Axiomatisation of constraint systems
to specify a tableaux calculus modulo theories

Damien Rouhling\textsuperscript{3}, Mahfuza Farooque\textsuperscript{1}
Stéphane Graham-Lengrand\textsuperscript{1,4,2}, Assia Mahboubi\textsuperscript{2}, and Jean-Marc Notin\textsuperscript{1}

CNRS\textsuperscript{1}, INRIA\textsuperscript{2}, ENS Lyon\textsuperscript{3}, Ecole Polytechnique\textsuperscript{4}

\begin{abstract}
In this paper we explore a proof-theoretic approach to the combination of generic proof-search and theory-specific procedures, in presence of quantifiers. Forming the basis of \textit{tableaux} methods, the sequent calculus is refined with meta-variables (a.k.a existential variables or free variables) to delay the choice of witnesses, and parameterised by theory-specific features based on a system of \textit{constraints} for meta-variables. An axiomatisation of these features is given, so that the soundness and completeness of the sequent calculus with meta-variables can be generically proved (with respect to the sequent calculus where the choice of witnesses is not delayed). We then describe a theory-generic proof-search implementation, that is parameterised by a theory-specific module whose specification is given by the above axiomatisation.
\end{abstract}

1 Introduction

The sequent calculus is a formalism that provides a proof-theoretical basis for proof-search procedures and theorem-proving techniques, such as \textit{tableaux} methods. Other procedures can also be described as the gradual construction of proof-trees in sequent calculus, ranging from computation in ProLog and \(\lambda\)-ProLog \cite{MNPS91}, to Dowek’s proof synthesis procedure \cite{Dow93} in Pure Type Systems \cite{LDM11}. Two concepts help the description of such procedures in a somewhat generic sequent calculus framework: \textit{polarities} and focusing \cite{Gir87,Gir91,And92}. In \cite{FGLM13,Far13} a variant of the focused sequent calculus \text{LK}^p(T) \cite{LM09} for polarised classical logic was used to describe the DPLL procedure for propositional SAT problems as a simple proof-tree construction strategy characterised by an appropriate management of polarities. Furthermore, this sequent calculus was generalised into a system called \textit{LK}^p(T) to work \textit{modulo a theory} \(T\), so as to capture well-known extensions of DPLL that address SAT-modulo-theories (SMT) problems, namely \text{DPLL}(T) \cite{NOT05,NOT06}. The interaction that \textit{LK}^p(T) describes, between a theory-generic proof-search technique and a theory-specific decision procedure, was implemented in a modular prototype: the \text{PSYCHE} prover \cite{GL13,Psy}.

However, the sequent calculus “modulo a theory” \textit{LK}^p(T) does not specify how a proof-search method should effectively handle quantifiers: on paper, proving an existential formula in \textit{LK}^p(T) requires the immediate production of a witness. Correspondingly, quantifiers were not supported in \text{PSYCHE} 1.6.
On the other hand, in the case of quantifiers in the empty theory, proof-search in the sequent calculus can be effectively implemented: when proving an existential formula or refuting a universal formula, the instantiation of a variable is delayed and eventually obtained by solving unification constraints when branches of the proof-tree are completed. This can be formalised in tableaux methods by using free variables, skolemisation and unification [RV01].

This paper provides a proof-theoretical foundation for performing proof-search modulo a theory in presence of quantifiers. More precisely, we use meta-variables\(^1\) to formalise how the instantiation of existentially quantified variables is delayed. We propose an abstract presentation of constraints, which can accommodate not only unification constraints but also theory-specific ones. We turn the sequent calculus into a sequent calculus with delayed instantiations, parameterised by a theory given by a constraint system and a decision procedure satisfying a collection of axioms. These are sought to be as weak as possible, but sufficient to prove the soundness and completeness of the calculus with delayed instantiations w.r.t. the calculus without delayed instantiations.

The above methodology aims at providing proof-theoretical foundations for a large class of proof-search procedures, such as those that can be described by the focused system \(\text{LK}^p(\mathcal{T})\) of [Far13] and be implemented in Psyche. In particular, we aim at covering proof-search procedures that may backtrack: without backtracking, propagating constraints\(^2\) from one branch of a proof-tree to another one can be done by simple side-effects. A specific aspect of our contribution is to explicitly support, in the sequent calculus itself, non-destructive propagation mechanisms based on persistent data-structures, which are better adapted to backtracking. This lead to a purely functional implementation, namely that of Psyche 2.0, which we describe at the end of this paper.

For the sake of simplicity, we choose to present our proof-theoretical foundations not in the focused system \(\text{LK}^p(\mathcal{T})\) that Psyche implements, but in a much simpler sequent calculus. The focused system, whose basic version can be found in Appendix A, would add more technicalities and may obfuscate our main argument; it does, however, advocate for a treatment of constraints that is robust to backtracking.

The simple sequent calculus that we use in this paper, which we call \(\text{LK}\), uses the standard first-order notions of term, literal, eigenvariable\(^3\) and formula. It handles sequents of the form \(\Gamma \vdash_n \Gamma\), and its rules are presented in Fig. 1, where:

- \(\Gamma\) is a multiset of first-order formulae, and \(\Gamma_{\text{lit}}\) the subset of its literals;
- \(A[x := t]\) denotes the substitution of a term \(t\) for a variable \(x\) in a formula \(A\);
- \(e_n\) is the eigenvariable of index \(n\);

\(^1\) Our meta-variables plays the same role as free variables in tableaux methods.

\(^2\) or propagating the instantiation of a meta-variable

\(^3\) Eigenvariables are explicitly indexed in \(\mathbb{N}^*\), and introduced in increasing order.
\( \models \) denotes a ground validity predicate, i.e. \( \models \Gamma_{\text{lit}} \) denotes the validity in the theory of the disjunction of literals in \( \Gamma \).\(^4\)

- \( n \) can be intuitively understood as the number of eigenvariables that are already introduced, as illustrated by the rule for the universal quantifier.

\[
\begin{array}{c}
\vdash_{n} \Gamma, A \\
\vdash_{n} \Gamma, A \land B \\
\vdash_{n} \Gamma, A \lor B \\
\vdash_{n} \forall x A
\end{array}
\quad
\begin{array}{c}
\vdash_{n} \Gamma_{\text{lit}} \\
\vdash_{n} \Gamma, A[x := t], \exists x A \\
\vdash_{n} \Gamma, A \\
\vdash_{n+1} \Gamma, A[x := e_{n+1}]
\end{array}
\]

**Figure 1.** The LK sequent calculus modulo theories

In Section 2 we present abstract systems of constraints. In Section 3 we transform LK into a sequent calculus with delayed instantiations LK\(^7\); we identify the axioms with which we then prove the soundness and completeness w.r.t. LK. In Section 4 we present a variant LK\(^7\)\rangle where the treatment of branching is asymmetric, reflecting a sequential implementation of proof-search. In Section 5, we prove soundness and completeness of LK\(^7\)\rangle w.r.t. LK\(^7\). In Section 6 we briefly overview the corresponding implementation in PSYCHE 2.0 and then we conclude.

## 2 Meta-variables and constraint systems

In this section we introduce and discuss the notions of meta-variables and of constraint structures used in the calculi presented in section 3 and 4.

A realistic proof-search procedure is in general unable to provide an appropriate witness "out of the blue" at the time when of applying an existential rule: this may require extra information from the underlying theory. In system LK, theory-dependent information is available only at the leaves of the proof tree, when calling the ground validity predicate in the axiom rule. Therefore, we propose to postpone the instantiation of existential witnesses. This requires extending the syntax of the formulae in the sequents with meta-variables, which are place-holders for yet-to-come instantiation. Instead of using Skolem function symbols, we make use of a detached data-structure, called domain, which records, along a branch of a proof-tree, the dependency of meta-variables on eigenvariables.

In this paper, a meta-variable is indexed by a natural number \( n \in \mathbb{N}^* \) and denoted \( \_n \). We call \textit{arity} of a meta-variable the number of eigenvariables it can depend on.\(^5\) When a meta-variable is introduced above another one in the same branch of the proof-tree, the arity of the former is greater than the arity

\(^{4}\) In other words, it denotes the inconsistency of the conjunction of their negations, as provided for instance by the decision procedures used in SMT-solving.

\(^{5}\) This determines which eigenvariables can be used to instantiate the meta-variable.
of the latter: hence, the arities of the meta-variables that exist at one node of
the proof-tree can be represented, as a whole, by a list of decreasing natural
numbers, whose head is the arity of the meta-variable that has been created
last. We use standard notations for operations on lists: \([\ ]\) denotes the empty
list and \(n::l\) denotes the list obtained by consing a list \(l\) with \(n \in \mathbb{N}\). The
concatenation of two lists \(l_1\) and \(l_2\) is denoted \(l_1@l_2\) and the length of a list \(l\)
is denoted \(|l|\). We say that a list \(l_2\) extends a list \(l_1\) if \(l_2 = l@l_1\) for some list \(l_1\).

**Definition 1 (Domain).** A domain is a list of decreasing natural numbers.

**Definition 2 (Terms, formulae, contexts with meta-variables).** A term
(resp. formula, context) of domain \(l\) is a term (resp. formula, context) with
meta-variables among \(?_1, \ldots, ?_{|l|}\) (besides bound variables and eigenvariables).

The arity of the meta-variable \(?_n\) in domain \(l\) is the \((|l| - n + 1)\)th
element of \(l\). A term of domain \([\ ]\) (i.e. without meta-variable) is called a
ground term.

In this setting, the axiom rule of system LK has to be adapted to the
presence of meta-variables in literals. But even at this point we want to avoid
blindly committing to an instantiation, since some meta-variables can be shared
by several branches. Instead, we propose to generate a constraint on (yet-to-
come) instantiations, whose satisfaction is sufficient to close the branch. When
possible, generating a necessary and sufficient constraint would be the best op-
tion, but we do not enforce this. We then propagate those constraints between
branches and combine them, backtracking whenever the carried constraint be-
comes unsatisfiable, but hopefully closing all branches.

An abstract structure of constraints is therefore formalized as follows:

**Definition 3 (Constraint structures).** A constraint structure is:

- a family of sets \((\Psi_l)_l\), indexed by domains; elements of \(\Psi_l\) are called
  constraints of domain \(l\), and are denoted \(\sigma, \sigma'\), etc.

- together with a family of mappings called projections from \(\Psi_{n::l}\) to \(\Psi_l\) for every
  \(n\) and \(l\), mapping a constraint \(\sigma \in \Psi_{n::l}\) to a constraint \(\sigma_{\downarrow} \in \Psi_l\).

  A meet constraint structure is a constraint structure \((\Psi_l)_l\) with a binary
  operator \((\sigma, \sigma') \mapsto \sigma \land \sigma'\) on each set \(\Psi_l\).

  A lift constraint structure is a constraint structure \((\Psi_l)_l\) with a map \(\sigma \mapsto \sigma_{\uparrow}\)
  from \(\Psi_l\) to \(\Psi_{n::l}\) for every \(l\) and \(n\).

Each mapping from \(\Psi_{n::l}\) to \(\Psi_l\) extracts a constraint concerning the meta-
variables \(?_1, \ldots, ?_{|l|}\) from a constraint concerning \(?_1, \ldots, ?_{(|l|+1)}\). Distinct the-
tories lead to possibly different constraint structures:

**Example 1 (Empty theory).** In the case of the empty theory, we can define a
constraint structure for unification by taking as \(\Psi_l\) the set of maps assigning,

\(^6\) Any term \(u\) (resp. formula, context) on a domain \(d_1\) also has domain \(d_2\) if \(d_2\) extends \(d_1\).
to every meta-variable \( ?_n \) with \( n \leq |l| \), a ground term whose eigenvariables have their indexes all below the arity of \( ?_n \) in \( l \). Projecting a map in \( \Psi_{n:l} \) is just erasing the entry assigned to \( ?_{|l|+1} \). The most general unifier provides a meet operator on this structure. The lift of a map \( \sigma \in \Psi_l \) in any \( \Psi_{n:l} \) is the extension of \( \sigma \) that maps \( ?_{|l|+1} \) to itself.

**Example 2 (Quantifier elimination).** We can also define a constraint structure for a theory that admits quantifier elimination (e.g. linear real arithmetic). We take \( \Psi_l \) to be the set of ground quantifier-free formulae without eigenvariables and \( \Psi_{n:l} \) to be the set of quantifier-free formulae with domain \( n :: l \) and whose eigenvariables are among \( e_1 \ldots e_n \). The projection is provided by quantifier elimination, the meet operator is the conjunction and the lift is the identity.

### 3 A system for proof-search with constraints

In the rest of this section \( (\Psi_l)_l \) denotes a fixed meet constraint structure.

#### 3.1 The sequent calculus with delayed instantiations \( \text{LK}^? \)

In order to formalise the delayed instantiations of meta-variables, and the propagation of constraints, we generalise the ground validity predicate in parameter of the \( \text{LK} \) calculus to a constraint-producing predicate.

**Definition 4 (\( \text{LK}^? \) sequent calculus).**

A constraint-producing predicate is a family of relations \( (\models_l)_{l} \), indexed by domains \( l \), relating sets \( A \) of literals of domain \( l \) with constraints \( \sigma \) in \( \Psi_l \); when it holds, we write \( \models_l A \rightarrow \sigma \).

Given such a predicate \( (\models_l)_{l} \), the sequent calculus with delayed instantiations \( \text{LK}^? \) manipulates sequents of the form \( \vdash_l \Gamma \rightarrow \sigma \), where \( \Gamma \) is a context and \( \sigma \) is a constraint, both of domain \( l \). Its rules are presented in Fig. 2.

![Figure 2. The sequent calculus with delayed instantiations \( \text{LK}^? \)](image)

In terms of process, a sequent \( \vdash_l \Gamma \rightarrow \sigma \) displays the inputs \( \Gamma, n, l \) and the output \( \sigma \) of proof-search, which would start building a proof-tree, in system \( \text{LK}^? \), from the root. The sequent at the root of the tree would typically be of the form \( \vdash_0 \Gamma \rightarrow \sigma \), with \( \sigma \in \Psi_0 \) to be produced as an output. Given
that the empty domain corresponds to the absence of meta-variables, a typical example for $\Psi[l]$ is the set of booleans (a constraint of $\Psi[l]$ simply being either the satisfiable constraint or the unsatisfiable constraint). The constraints are produced at the leaves, and propagated back down towards the root.

### 3.2 Instantiations and compatibility with constraints

Notice that, in system $\text{LK}^2$, no instantiation for meta-variables is actually ever produced. Instantiations would only come up when reconstructing, from an $\text{LK}^2$ proof, a proof in the original calculus $\text{LK}$. For this we need to formalise how constraints relate to actual instantiations, or more precisely, what it means for an instantiation to satisfy, or be compatible with, a constraint. The instantiations that are compatible with a constraint of domain $l$ should provide, for each meta-variable, a term that at least respects the eigenvariable dependencies specified in $l$, as formalised in definition 5. Beyond this, what it means for an instantiation to be compatible with a constraint is specific to the theory and we simply identify in definition 6 some minimal axioms.

**Definition 5 (Instantiation and substitution).** For $n \in \mathbb{N}$, let $T_n$ be the set of ground terms whose eigenvariables have indexes below $n$.

We extend this notation to domains pointwisely: let $T[[l]] := \{[]\}$ and $T_{n; l} := \{t :: \rho | t \in T_n \text{ and } \rho \in T_l\}$. An instantiation on domain $l$ is an element of $T_l$.

For a term $t$ (resp. a formula $A$, a context $\Gamma$) of domain $l$ and an instantiation $\rho \in T_l$, we denote by $\rho(t)$ (resp. $\rho(A)$, $\rho(\Gamma)$) the result of substituting in $t$ (resp. $A$, $\Gamma$) each meta-variable $?_i$ by the $(|l| - i + 1)$th element of $\rho$.

**Definition 6 (Compatibility relation).** A compatibility relation is a (family of) relation(s) between instantiations $\rho \in T_l$ and constraints $\sigma \in \Psi_l$ for each domain $l$, denoted $\rho \in \sigma$, that satisfies the axiom

$$B1 \quad \forall t, \rho, (t :: \rho) \in \sigma \Rightarrow \rho \in \sigma$$

If the constraint structure is a meet constraint structure, we say that the compatibility relation distributes over $\land$ if

$$C1 \quad \forall \rho, \sigma, \sigma', \begin{cases} \rho \in \sigma \Rightarrow \rho \in \sigma' \\ \rho \in \sigma \end{cases} \Leftrightarrow \rho \in (\sigma \land \sigma')$$

Note that a meet constraint structure can be defined by taking constraints to be sets of instantiations themselves: the $\in$ relation is just set membership, set intersection provides a meet operator and projections can be easily defined.

**Example 3.** For the constraint structure defined in Example 1 for unification, we can take compatibility to be: $\rho \in \sigma$ if $\rho$ is a (ground) instance of substitution $\sigma$. For the constraint structure defined in Example 2 for theories with quantifier elimination, we can take: $\rho \in F$ if the ground formula $\rho(F)$ is valid in the theory.
Another ingredient we need for relate the two sequent calculi is a mechanism for producing instantiations. We formalise a binding operator which maps every constraint of $\Psi_{n:l}$ to a function, which outputs an “appropriate” instantiation for the $(|l|+1)$-th meta-variable when given as input an instantiation for the first $|l|$ variables. What “appropriate” means is defined by the following axiom.

**Definition 7 (Binding).** A binding operator for a compatibility relation $\epsilon$ is a (family of) function(s) that maps every $\sigma \in \Psi_{n:l}$ to $f_{\sigma} \in (T_{l} \rightarrow T_{n})$, for every $n$ and $l$, and that satisfies the following axiom:

$$\forall \rho \epsilon \sigma_{\downarrow}, (f_{\sigma}(\rho) :: \rho) \epsilon \sigma$$

**Example 4 (Unification).** The constraint structure from Example 1 provides a natural binding operator: for $\sigma \in \Psi_{n:l}$, $f_{\sigma}(\rho)$ is the ground term $\rho(\sigma(?_{|l|+1}))$.

**Example 5 (Theories with quantifier elimination).** In the case of Example 2, from a formula $F \in \Psi_{n:l}$ and an instantiation $\rho$, the binding operator should compute an existential witness $f_{F}(\rho)$ for the formula $\rho(F)$, which features a single meta-variable. In the general case, we might need to resort to a Hilbert-style choice operator to construct the witness. For instance in linear arithmetic, $f_{F}(\rho)$ denotes a solution of a disjunction of systems of linear equations: denoting the witness requires extending the syntax of terms. Note however that proof-search in LK? does not require implementing such a binding operator.

Interestingly enough, such a binding operator elects a canonical instantiations among all of those that are compatible with a given constraint, by iterating projections down to the empty domain as follows:

**Definition 8 (Folding a constraint).** Given a binding operator, the fold of a constraint $\sigma \in \Psi_{l}$ is an instantiation of $T_{l}$ defined by induction on $l$:

$$\text{fold} \sigma := \left[ \begin{array}{ll} \text{fold} & \text{if } \sigma \in \Psi_{|l|} \\ f_{\sigma}(\text{fold}(\sigma_{\downarrow})) & \text{if } \sigma \in \Psi_{n:l} \end{array} \right.$$  

**Remark 1.** The main information about a constraint of empty domain $\Psi_{[]}$ is whether $[]$, the unique instantiation in $T_{[]}$, is compatible with it or not.

For every $l$ and every $\sigma \in \Psi_{l}$, there exists $\rho$ such that $\rho \epsilon \sigma$ if and only if $\text{fold}(\sigma) \epsilon \sigma$ (the left-to-right direction is proved by induction on $l$ using axioms B1 and B2). When this is the case, we say that $\sigma$ is satisfiable.

We also need a relation between the two nature of predicates: intuitively, the instantiations that turn a set $A$ of literals of domain $l$ into a valid set of (ground) literals should coincide with the instantiations that are compatible with some constraint produced for $A$. This is expressed by the following axiom:

**Definition 9 (Relating predicates).** For a compatibility relation $\epsilon$, we say that a constraint-producing predicate $(=_{l})$ relates to a ground validity predicate $|=_{l}$, for all domains $l$ and all sets $A$ of literals of domain $l$,
A \{ \rho \mid \models \rho(A) \} = \bigcup \{ \rho \mid \rho \in \sigma \} \{ \sigma \mid \models \lambda \to \sigma \} \}

There may be several constraints (finitely or infinitely many) allowing a leaf to be closed. In practice this means we expect a stream of constraints to be produced at the leaves, corresponding to the (possibly infinite) union in axiom A: each one of them is sufficient to close the branch; we take the first one and if we later find out that we need a different one, we can backtrack and try the next one in the stream.

3.3 Soundness and completeness

We now prove the equivalence between System $\text{LK}^\|$ and System $\text{LK}$, from the axioms A, B1, B2, C1. In other words we assume that we have a compatibility relation that distributes over $\land$, equipped with a binding operator, plus a constraint-producing predicate $(\models^\|)$ related to a ground validity predicate $\models$.

**Theorem 1 (Soundness and completeness of $\text{LK}^\|$).**

- If $\vdash^\| \Gamma \to \sigma$ is derivable in $\text{LK}^\|$, then for all $\rho \in \sigma$, $\vdash \rho(\Gamma)$ is derivable in $\text{LK}$.
- For all domains $l$, for all contexts $\Gamma$ of domain $l$, for all $\rho \in T_l$, if $\vdash^\| \rho(\Gamma)$ is derivable in $\text{LK}$, then there exists $\sigma \in \Psi_l$ such that $\vdash^\| \rho(\Gamma) \to \sigma$ is derivable in $\text{LK}^\|$ and $\rho \in \sigma$.

**Proof.** See the proofs in the appendix B.

This can be directly applied to the canonical instantiation given by fold($\sigma$):

**Corollary 1 (Soundness with canonical instantiation).**

If $\vdash^\| \Gamma \to \sigma$ is derivable in $\text{LK}^\|$, with $\sigma$ being satisfiable, then $\vdash \text{fold}(\sigma)(\Gamma)$ is derivable in $\text{LK}$.

Since there is no eigenvariables nor meta-variables at the root of a closed proof-tree, the root is of the form $\vdash^\|_0 \Gamma \to \sigma$ for some constraint $\sigma \in \Psi_l$. In the particular case of the domain $\lambda$, remember that the set $T^\|_l$ is the singleton $\{\lambda\}$ and notice that if $\Gamma$ is a context of domain $\lambda$, we have $\lambda(\Gamma) = \Gamma$. Therefore, soundness and completeness can be rewritten as follows:

**Corollary 2 (Soundness and completeness on the empty domain).**

There exists $\sigma \in \Psi^\|$ such that $\vdash^\|_l \Gamma \to \sigma$ is derivable in $\text{LK}^\|$ and $\lambda \epsilon \sigma$, if and only if $\vdash^\|_l \Gamma$ is derivable in $\text{LK}$.

4 Sequentialising

The soundness and completeness properties of System $\text{LK}^\|$ rely on constraints that are satisfiable. A proof-search process based on it should therefore not
proceed any further with a constraint that has become unsatisfiable. Since
the meet of two satisfiable constraints may be unsatisfiable, exploring the two
branches created by the introduction rule for conjunction may take advantage
of a sequential treatment: a constraint produced to close one branch may direct
the exploration of the other branch, which may be more efficient than waiting
until both branches have independently produced constraints and only then
checking that their meet is still satisfiable. This section develops a variant of
System $\text{LK}^2$ to support this sequentialisation of branches.

In the rest of this section $(\Psi_l)_l$ is a fixed lift constraint structure.

4.1 Definition of the proof system

Thus, the proof rules enrich a sequent with two constraints: the input one and
the output one, the latter being “stronger” than the former, in a sense that we
will make precise when we relate the different systems. At the leaves, a new
predicate $(|=\langle_l)_l$ is used that also takes the input substitution as an extra input.

Definition 10 ($\text{LK}^\langle_l$ sequent calculus).

A constraint-refining predicate is a family of relations $(|=\langle_l)_l$, indexed by
domains $l$, relating sets $A$ of literals of domain $l$ with pairs of constraints $\sigma$
and $\sigma'$ in $\Psi_l$; when it holds, we write $\sigma\rightarrow|=\langle_l A\to\sigma'$.

Given such a predicate $(|=\langle_l)_l$, the sequent calculus with sequential delayed
instantiation, denoted $\text{LK}^\langle_l$, manipulates sequents of the form $\sigma\rightarrow|=\langle_l \Gamma\to\sigma'$,
where $\Gamma$ is a context and $\sigma$ and $\sigma'$ are constraints, all of domain $l$. Its rules
are presented in Fig. 3.

\begin{align*}
\sigma\rightarrow|=\langle_l \Gamma \rightarrow \sigma' & \quad \sigma\rightarrow|=\langle_l \Gamma, A \rightarrow \sigma', \exists x A \rightarrow \sigma'' \quad \sigma\rightarrow|=\langle_l \Gamma, \exists x A \rightarrow \sigma''
\end{align*}

$\Gamma\rightarrow\vdash=\langle_l _l$ $\rightarrow\vdash=\langle_l _l$

Figure 3. The sequent calculus with sequential delayed instantiation $\text{LK}^\langle_l$

The branching rule introducing conjunctions allows an arbitrary sequential-
isation of the branches when building a proof-tree, proving $A_0$ first if $i = 0$,
or proving $A_1$ first if $i = 1$.

4.2 Soundness and completeness

We now relate system $\text{LK}^\langle_l$ to system $\text{LK}^2$. For this we need some axioms about
the notions used in each of the two systems. These are distinct from the axioms
that we used to relate system LK^2 to LK, since we are not (yet) trying to relate system LK^3 to LK. In the next section however, we will combine the two steps.

**Definition 11 (Decency).** When \( \preceq \) is a family of pre-orders over each \( \Psi_l \), \( \land \) is a family of binary operators over each \( \Psi_l \), and \( P \) is a family of predicates over each \( \Psi_l \), we say that \( (\preceq, \land, P) \) is decent if the following axioms hold:

- **D1** \( \forall \sigma, \sigma' \in \Psi_l, \sigma \land \sigma' \) is a greatest lower bound of \( \sigma \) and \( \sigma' \) for \( \preceq \)
- **D2** \( \forall \sigma \in \Psi_l, \forall \sigma', \sigma'' \in \Psi_{n:l}, \sigma'' \preceq \sigma' \land \sigma' \Rightarrow \sigma'' \preceq \sigma \land \sigma' \)
- **P1** \( \forall \sigma \in \Psi_{n:l}, P(\sigma) \iff P(\sigma_l) \)
- **P2** \( \forall \sigma, \sigma' \in \Psi_l, \begin{cases} P(\sigma) \\ \sigma \preceq \sigma' \end{cases} \Rightarrow P(\sigma') \)

where \( \preceq \) denotes the equivalence relation generated by \( \preceq \).

Notice that this makes \( (\Psi_l/\preceq, \land) \) a meet-semilattice that could equally be defined by the associativity, commutativity, and idempotency of \( \land \).

**Definition 12 (Relating constraint-producing/refining predicates).**

Given a family of binary operators \( \land \) and a family of predicates \( P \), we say that a constraint-refining predicate \( (\models^l) \) relates to a constraint-producing predicate \( (\models^l) \) if, for all domains \( l \), for all sets \( A \) of literals of domain \( l \) and for all \( \sigma \in \Psi_l \),

- **A1** \( \forall \sigma' \in \Psi_l, \sigma \models^l A \rightarrow \sigma' \Rightarrow \exists \sigma'' \in \Psi_l, \begin{cases} \sigma' \preceq \sigma \land \sigma'' \\ P(\sigma \land \sigma'') \\ \models^l A \rightarrow \sigma'' \end{cases} \)
- **A2** \( \forall \sigma' \in \Psi_l, \begin{cases} P(\sigma \land \sigma') \\ \sigma \models^l A \rightarrow \sigma' \end{cases} \Rightarrow \exists \sigma'' \in \Psi_l, \begin{cases} \sigma'' \preceq \sigma \land \sigma' \\ \sigma \rightarrow \models^l A \rightarrow \sigma'' \end{cases} \)

In the rest of this sub-section, we assume that we have a decent triple \( (\preceq, \land, P) \), and a constraint-refining predicate \( (\models^l) \) that relates to a constraint-producing predicate \( (\models^l) \).

**Theorem 2 (Soundness of LK^3).**

If \( \sigma \models^l_{\models^3} \Gamma \rightarrow \sigma' \) is derivable in LK^3, then there exists \( \sigma'' \in \Psi_l \) such that \( \sigma' \preceq \sigma \land \sigma'' \), \( P(\sigma \land \sigma'') \) and \( \models^l_{\models^3} \Gamma \rightarrow \sigma'' \) is derivable in LK^3.

**Proof.** See the proof in the appendix B.

Notice that the statement for soundness of LK^3 w.r.t. LK^2 is merely a generalisation of axiom A1 where the reference to \( \models^l \) and \( \models^l \) have respectively been replaced by derivability in LK^2 and LK^3. A natural statement for completeness of LK^3 w.r.t. LK^2 would be the symmetric generalisation of axiom A2:

If \( \models^l_{\models^3} \Gamma \rightarrow \sigma' \) is derivable in LK^3, then for all \( \sigma \in \Psi_l \) such that \( P(\sigma \land \sigma') \), there exists \( \sigma'' \in \Psi_l \) such that \( \sigma'' \preceq \sigma \land \sigma' \) and \( \models^l_{\models^3} \Gamma \rightarrow \sigma'' \) is derivable in LK^3.

This statement can be proved, but it fails to capture an important aspect of system LK^3: choosing, in a branching rule, which branch is on the left and
which one is on the right (i.e. which branch the root-first proof-search procedure treats first), should not matter for completeness. Indeed, the above statement for completeness entails that there exists a sequentialisation of branches that leads to a complete proof-tree in $\mathcal{L}K\rangle$, but the proof-search procedure should either guess it or investigate all possibilities. So what we really want for completeness is a stronger statement whereby, for all possible sequentialisations of branches, there exists a complete proof-tree. Therefore, when the proof-search procedure decides to apply the branching rule, choosing which branch to complete first can be treated as “don’t care non-determinism” rather than “don’t know non-determinism”: if a particular choice proves unsuccessful, there should be no need to explore the alternative choice.

**Definition 13 (Black and white tree).**

A black and white tree is an infinite binary tree whose nodes are labelled as either “black” or “white”.

We define the property, for a proof-tree $\pi$ of system $\mathcal{L}K\rangle$, to follow a black and white tree $r$, by induction on $\pi$:
- when the last rule of $\pi$ has no premiss, $\pi$ follows $r$;
- when the last rule of $\pi$ has one premiss, $\pi$ follows $r$ if its direct sub-proof-tree does;
- when the last rule of $\pi$ is of the form

$$
\begin{align*}
\pi_i & : \sigma \models^1_n \Gamma, A_i \rightarrow \sigma'' \\
\pi_1-i & : \sigma'' \models^1_n \Gamma, A_{1-i} \rightarrow \sigma' \\
\sigma & \models^1_n \Gamma, A_0 \land A_1 \rightarrow \sigma'
\end{align*}
$$

$\pi$ follows $r$ if
- $\pi_i$ follows the left direct sub-tree of $r$
- $\pi_{1-i}$ follows the right direct sub-tree of $r$
- either $i = 0$ and the root of $r$ is “white”, or $i = 1$ and the root of $r$ is “black”.

We can now state the strong version of completeness:

**Theorem 3 (Completeness of $\mathcal{L}K\rangle$).**

If $\sigma \rightarrow^1_n \Gamma \rightarrow \sigma'$ is derivable in $\mathcal{L}K\rangle$, then for all $\sigma \in \Psi_1$ such that $P(\sigma \land \sigma')$, and for all black and white tree $r$, there exists $\sigma'' \in \Psi_1$ such that $\sigma'' \equiv \sigma \land \sigma'$ and $\sigma \rightarrow^1_n \Gamma \rightarrow \sigma''$ is derivable in $\mathcal{L}K\rangle$ with a proof-tree that follows $r$.

**Proof.** See the proof in the appendix B.

5 Relating $\mathcal{L}K\rangle$ to LK

In this section we combine the two steps: from LK to $\mathcal{L}K\rangle$ and from $\mathcal{L}K\rangle$ to $\mathcal{L}K\rangle$, so that we can relate $\mathcal{L}K\rangle$ to LK. For this we aggregate (and consequently simplify) the axioms that we used for the first step with the axioms that we used for the second step.
Definition 14 (Compatibility-based pre-order).
Assume we have a family of compatibility relations $\mathcal{C}$ for a constraint structure $(\Psi^i)$. We define the following pre-order on each $\Psi^i$:
\[
\forall \sigma, \sigma' \in \Psi^i, \sigma \leq \mathcal{C} \sigma' \iff \{ \rho \in T_i \mid \rho \mathcal{C} \sigma \} \subseteq \{ \rho \in T_i \mid \rho \mathcal{C} \sigma' \}
\]
and let $\leq \mathcal{C}$ denote the symmetric closure of $\leq \mathcal{C}$.

We now assume that we have a lift constraint structure and a constraint-refining predicate $([ \vdash ]_l)$ used to define $\text{LK}^l$, and the existence of
- a binary operator $\land$
- a compatibility relation $\mathcal{C}$ that distributes over $\land$ (B1 and C1 in Fig. 4)
- a binding operator for $\mathcal{C}$ (B2 in Fig. 4)
- a constraint-producing predicate $([ = ]_l)$ that relates to $=$ (A in Fig. 4)
- a predicate $P$
satisfying the axioms of Fig. 4. These entail decency (see Appendix B):

Lemma 1. Given the axioms of Fig. 4, $(\leq \mathcal{C}, \land, P)$ is decent.

| B1 | $\forall (t : \rho) \mathcal{C} \sigma, \rho \mathcal{C} \sigma_1$ | C4 | $\forall \sigma, \sigma', \rho, (f_{\sigma'}(\rho) : \rho) \mathcal{C} \sigma' \iff \rho \mathcal{C} \sigma$ |
| B2 | $\forall \rho \mathcal{C} \sigma_1, (f_{\sigma}(\rho) : \rho) \mathcal{C} \rho$ | P1 | $\forall \sigma \in \Psi_{n+1}, P(\sigma) \iff P(\sigma_l)$ |
| C1 | $\forall \rho, \sigma, \sigma', \rho \mathcal{C} \sigma' \iff \rho \mathcal{C} (\sigma \land \sigma')$ | P2 | $\forall \rho, \sigma' \in \Psi_l, \{ \sigma \leq \mathcal{C} \sigma' \Rightarrow P(\sigma') \}$ |
| A  | $\forall l, \forall A, \{ \rho \mid \vdash (A) \} = \bigcup \{ \rho \mid \rho \mathcal{C} \sigma \}$ |         |         |
| A1 | $\forall l, \forall A, \forall \sigma, \sigma' \in \Psi_l, \sigma \vdash [\vdash ]_l \vdash A \rightarrow \sigma' \Rightarrow \exists \sigma'' \in \Psi_l, \{ \forall \sigma \leq \mathcal{C} \sigma'' \Rightarrow P(\sigma \land \sigma'') \}$ |
| A2 | $\forall l, \forall A, \forall \sigma, \sigma' \in \Psi_l, \{ P(\sigma \land \sigma') \vdash [\vdash ]_l \vdash A \rightarrow \sigma' \Rightarrow \exists \sigma'' \in \Psi_l, \{ \forall \sigma \leq \mathcal{C} \sigma'' \Rightarrow P(\sigma \land \sigma') \}$ |

Figure 4. Full axiomatisation for $\text{LK}^l$

Hence, we have soundness and completeness of $\text{LK}^l$ w.r.t. LK on the empty domain, as a straightforward consequence of Corollary 2 and Theorems 2 and 3:

Theorem 4 (Soundness and completeness on the empty domain).
- If $\sigma \vdash [\vdash ]_l \Gamma \rightarrow \sigma'$ is derivable in $\text{LK}^l$ and $[\vdash ]_l \mathcal{C} \sigma'$, then $\vdash_n \Gamma$ is derivable in LK.
  In particular when $P$ is the predicate “being satisfiable”, if $\sigma \vdash [\vdash ]_l \Gamma \rightarrow \sigma'$ is derivable in $\text{LK}^l$, then $\vdash_n \Gamma$ is derivable in LK.
- Assume $P$ is always true or is “being satisfiable”.
  If $\vdash_n \Gamma$ is derivable in LK, then for all $\sigma \in \Psi_l$ such that $[\vdash ]_l \mathcal{C} \sigma$ and for all black and white tree $r$, there exists $\sigma' \in \Psi_l$ such that $[\vdash ]_l \mathcal{C} \sigma'$ and $\sigma \vdash [\vdash ]_l \Gamma \rightarrow \sigma'$ is derivable in $\text{LK}^l$ with a proof-tree that follows $r$.  

---

7 In particular, notice that two predicates $P$ of particular interest satisfy P1 and P2: the constant predicate that is always true, and the predicate “being satisfiable”. 
Remark 2 (Soundness of \( \text{LK}^? \)). The general statement of soundness for \( \text{LK}^? \) is easily stated and is a straightforward consequence of Theorem 1 and Theorem 2: If \( \sigma \vdash_{\text{LK}} \Gamma \rightarrow \sigma' \) is derivable in \( \text{LK}^? \), then \( P(\sigma') \) and for all \( \rho \in \sigma' \), \( \vdash_{\text{LK}} \rho(\Gamma) \) is derivable in \( \text{LK} \). For the sake of brevity, we omit the general statement of completeness, which requires quantifying over black and white trees.

As we shall see in Section 6, it is useful to have a “top element” \( \top \) in \( \Psi[\] with \( [\top] \in \top \), which we feed to a proof-search procedure based on \( \text{LK}^? \), as the initial input constraint \( \sigma \) mentioned in the soundness and completeness theorems.

6 Implementation

PSYCHE is a platform for proof-search where different search strategies can be programmed as plugins via an API that guarantees the correctness of the search output [GL13]. Its architecture uses extensively OCaml’s system of modules and functors. The kernel provides an API for proof-search whose primitives are based on the focused sequent calculus developed in [FGLM13,Far13] and presented in Appendix A. In version 1.6, PSYCHE’s kernel is parameterised by a theory in the sense of \( \text{LK} \), i.e. a module implementing a ground validity predicate, and quantifiers are not supported.

The axiomatisations proposed in the previous sections were used to design version 2.0 of PSYCHE [Psy]. The kernel now implements the focused version of System \( \text{LK}^? \), and therefore the theory is required to provide the lift constraint structure in the form of a module \( \text{Constraint} \), with a type for constraints, the projection and lift maps. It must also provide a top constraint (always satisfied) with which proof-search will start, and we also require a meet operation. The module type on the right expresses, in programming terms, the concepts developed in the previous sections:

While the theory of complete proofs as expressed by \( \text{LK}^? \) does not need it, the meet operation is useful when implementing a backtracking proof-search procedure: imagine a proof-tree has been completed for some sequent \( S \), with input constraint \( \sigma_0 \) and output constraint \( \sigma_1 \); at some point the procedure may have to search again for a proof of \( S \) but with a different input constraint \( \sigma_0' \). We can check whether the first proof can be re-used by simply checking whether \( \sigma_0' \wedge \sigma_1 \) is satisfiable. Hence the requirement val meet : t -> t -> t option which should output None if the meet of the two input constraints is not satisfiable, and Some sigma if the satisfiable meet is sigma.
Finally, the function that is called at the leaves of proof-trees is \textit{consistency}, which implements the constraint-refining predicate; \texttt{ASet.t} is the type for sets of literals with meta-variables and the function returns a stream: providing an input constraint triggers computation and pops the next element of the stream if it exists. It is a pair made of an output constraint and a subset of the input set of literals. The latter indicates which literals of the input have been used to close the branch, which is useful information for lemma learning \cite{GL13}.

So far, the only instance of the module type \texttt{Theory} implemented in \textsc{Psyche} is the one for pure first-order logic based on unification. Our next steps will be to implement other theories, such as those admitting quantifier elimination like linear rational arithmetic.

7 Conclusion and Related works

The idea of using streams in tableaux is developed in \cite{Gie00}, but in the specific context of pure first-order logic: streams of instances are used. We show here how streams of constraints, from an abstract constraint structure, can be used for proof-search in an arbitrary theory that satisfies some minimal specifications. Our contribution is really about identifying those specifications in the form of an axiomatic system.

Constraints have also been integrated to various \textit{tableaux} calculi, as surveyed for instance in \cite{GH03} which proposes a nomenclature for such integrations. In this nomenclature, our approach is closest to \textit{constrained formula tableaux} or \textit{constrained branch tableaux} which propagate constraints between branches (rather than \textit{constrained tableaux} which have a global management of constraints). Again, the \textit{tableaux} calculi cited by \cite{GH03} in these categories are for specific theories and logics (pure classical logic, equality, linear temporal logic or bunched implications), while our approach is about axiomatising abstract constraint systems.

Abstract constraint systems are actually used in Constraint Logic Programming \cite{JM94,LNRA01} and Concurrent Constraint Programming \cite{SR90,SRP91}, but the applicability of constraint systems to programming leads to different (usually more demanding) axiomatisations of constraints and to a global management of constraints (with a global store that is reminiscent of \textit{constrained tableaux}). Our local management of constraints allows for subtle backtracking strategies in proof-search, undoing some steps in one branch while sticking to some more recent decisions that have been made in a different branch.

References

And92. J. M. Andreoli. Logic programming with focusing proofs in linear logic. \textit{J. Logic Comput.}, 2(3):297–347, 1992.
Dow93. G. Dowek. A complete proof synthesis method for type systems of the cube. *J. Logic Comput.*, 3(3):287–315, 1993.

Far13. M. Farrooque. *Automated reasoning techniques as proof-search in sequent calculus*. PhD thesis, Ecole Polytechnique, 2013.

FGLM13. M. Farrooque, S. Graham-Lengrand, and A. Mahboubi. A bisimulation between DPLL(T) and a proof-search strategy for the focused sequent calculus. In A. Momigliano, B. Pientka, and R. Pollack, editors, *2013 Int. Work. on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2013)*. ACM Press, 2013.

GH03. M. Giese and R. Höhne. Tableaux + constraints. In M. C. Mayer and F. Pirri, editors, *16th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux’03)*, vol. 2796 of *LNCS*. Springer, 2003.

Gie00. M. Giese. Proof search without backtracking using instance streams, position paper. In P. Baumgartner and H. Zhang, editors, *3rd Int. Workshop on First-Order Theorem Proving (FTP)*, St. Andrews, Scotland, TR 5/2000 Univ. Koblenz, pages 227–228, 2000.

Gir87. J.-Y. Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–101, 1987.

Gir91. J.-Y. Girard. A new constructive logic: Classical logic. *Math. Structures in Comput. Sci.*, 1(3):255–296, 1991.

GL13. S. Graham-Lengrand. Psyche: a proof-search engine based on sequent calculus with an LCF-style architecture. In D. Galmiche and D. Larchey-Wendling, editors, *22nd Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux’13)*, vol. 8123 of *LNCS*, pages 149–156. Springer, 2013.

JM94. J. Jaffar and M. J. Maher. Constraint logic programming: a survey. *J. Logic Programming*, 19–20, Supplement 1(0):503–581, 1994. Special Issue: Ten Years of Logic Programming.

LDM11. S. Lengrand, R. Dyckhoff, and J. McKinna. A focused sequent calculus framework for proof search in Pure Type Systems. *Logic. Methods Comput. Science*, 7(1), 2011.

LM09. C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoret. Comput. Sci.*, 410(46):4747–4768, 2009.

LNRA01. J. Leach, S. Nieva, and M. Rodríguez-Artalejo. Constraint logic programming with hereditary harrop formula. *Theory and Practice of Logic Programming*, 1(4):409–445, 2001.

MNP91. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl. Logic*, 51:125–157, 1991.

NOT05. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Abstract DPLL and abstract DPLL Modulo Theories. In F. Baader and A. Voronkov, editors, *the 11th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR’04)*, vol. 3452 of *LNCS*, pages 36–50. Springer, 2005.

NOT06. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. of the ACM Press*, 53(6):937–977, 2006.

Psy. Psyche: the Proof-Search factory for Collaborative HEuristics. Available at [http://www.lix.polytechnique.fr/~lengrand/Psyche](http://www.lix.polytechnique.fr/~lengrand/Psyche)

RV01. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and The MIT Press, 2001.

SR90. V. A. Saraswat and M. C. Rinard. Concurrent constraint programming. In P. Hudak, editor, *17th Annual ACM Symp. on Principles of Programming Languages (POPL’90)*, pages 232–245. ACM Press, 1990.

SRP91. V. A. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In D. S. Wise, editor, *18th Annual ACM Symp. on Principles of Programming Languages (POPL’91)*, pages 339–352. ACM Press, 1991.
A Focused sequent calculus

In this section we give (a fragment of) the focused sequent calculus from \[FGLM13,Fr13\], called \(\text{LK}^p(T)\), on which the Psyche implementation is based.

### Synchronous rules

\[
\begin{align*}
\Gamma \vdash [A_1] & \quad \Gamma \vdash [A_1 \lor A_2] \\
\Gamma, l^+ \models l & \quad \Gamma \vdash [A[x := t]] \\
\text{Init}_1 & \quad \Gamma \vdash [\exists x A] \\
\text{Release} & \quad \Gamma \vdash [N] \\
\end{align*}
\]

- \(\text{Init}_1\): \(l^+ \models l\) if \(l\) is positive
- \(\text{Release}\): \(N\) is negative

### Asynchronous rules

\[
\begin{align*}
\Gamma \vdash A, \Delta & \quad \Gamma \vdash B, \Delta \\
\Gamma \vdash A \land B, \Delta & \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash A, \Delta & \quad (\forall) \quad \Gamma \vdash \forall x A, \Delta \\
\Gamma \vdash \forall x A, \Delta & \quad x \notin \text{FV}(\Gamma, \Delta) \\
\end{align*}
\]

- \(\text{Store}\): \(\Gamma, A^+ \vdash \Delta\) if \(A\) is a literal or is positive

### Structural rules

\[
\begin{align*}
\Gamma, P^+ \vdash [P] & \quad \Gamma, P^+ \vdash \text{Select} \\
\Gamma \vdash \text{Init}_2 & \quad \Gamma \vdash \text{Init}_1 \models \\
\end{align*}
\]

#### Figure 5. System \(\text{LK}^p(T)\)

B Full proofs

**Theorem 1** (Soundness and completeness of \(\text{LK}^2\)).

- If \(\vdash_n \Gamma \rightarrow \sigma\) is derivable in \(\text{LK}^2\), then for all \(\rho \epsilon \sigma\), \(\vdash_n \rho(\Gamma)\) is derivable in \(\text{LK}\).
- For all domains \(l\), for all contexts \(\Gamma\) of domain \(l\), for all \(\rho \in T_l\), if \(\vdash_n \rho(\Gamma)\) is derivable in \(\text{LK}\), then there exists \(\sigma \in \Psi_l\) such that \(\vdash_n \Gamma \rightarrow \sigma\) is derivable in \(\text{LK}^2\) and \(\rho \epsilon \sigma\).

**Proof.** We first prove the soundness of \(\text{LK}^2\) by induction on the derivation of \(\vdash_n \Gamma \rightarrow \sigma\):

- **Theory**

\[
\vdash_n \Gamma \rightarrow \sigma \quad \models l \Gamma_{\text{lit}} \rightarrow \sigma
\]

Let \(\rho \epsilon \sigma\). By A (right-to-left inclusion), we have \(\rho \in \{\rho \mid \models \rho(\Gamma_{\text{lit}})\}\).
\exists\text{-intro}

\[
\frac{\vdash^m_n \Gamma; A \left[ x := \varphi_{|l| + 1} \right], \exists x A \to \sigma}{\vdash^l_n \Gamma; \exists x A \to \sigma}
\]

Let \( \rho \subseteq \sigma \). By B2, we have \((f_\sigma(\rho) :: \rho)\) and applying the induction hypothesis we can construct

\[
\vdash^l_n \rho(\Gamma), (\rho(A))[x := f_\sigma(\rho)], \rho(\exists x A) \to \sigma
\]

\wedge\text{-intro}

\[
\frac{\vdash^l_n \Gamma; A \to \sigma \quad \vdash^l_n \Gamma; B \to \sigma'}{\vdash^l_n \Gamma; A \wedge B \to \sigma \wedge \sigma'}
\]

Let \( \rho \subseteq \sigma \wedge \sigma' \). By C1 (right-to-left), we have \( \rho \subseteq \sigma \) and \( \rho \subseteq \sigma' \) and we can conclude by applying the induction hypothesis.

\vee\text{-intro}

\[
\frac{\vdash^l_n \Gamma; A, B \to \sigma}{\vdash^l_n \Gamma; A \vee B \to \sigma}
\]

We conclude immediately with the induction hypothesis.

\forall\text{-intro}

\[
\frac{\vdash^q_{n+1} \Gamma; A \left[ x := e_{n+1} \right] \to \sigma}{\vdash^q_n \Gamma; \forall x A \to \sigma}
\]

We conclude immediately with the induction hypothesis.

We now prove the completeness result. We first draw a simple observation about definition 5:

Remark 3. Consider two domains \( l_1 \) and \( l_2 \) such that \( l_2 \) extends \( l_1 \) and two instantiations \( \rho_1 \in T_{l_1} \) and \( \rho_2 \in T_{l_2} \) such that \( \rho_2 \) extends \( \rho_1 \). Then the instantiations \( \rho_1 \) and \( \rho_2 \) coincide on domain \( l_1 \): for any term \( u \) (resp. formula \( A \), context \( \Gamma \)), \( \rho_1(u) \) (resp. \( \rho_1(A), \rho_1(\Gamma) \)) is syntactically equal to \( \rho_2(u) \) (resp. \( \rho_2(A), \rho_2(\Gamma) \)).

Consider a domain \( l \), an instantiation \( \rho \in T_l \) and a context \( \Gamma \) of domain \( l \). The proof goes by induction on the derivation of \( \vdash^l_n \rho(\Gamma) \) in the original calculus:

Theory

\[
\vdash^l_n \rho(\Gamma) \models \Gamma_{\text{lit}}
\]

Since the constraint-producing predicate relates to the ground validity predicate, by definition 9 there exists a constraint \( \sigma \in \Psi_l \) such that:

\[
\models \Gamma_{\text{lit}} \to \sigma \text{ and } \rho \subseteq \sigma
\]

The proof follows by applying the theory rule of the LK\(^3\) system.

\exists\text{-intro}
Note that instantiations do not feature meta-variables, hence for any context \( \Gamma \) and any formula \( A, \rho(\Gamma, \exists x A) \) is syntactically equal to \( (\rho(\Gamma), \exists x, \rho(A)) \). In the present case of the induction, the derivation hence ends with a rule of the form:

\[
\vdash_n \rho(\Gamma), \rho(A) [x := t], \exists x \rho(A)
\]

Introduce the instantiation \( \rho' := t :: \rho \in T_{n:t} \). The formula \( \rho(A) [x := t] \) is syntactically equal to \( \rho'(A) [x := ?_{(|l|+1)}] \). By remark 3, the premiss of the former rule can hence be written as:

\[
\vdash_n \rho'(\Gamma), \rho'(A) [x := ?_{(|l|+1)}], \exists x \rho'(A)
\]

which is in turn of the form:

\[
\vdash_n \rho'(\Gamma), A [x := ?_{[n;|l|]}], \exists x A
\]

By induction hypothesis, there exists \( \sigma' \) such that:

\[
\vdash_n \Gamma, A [x := ?_{[n;|l|]}], \exists x A \rightarrow \sigma' \text{ and } \rho \varepsilon \sigma'
\]

Therefore \( \vdash_n \Gamma, \exists x A \rightarrow \sigma \) with \( \sigma = \sigma'_t \) and \( \rho \varepsilon \sigma \) follows from axiom B1 of Definition 6.

\[\land\text{-intro}\]

\[
\vdash_n \rho(\Gamma), \rho(A), \vdash_n \rho(\Gamma), \rho(B)
\]

By induction hypothesis, there exist \( \sigma_1, \sigma_2 \in \Psi_l \) such that:

\[
\rho \varepsilon \sigma_1, \quad \rho \varepsilon \sigma_2, \quad \vdash_n \Gamma, A \rightarrow \sigma_1, \quad \vdash_n \Gamma, B \rightarrow \sigma_2
\]

Hence \( \vdash_n \Gamma, A \land B \rightarrow \sigma_1 \land \sigma_2 \) and \( \rho \varepsilon \sigma_1 \land \sigma_2 \) follows from axiom C1 of Definition 6.

\[\lor\text{-intro}\]

We conclude immediately by induction hypothesis.

\[\forall\text{-intro}\]

We conclude immediately by induction hypothesis since instantiations do not affect eigenvariables.

**Theorem 2 (Soundness of LK\(^n\)).**

If \( \sigma \rightarrow \vdash_n \Gamma \rightarrow \sigma' \) is derivable in LK\(^n\), then there exists \( \sigma'' \in \Psi_l \) such that \( \sigma' \simeq \sigma \land \sigma'' \), \( P(\sigma \land \sigma'') \) and \( \vdash_n \Gamma \rightarrow \sigma'' \) is derivable in LK\(^n\).

**Proof.** By induction on the derivation of \( \sigma \rightarrow \vdash_n \Gamma \rightarrow \sigma' \):

Theory

\[
\sigma \rightarrow \vdash_n \Gamma \rightarrow \sigma' \quad \sigma \rightarrow \vdash_{\Gamma_{\text{lit}}} \rightarrow \sigma'
\]

By A1, there exists \( \sigma'' \in \Psi_l \) such that \( \sigma' \simeq \sigma \land \sigma'' \), \( P(\sigma \land \sigma'') \) and \( \vdash_{\Gamma_{\text{lit}}} \rightarrow \sigma'' \). We can then immediately conclude.
∃-intro

\[ \sigma^\uparrow \vdash \neg n : \Gamma, A[x := ?_{n+1}] \vdash \exists x A \to \sigma' \]

\[ \sigma \vdash l_n \Gamma, \exists x A \to \sigma'_i \]

By the induction hypothesis, there exists \( \sigma'' \in \Psi_{n:i} \) such that

\[ \sigma' \simeq \sigma^\uparrow \land \sigma'' \)

\( P\left( \sigma^\uparrow \land \sigma'' \right) \)

and

\[ \vdash \neg n : \Gamma, \exists x A \to \sigma'' \]

is derivable in \( LK^2 \). Thus,

\[ \vdash l_n \Gamma, \exists x A \to \sigma''_+ \]

is derivable in \( LK^2 \). Let us then show that \( \sigma''_+ \) satisfies each constraint:

Since \( \sigma'' \in \Psi_{n:i} \), we have \( \sigma''_+ \in \Psi_i \). By D2, knowing \( \sigma' \simeq \sigma^\uparrow \land \sigma'' \), we get

\[ \sigma'_+ \simeq \sigma \land \sigma''_+ \]

Since \( \sigma^\uparrow \land \sigma'' \simeq \sigma' \) and \( P\left( \sigma^\uparrow \land \sigma'' \right) \), by P2 we get \( P\left( \sigma' \right) \). Then, by P1 (left-to-right), \( P\left( \sigma'_+ \right) \) holds. Again by P2 we get \( P\left( \sigma \land \sigma''_+ \right) \).

∧-intro

\[ \sigma \vdash l_n \Gamma, A_1 \to \sigma'' \]

\[ \sigma'' \vdash l_n \Gamma, A_{1-i} \to \sigma' \]

\[ \sigma \vdash l_n \Gamma, A_0 \land A_1 \to \sigma' \]

By applying the induction hypothesis to both premisses, we get \( \sigma_0, \sigma_1 \in \Psi_I \) such that \( \sigma'' \simeq \sigma \land \sigma_0, \sigma' \simeq \sigma'' \land \sigma_1, P\left( \sigma \land \sigma_0 \right), P\left( \sigma'' \land \sigma_1 \right) \), and \( \vdash l_n \Gamma, A_i \to \sigma_0 \) and \( \vdash l_n \Gamma, A_{1-i} \to \sigma_1 \) are derivable in \( LK^2 \). Thus,

\[ \vdash l_n \Gamma, A_0 \land A_1 \to \sigma_0 \land \sigma_1 \]

is derivable in \( LK^2 \). Let us show that \( \sigma_0 \land \sigma_1 \) satisfies each constraint:

First, \( \sigma_0 \land \sigma_1 \in \Psi_I \) holds. Then, we have to show that \( \sigma' \simeq \sigma \land \sigma_0 \land \sigma_1 \). Using D1,

\[ \sigma'' \land \sigma_1 \simeq \sigma'' \leq \sigma \land \sigma_0 \land \sigma_1 \]

Similarly, \( \sigma'' \land \sigma_1 \leq \sigma_1 \), and then \( \sigma'' \land \sigma_1 \leq \sigma \land \sigma_0 \land \sigma_1 \). Thus, by transitivity,

\[ \sigma' \leq \sigma \land \sigma_0 \land \sigma_1 \]

The other inequality holds by the same argument so that \( \sigma' \simeq \sigma \land \sigma_0 \land \sigma_1 \).

We have seen that \( \sigma'' \land \sigma_1 \leq \sigma \land \sigma_0 \land \sigma_1 \). Hence, using P1 (left-to-right) and knowing \( P\left( \sigma'' \land \sigma_1 \right) \), we get \( P\left( \sigma \land \sigma_0 \land \sigma_1 \right) \).

∨-intro

\[ \sigma \vdash l_n \Gamma, A, B \to \sigma' \]

\[ \sigma \vdash l_n \Gamma, A \lor B \to \sigma' \]

We conclude immediately with the induction hypothesis.

∀-intro

\[ \sigma \vdash l_{n+1} \Gamma, A[x := e_{n+1}] \to \sigma' \]

\[ \sigma \vdash l_n \Gamma, \forall x A \to \sigma' \]

We conclude immediately with the induction hypothesis.
Theorem 3 (Completeness of $\text{LK}^7$).

If $\vdash_n^l \Gamma 
\rightarrow \sigma'$ is derivable in $\text{LK}^7$, then for all $\sigma \in \Psi_l$ such that $P(\sigma \land \sigma')$, and for all black and white tree $r$, there exists $\sigma'' \in \Psi_l$ such that $\sigma'' \simeq \sigma \land \sigma'$ and $\sigma \rightarrow \vdash_n^l \Gamma \rightarrow \sigma''$ is derivable in $\text{LK}^7$ with a proof-tree that follows $r$.

Proof. By induction on the derivation of $\vdash_n^l \Gamma \rightarrow \sigma'$:

Theory

\[
\vdash_n^l \Gamma \rightarrow \sigma' \quad \vdash_n^l \Gamma_{\text{lit}} \rightarrow \sigma'
\]

Let $\sigma \in \Psi_l$ such that $P(\sigma \land \sigma')$ and $r$ be a black and white tree. By A2, there exists $\sigma'' \in \Psi_l$ such that $\sigma'' \simeq \sigma \land \sigma'$ and $\sigma \rightarrow \vdash_n^l \Gamma \rightarrow \sigma''$. Thus, $\sigma \rightarrow \vdash_n^l \Gamma \rightarrow \sigma''$ is derivable in $\text{LK}^7$ using the theory rule. The proof-tree follows $r$, for it has no premiss.

$\exists$-intro

\[
\vdash_n^{l;A} \Gamma, A [x := ?_{|l|+1}], \exists x A \rightarrow \sigma' \\
\vdash_n^l \Gamma, \exists x A \rightarrow \sigma''
\]

By induction hypothesis, for all $\sigma \in \Psi_n;\vdash$ such that $P(\sigma \land \sigma')$, for all black and white tree $r$, there exists $\sigma_0 \in \Psi_n;\vdash$ such that $\sigma_0 \simeq \sigma \land \sigma'$ and $\sigma \rightarrow \vdash_n^{l;A} \Gamma, A [x := ?_{|l|+1}], \exists x A \rightarrow \sigma_0$

is derivable in $\text{LK}^7$ with a proof-tree following $r$. Let $\sigma \in \Psi_l$ such that $P(\sigma \land \sigma')$ and $r$ be a black and white tree. If we find $\sigma'' \in \Psi_n;\vdash$ such that $\sigma'' \simeq \sigma^\uparrow \land \sigma'$ and $\sigma^\uparrow \rightarrow \vdash_n^{l;A} \Gamma, A [x := ?_{|l|+1}], \exists x A \rightarrow \sigma''$

is derivable in $\text{LK}^7$ with a proof-tree following $r$, then we can conclude by using D2 and Definition 13. Since $\sigma^\uparrow \in \Psi_n;\vdash$, we just need to show that $P(\sigma^\uparrow \land \sigma')$ holds and to apply the induction hypothesis. By D2, $(\sigma^\uparrow \land \sigma') \simeq \sigma \land \sigma'_1$. Hence, from the assumption $P(\sigma \land \sigma'_1)$ we derive $P((\sigma^\uparrow \land \sigma')_1)$ with P2, and then by P1 (right-to-left) we conclude $P(\sigma^\uparrow \land \sigma')$.

$\land$-intro

\[
\vdash_n^l \Gamma, A_0 \rightarrow \sigma_0 \\
\vdash_n^l \Gamma, A_1 \rightarrow \sigma_1 \\
\vdash_n^l \Gamma, A_0 \land A_1 \rightarrow \sigma_0 \land \sigma_1
\]

Let $\sigma \in \Psi_l$ such that $P(\sigma \land \sigma_0 \land \sigma_1)$ and $r$ be a black and white tree. Without loss of generality, we can assume that the root of $r$ is white. D1 gives us $\sigma \land \sigma_0 \land \sigma_1 \leq \sigma \land \sigma_0$. Then, with P1 (left-to-right), we get $P(\sigma \land \sigma_0)$.

Hence, we can apply the induction hypothesis on the left subtrees and on $\sigma$. So, there exists $\sigma' \in \Psi_l$ such that $\sigma' \simeq \sigma \land \sigma_0$ and $\sigma \rightarrow \vdash_n^l \Gamma, A_0 \rightarrow \sigma'$.
is derivable in $\mathrm{LK}^{\triangledown}$ with a proof-tree following the left subtree of $r$.

Using D1, we easily obtain $\sigma \land \sigma_0 \land \sigma_2 \subseteq \sigma' \land \sigma_1$ and then with P1 (left-to-right) we get $P(\sigma' \land \sigma_1)$\(^8\). Thus, by applying the induction hypothesis to the right subtrees and to $\sigma'$, we get $\sigma'' \in \Psi_i$ such that $\sigma'' \simeq \sigma' \land \sigma_1$ and

$$\sigma' \vdash^l_n \Gamma, A_1 \rightarrow \sigma''$$

is derivable in $\mathrm{LK}^{\triangledown}$ with a proof-tree following the right subtree of $r$.

Thus,

$$\sigma \vdash^l_n \Gamma, A_0 \land A_1 \rightarrow \sigma''$$

is derivable in $\mathrm{LK}^{\triangledown}$ with a proof-tree following $r$. There remains to show that $\sigma'' \simeq \sigma \land \sigma_0 \land \sigma_1$, knowing $\sigma'' \simeq \sigma' \land \sigma_1$ and $\sigma' \simeq \sigma \land \sigma_0$. This kind of proof has been done in the $\land$-intro part of the proof of Theorem 2.

$\lor$-intro

$$\vdash^l_n \Gamma, A, B \rightarrow \sigma'$$

$$\vdash^l_n \Gamma, A \lor B \rightarrow \sigma'$$

We conclude immediately with the induction hypothesis.

$\forall$-intro

$$\vdash^l_{n+1} \Gamma, A[x := e_{n+1}] \rightarrow \sigma'$$

$$\vdash^l_n \Gamma, \forall x A \rightarrow \sigma'$$

We conclude immediately with the induction hypothesis.

**Lemma 1.** Given the axioms of Fig. 4, $(\leq, \land, P)$ is decent.

**Proof.** First, notice that Axiom C1 (together with Definition 14) makes $\land$ implement a set-theoretic intersection in the following sense:

$$\{\rho \in T_i \mid \rho \epsilon(\sigma \land \sigma')\} = \{\rho \in T_i \mid \rho \epsilon \sigma\} \cap \{\rho \in T_i \mid \rho \epsilon \sigma'\}$$

D1 This is a direct consequence of the above remark.

D2 Assume $\sigma'' \simeq_{\epsilon} \sigma^\uparrow \land \sigma'$. We prove $\sigma'' \simeq_{\epsilon} \sigma \land \sigma'$.\(^9\)

Take $\rho \epsilon \sigma''$. By Axiom B2 we have $(f_{\sigma''}(\rho) :: \rho) \epsilon \sigma''$. By Definition 14 and the above remark we have $(f_{\sigma''}(\rho) :: \rho) \epsilon \sigma^\uparrow$ and $(f_{\sigma''}(\rho) :: \rho) \epsilon \sigma'$. By Axiom C4 (left-to-right) we have $\rho \epsilon \sigma$. By Axiom B1 we also have $\rho \epsilon \sigma'$. Hence we have $\rho \epsilon \sigma$. So $\rho \epsilon (\sigma \land \sigma')$ by the above remark. Hence, $\sigma^\uparrow \simeq_{\epsilon} \sigma \land \sigma'$.

Conversely, take $\rho \epsilon (\sigma \land \sigma')$. By the above remark, $\rho \epsilon \sigma$ and $\rho \epsilon \sigma'$. By Axiom B2 we have $(f_{\sigma'}(\rho) :: \rho) \epsilon \sigma'$, and by Axiom C4 (right-to-left) we have $(f_{\sigma'}(\rho) :: \rho) \epsilon \sigma^\uparrow$.

By the above remark we have $(f_{\sigma'}(\rho) :: \rho) \epsilon (\sigma \land \sigma')$ and by Definition 14 we have $(f_{\sigma'}(\rho) :: \rho) \epsilon \sigma''$. By Axiom B1 we have $\rho \epsilon \sigma''$. Hence, $\sigma \land \sigma' \simeq_{\epsilon} \sigma''$.

---

\(^8\) Recall the manipulations done in the $\land$-intro part of the proof of Theorem 2; those are similar.

---

\(^9\)