$ without using the (Cut) rule.*

Proof. The if direction is obvious. Conversely, we reprove all the results of Section 3.4 for $\mathrm{LKc}_{\approx}^{ind'}$. Lemma 3.24 extends trivially, since the new induction rules $(Ind'_\tau R)$ and $(Ind'_\tau L)$ have no principal formula. We then conclude by the same argument as in Theorem 3.25. $\square$

This result is so easy that it is almost vacuous. We can do better by implementing recursor reductions. *Recursor redexes* are proofs of the form:

$$\dfrac{\begin{array}{c} \vdots\ \pi_i \\ \sigma;\Gamma \vdash \forall x^1_{\tau_{i1}},\ldots,x^{n_i}_{\tau_{in_i}} \cdot \bigwedge_{\tau_{ij}=\tau} F[x_\tau := x^j_{\tau_{ij}}] \\ \supset F[x_\tau := c_i(x^1,\ldots,x^{n_i})],\Delta \\ (1 \leq i \leq p) \end{array} \qquad \begin{array}{c} \vdots\ \pi_0 \\ \sigma;\Gamma, F[x := c_k(t_1,\ldots,t_{n_k})] \vdash \Delta \end{array}}{\sigma;\Gamma \vdash \Delta}\ (Ind'_\tau R)$$

which we can reduce to a proof built as follows. First, build the following proofs π'_j, for every j such that $\tau_{kj} = \tau$:

$$\dfrac{\begin{array}{c} \vdots\ \pi_i \\ \sigma;\Gamma \vdash \forall x^1_{\tau_{i1}},\ldots,x^{n_i}_{\tau_{in_i}} \cdot \bigwedge_{\tau_{ij}=\tau} F[x_\tau := x^j_{\tau_{ij}}] \\ \supset F[x_\tau := c_i(x^1,\ldots,x^{n_i})], F[x := x^j_{\tau_{kj}}],\Delta \\ (1 \leq i \leq p) \end{array} \qquad \dfrac{}{\begin{array}{c}\sigma;\Gamma, F[x := x^j_{\tau_{kj}}] \vdash \\ F[x := x^j_{\tau_{kj}}],\Delta\end{array}}\ (Close')}{\sigma;\Gamma \vdash F[x := x^j_{\tau_{kj}}],\Delta}\ (Ind'_\tau R)$$

where we have tacitly weakened $\pi_1, \ldots, \pi_p$. Now apply $(\forall E)$ (see Lemma 3.16, this involves some additional cuts) n_k times below π_k to get a proof of:

$$\sigma;\Gamma \vdash \bigwedge_{\tau_{kj}=\tau} F[x_\tau := x^j_{\tau_{kj}}] \supset F[x_\tau := c_k(x^1,\ldots,x^{n_k})],\Delta$$

Apply $(\supset E)$ (see Lemma 3.16, this again involves some new cuts) with the conclusions of the proofs π'_j, for all j such that $\tau_{kj} = \tau$, to get a proof of:

$$\sigma;\Gamma \vdash F[x_\tau := c_k(x^1,\ldots,x^{n_k})],\Delta$$

Then cut the latter with the conclusion of π_0 to get a proof of $\sigma;\Gamma \vdash \Delta$. We let the reader design the symmetrical redex and reduction rule involving $(Ind'_\tau L)$.

If the reduction process combined with cut elimination terminates (weakly), which we conjecture, then we can restrict the term t in $(Ind'_\tau R)$, resp. $(Ind'_\tau L)$, to be a variable y_τ. Combined with the trick consisting in instantiating all variables y_τ that are not used as eigen-variables in $(\forall R)$ or $(\exists L)$ to ground terms, this means that we may assume y_τ in the induction rules to be a universal variable used below the induction rule as eigen-variable. In short, provided the conjecture holds, the tableau induction rules are:

$$\begin{array}{c} \vec{x}\ ; K\ ; E\ ; S \\ \hline \underbrace{\begin{array}{r} \vec{x}\ ; K\ ; E\ ; S, -\forall x^1_{\tau_{i1}},\ldots,x^{n_i}_{\tau_{in_i}} \cdot \bigwedge_{\tau_{ij}=\tau} F[x_\tau := x^j_{\tau_{ij}}] \\ \supset F[x_\tau := c_i(x^1,\ldots,x^{n_i})] \end{array}}_{(1\leq i \leq p)} \;\Bigg|\; \vec{x}\ ; K\ ; E\ ; S, +F[x := y_\tau] \end{array}$$

where $y \in \vec{x}$ and has the same type as x_τ. (Similarly for the other induction rule.) So, although we have to invent the formula F (this is induction loading), we don't have to invent the term t of rule $(Ind'_\tau R)$: it must be one of the variables in $\vec{x}$.

So $\mathrm{LKc}_{\approx}^{ind'}$ is not practical, unless we have heuristics to invent the formula F. We claim that using $\mathrm{LKc}_{\approx}^{ind}$ without (Cut) is enough for most simple inductive proofs, so that we can be content with the following tableau rules:

$$\frac{\vec{x}\,;K\,;E\,;S,-\forall x_\tau\cdot F}{\underbrace{\begin{array}{r}\vec{x}\,;K\,;E\,;S,-\forall x^1_{\tau_{i1}},\ldots,x^{n_i}_{\tau_{in_i}}\cdot\bigwedge\limits_{\tau_{ij}=\tau}F[x_\tau:=x^j_{\tau_{ij}}]\\ \supset F[x_\tau:=c_i(x^1,\ldots,x^{n_i})]\end{array}}_{(1\leq i\leq p)}}$$

and:

$$\frac{\vec{x}\,;K\,;E\,;S,+\exists x_\tau\cdot F}{\underbrace{\begin{array}{r}\vec{x}\,;K\,;E\,;S,+\exists x^1_{\tau_{i1}},\ldots,x^{n_i}_{\tau_{in_i}}\cdot\bigwedge\limits_{\tau_{ij}=\tau}\neg F[x_\tau:=x^j_{\tau_{ij}}]\\ \wedge F[x_\tau:=c_i(x^1,\ldots,x^{n_i})]\end{array}}_{(1\leq i\leq p)}}$$

where τ is an inductive type. At least, sticking to this restriction for inductive rules does not make the constraint solving problem (see Section 4.1) more complicated than it was already in LKc$_\approx$.

6 Related Works

First, it appears that reducing equality to syntactic equality as we did is not a new idea. Jean-Yves Girard ([Gir92], Section 3.1) mentioned a similar trick in the framework of linear logic and proposed the following rules:

$$\frac{}{\vdash t\approx t}\qquad\frac{mgu(t\approx u)=\bot}{\vdash\Gamma,\neg(t\approx u)}\qquad\frac{\vdash\Gamma\sigma\quad(\sigma=mgu(t\approx u))}{\vdash\Gamma,\neg(t\approx u)}$$

This is very close to our rules $(Refl)$, $(\approx L)/(\bot L)$ and $(\approx L)$ with some immediate instantiation going on, respectively. Again, this forces an encoding of non-constructor functions as predicates, as we did. Our import is then, apart from a rigorous account of the idea above, the notion of functionalities of predicates, and the associated $(\approx\uparrow)$ rule, plus the design of a corresponding tableau system and its accompanying constraint solving problems.

The idea that values of datatypes with free constructors can be equated with terms, that semantical equality on datatypes is syntactical equality, and that consequently left equality rules—or inversion principles, as they are called in type theory—are first-order unification, is also an idea that Conor McBride has been defending for some time now [McB96].

Encoding functions as predicates is of course a very old idea, and the Principia [WR27] is already based on a formalization of logic without function symbols. Parikh [Par73] warns against it, arguing that this may increase proof length greatly: reflexivity proofs (of $s\approx s$, where s contains non-constructor function symbols) may take $O(|s|)$ proof steps to derive. However, Girard [Gir92] argues in favor of it on esthetic grounds, while Baumgartner [Bau] shows that a similar technique, based on translating clauses with equality to Horn clauses without equality, gives good results in practice, while implementations remain simple enough. It does not seem that Baumgartner realized that constructors were easily dealt with by predicative translations, but to be fair, this was definitely not his goal either.

As far as induction is concerned, Baaz, Egly and Fermüller [BEF97] have proposed another practical scheme for induction proofs. As ours, their scheme, once translated to sequent format, does not admit cut elimination. However, their system is limited to do induction on atomic formulas (including equations), while ours is limited to bodies of universal quantifications (or negated existential quantifications) and structural induction. Naturally, we are free to code any other induction scheme, say with respect to any given well-founded ordering on terms, or any provably well-founded ordering on values. Our point here is that structural induction schemes are particularly natural in the context of theories of constructors. The main advantage of [BEF97] is the fact that inductive reasoning is done lazily, and is triggered by the closing rules of tableaux. (This involves considering Skolem functions on inductive types as choosing a minimal term in the considered well-founded ordering, and extending the closing rules accordingly.) Our way of dealing with induction is more primitive: the basic way is to trigger an induction rule to prove, say, $S, -\forall x_\tau \cdot F$, when the δ rule on $-\forall x_\tau \cdot F$ has failed to produce a proof in a given time (or Herbrand multiplicity) bound. We did not follow the path of skolemization as in [BEF97], simply because skolemization is unsound in our calculus. This needs further work, all the more so as, in the absence of skolemization, we may have to backtrack on the application of γ rules, because of the impermutabilities between quantifier rules; we can limit this by expanding α, β and δ rules eagerly, but we shall still need to backtrack on branches $\vec{x}\,; K\,; E\,; S$ where S contains no α, β, or δ formula, but contains at least two γ formulas.

7 Conclusion

We have presented a sound and complete system for first-order logic with equality and a theory of constructors. The latter is a natural theory of datatypes in programming languages like ML, and accommodates rather easily notions of proofs by structural induction. This paper can then be seen as an attempt to automatize, at least partially, proofs involving equality and induction in modern theorem proving assistants like Coq [BBC+97]. The calculus $\mathrm{LKc}_\approx$ is not only sound and complete, but cuts can even be eliminated, allowing for practical proof search. The only negative point is the complexity of constraint solving. We believe that practical cases are in fact easy to deal with, as is witnessed by the facts that constraint solving is polynomial-time decidable in the case of formulas that are essentially Prolog programs with the equality predicate (denoting sound unification); and that $\mathrm{LKc}_\approx$ without constructors has a decidable constraint solving problem. The latter makes $\mathrm{LKc}_\approx$ (without cut) the first—as far as we know—sound and complete tableau system for first-order logic with equality that has a decidable closing problem, contrary notably to equational matings and simultaneous rigid E-unification.

There are several directions in which the present work may be extended. First, it would be interesting to give a proof-term language for $\mathrm{LKc}_\approx$ and $\mathrm{LKc}_\approx^{ind}$ proofs, with a decidable type-checking problem: we believe that there is a rather simple one, based on typed λ-calculus, and where equality proofs would be computationally irrelevant, so that proof-terms would be particularly compact. This will be the subject of a future paper. Second, since constraint solving already involves some form of second-order unification, it is natural to extend $\mathrm{LKc}_\approx$ to higher-order reasoning: constraint solving should not be much more complicated than higher-order unification [Hue75], however the lack of the subformula property makes proof search more complicated (see, e.g. [Koh95]). To avoid the latter problem, we can however already consider a first-order theory of higher-order functions, where the languages of terms, but not that of formulae, is extended to include typed λ-abstraction and application, modulo $\beta\eta$-equivalence. Third, and more realistically, it would be useful to extend $\mathrm{LKc}_\approx$ to the case of non-free constructors, e.g. defining $\mathbb{Z}$ with the constructors 0, -1, $x \mapsto 2.x$ and $x \mapsto 2.x + 1$, submitted to the equations $2.0 = 0$, $2.(-1) + 1 = -1$. This can be done by describing datatypes with non-free constructors with the help of bottom-up tree automata [JB97]; but then $\mathrm{LKc}_\approx$ has to be modified, and the computation of mgus in rule $(\approx L)$ has to be replaced by something more complicated. Similar considerations apply to constructors obeying algebraic laws, say, associativity and commutativity. Fourth, we have considered free interpretations where in particular a term t with x free in t, $x \neq t$, can never denote the same value as x. Although this is well-suited to modeling common data structures like natural integers, finite lists, finite trees, etc., this cannot be used to model infinite data structures like streams. It might then be interesting to allow certain sorts to have interpretations where fixpoint equations like $x \approx t$, x free in t, $x \neq t$, have solutions; regular term unification then seems the prime candidate to replace ordinary first-order unification. Fifth, and finally, our undecidability proof for constraint solving depends on having at least one binary constructor. So it does not apply to the case of arithmetic, where we only have the constant `0` and the unary constructor `S`. In the case that constraint solving restricted to these constructors were decidable, then Kreisel's problem for $\mathrm{LKc}_\approx$ (the decidability of the existence of a proof of a given formula of size less that n, given as input) would be answered positively. The case of $\mathrm{LKc}_\approx^{ind}$, and in particular of $\mathbf{PA}_1$, would remain open. We however conjecture that constraint solving, even in this restricted case, remains undecidable.

References

[Ami90] Gilles Amiot. The undecidability of the second order predicate unification problem. *Archive for Mathematical Logic*, 30:193–199, 1990.

[Bau] Peter Baumgartner. A predicative encoding of equality. Slides given at a talk at the Meeting of the Implementation Group within the DFG Schwerpunkt "Deduktion", available at `http://www.uni-koblenz.de/~peter/DFG-Impl-Equality.ps.gz`.

[BBC+97] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, Gérard Huet, César Muñoz, Chet Murthy, Catherine Parent, Christine Paulin, Amokrane Saïbi, and Benjamin Werner. The Coq proof assistant reference manual – version V6.1. Technical Report 0203, INRIA, August 1997.

[BEF97] Matthias Baaz, Uwe Egly, and Christian Fermüller. Lean induction principles for Tableaux. In Didier Galmiche, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *LNAI*, pages 62–75. Springer, May 1997.

[BHS93] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. The even more liberalized δ-rule in free variable semantic tableaux. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *3rd Kurt Gödel Colloquium*, volume 713 of *LNCS*. Springer Verlag, 1993.

[Bol96] Dominique Bolignano. Formal verification of cryptographic protocols. In *Proceedings of the third ACM Conference on Computer and Communication Security*, 1996. Abstract at `http://www.dyade.fr/actions/VIP/approach-description.html`.

[CC77] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction of approximation of fixpoints. In *Fourth Annual ACM Symp. on Principles of Programming Languages*, pages 238–252, Los Angeles, January 1977.

[Com91] Hubert Comon. Disunification: a survey. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.

[Dow93] Gilles Dowek. A unification algorithm for second-order linear terms. Available at `http://pauillac.inria.fr/~dowek/linear.ps.gz`, 1993.

[DV96] Anatoli Degtyarev and Andrei Voronkov. The undecidability of simultaneous rigid E-unification. *Theoretical Computer Science*, 166(1–2):291–300, 1996.

[Fef84] Solomon Feferman, editor. *Kurt Gödel's Collected Works*. Oxford University Press, Oxford, UK, 1984.

[Gir92] Jean-Yves Girard. A fixpoint theorem in linear logic. Note posted to the linear logic mailing list (`linear@cs.stanford.edu`), available at `http://www.csl.sri.com/linear/mailing-list-traffic/www/07/mail_3.html`, February 1992.

[GL97] Jean Goubault-Larrecq. Ramified higher-order unification. In *12th Annual IEEE Symposium on Logics in Computer Science (LICS'97)*, Warsaw, Poland, July 1997.

[GLM97] Jean Goubault-Larrecq and Ian Mackie. *Proof Theory and Automated Deduction*, volume 6 of *Applied Logic Series*. Kluwer, May 1997. ISBN 0-7923-4593-2.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[HMT90] Robert Harper, Robin Milner, and Mads Tofte. *The Definition of Standard ML.* MIT Press, 1990.

[Hue75] Gérard P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[JB97] Jean-Pierre Jouannaud and Adel Bouhoula. Automata-driven automated induction. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 14–25, Warsaw,Poland, June 1997. IEEE Comp. Soc. Press.

[JK90] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. Technical report, LRI, CNRS UA 410: Al Khowarizmi, March 1990.

[Joh92] Peter T. Johnstone. *Notes on Logic and Set Theory.* Cambridge University Press, 1992.

[Jon90] Cliff B. Jones. *Systematic Software Development using VDM.* Prentice-Hall, 2nd edition, 1990.

[Koh95] Michael Kohlhase. Higher-order tableaux. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *4th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, volume 918 of *LNAI*, pages 294–309. Springer Verlag, May 1995.

[Mah88] Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science*, pages 348–357, Edinburgh, July 1988.

[McB96] Conor McBride. Inverting inductively defined relations in LEGO. In E. Giménez and C. Paulin-Mohring, editors, *Proceedings of TYPES'96*, LNCS 1512, pages 236–253. Springer-Verlag, 1996.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[MM98] Raymond McDowell and Dale Miller. Cut elimination for a logic with definitions and induction. In *TYPES'96*. Springer Verlag, 1998. Available at `ftp://ftp.cis.upenn.edu/pub/papers/miller/cut_elim.dvi.gz`.

[Par73] Rohit Parikh. Some results on the lengths of proofs. *Transactions of the American Mathematical Society*, 177:29–36, 1973.

[Pau97] Lawrence Paulson. Proving properties of security protocols by induction. In *IEEE Computer Security Foundations Workshop X*, pages 70–83, 1997.

[Plo72] Gordon Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[Rob65] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[Sti85] Mark E. Stickel. Automated deduction by theory resolution. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 1181–1186, Los Angeles, 1985.

[Str93] Martin Streicher. *Investigations into Intensional Type Theory.* Habilitationsschrift, LMU München, 1993.

[Wal88] Christoph Walther. Many-sorted unification. *Journal of the ACM*, 35(1):1–17, 1988.

[WR27] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1910, 1927.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105,
78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS
Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399