Axiomatic constraint systems for proof search modulo theories

Damien Rouhling[1], Mahfuza Farooque[2],
Stéphane Graham-Lengrand[2,3,4], Assia Mahboubi[3], and Jean-Marc Notin[2]

[1] École Normale Supérieure de Lyon, France
[2] CNRS - École Polytechnique, France
[3] INRIA, Centre de Recherche en Informatique Saclay-Île de France
[4] SRI International, USA

Abstract Goal-directed proof search in first-order logic uses meta-variables to delay the choice of witnesses; substitutions for such variables are produced when closing proof-tree branches, using first-order unification or a theory-specific background reasoner. This paper investigates a generalisation of such mechanisms whereby *theory-specific constraints* are produced instead of substitutions. In order to design modular proof-search procedures over such mechanisms, we provide a sequent calculus with meta-variables, which manipulates such constraints abstractly. Proving soundness and completeness of the calculus leads to an axiomatisation that identifies the conditions under which abstract constraints can be generated and propagated in the same way unifiers usually are. We then extract from our abstract framework a component interface and a specification for concrete implementations of background reasoners.

1 Introduction

A broad literature studies the integration of theory reasoning with generic automated reasoning techniques. Following Stickel's seminal work [16], different levels of interaction have been identified [2] between a theory-generic *foreground reasoner* and a theory-specific *background reasoner*, with a specific scheme for the *literal level* of interaction. In absence of quantifiers, the $\mathsf{DPLL}(\mathcal{T})$ architecture [11] is an instance of the scheme and a successful basis for SMT-solving, combining SAT-solving techniques for boolean logic with a procedure that decides whether a conjunction of ground literals is consistent with a background theory $\mathcal{T}$.

Our contribution falls into such a scheme, but in presence of quantifiers, and hence of non-ground literals. When given a conjunction of these, the background reasoner provides a means to make this conjunction inconsistent with $\mathcal{T}$, possibly by instantiating some (meta-)variables [2]. Technically, it produces a $\mathcal{T}$*-refuter* that contains a substitution.

Beckert [5] describes how this approach can be applied to *analytic tableaux*, in particular *free variable tableaux*: $\mathcal{T}$-refuters are produced to extend and eventually close a tableau branch, while the substitutions that they contain are globally applied to the tableau, thus affecting the remaining open branches. In fact, the *only* way in which closing a branch affects the other branches is the propagation of these substitutions, as it is the case for tableaux without theory reasoning. This is well-suited for some theories like equality, for which *rigid E-unification* provides a background reasoner (see e.g. [4]), but maybe not for other theories. For instance, the case of Linear Integer Arithmetic (LIA) was addressed by using arithmetic constraints, and quantifier elimination, in the Model Evolution calculus [1] and the Sequent Calculus [14] (which is closer to the above tableaux).

This paper develops sequent calculi with a more general *abstract* notion of constraints so that more theories can be treated in a similar way, starting with all theories admitting quantifier elimination. But it also covers those total theories (total in the sense that $\mathcal{T}$-refuters are just substitutions) considered by Beckert [5] for free variable tableaux, for which constraints are simply substitutions.

Sect. 2 presents a sequent calculus LK_1 with ground theory reasoning (as in $\mathsf{DPLL}(\mathcal{T})$) and various target theories that we intend to capture. Sect. 3 introduces our abstract systems of constraints. Sect. 4 presents a sequent calculus $\mathsf{LK}_1^?$ similar to Rümmer's PresPred^C_S calculus [14], but generalised with abstract constraints. It collects constraints from the parallel/independent exploration of branches, with the hope that their combination remains satisfiable. Sect. 5 and 6 present a variant $\mathsf{LK}_1^{?\rangle}$ where the treatment of branching is asymmetric, reflecting a sequential implementation of proof search: the constraint that is produced to close one branch affects the exploration of the next branch, as in free variable tableaux [5]. Each time, we prove soundness and completeness relative to the reference sequent calculus LK_1. From these proofs we extract an axiomatisation for our background theory reasoner and its associated constraints. In Sect. 7 this axiomatisation is used to define a component interface with a formal specification, for our quantifier-handling version 2.0 of the Psyche platform for theorem proving [12]. We conclude by discussing related works and future work.

2 Ground calculus and examples

The simple sequent calculus that we use in this paper uses the standard first-order notions of term, literal, eigenvariable, and formula. Following standard practice in *tableaux* methods or the linear logic tradition, we opt for a compact one-sided presentation of the sequent calculus, here called LK_1. Its rules are presented in Fig. 1, where Γ is a set (intuitively seen as a disjunction) of first-order formulae (in negation-normal form) and Γ_{lit} is the subset of its literals; $A[x := t]$ denotes the substitution of term t for all free occurrences of variable x in formula A; finally, $\models$ denotes a specific predicate, called the *ground validity predicate*, on sets of *ground* literals (i.e. literals whose variables are all eigenvariables). This predicate is used to model a given theory $\mathcal{T}$, with the intuition that $\models \Gamma_{\mathsf{lit}}$ holds when the disjunction of the literals in Γ is $\mathcal{T}$-valid. Equivalently, it holds when the conjunction of their negations is $\mathcal{T}$-inconsistent, as checked by the decision procedures used in SMT-solving. Likewise, checking whether $\models \Gamma_{\mathsf{lit}}$ holds is performed by a background reasoner, while the bottom-up application of the rule of LK_1 can serve as the basis for a *tableaux*-like foreground reasoner.

$$\frac{}{\vdash \Gamma} \models \Gamma_{\mathsf{lit}} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B}$$

$$\frac{\vdash \Gamma, A\,[x := t]\,, \exists x A}{\vdash \Gamma, \exists x A} \qquad \frac{\vdash \Gamma, A\,[x := \mathrm{x}]}{\vdash \Gamma, \forall x A}\ \text{where x is a fresh eigenvariable}$$

Figure 1. The LK_1 sequent calculus modulo theories

But a realistic proof-search procedure is in general unable to provide an appropriate witness t "out of the blue" at the time of applying an existential rule. We shall use *meta-variables* (called *free variables* in tableaux terminology) to delay the production of such instances until the constraints of completing/closing branches impact our choice possibilities. The way this happens heavily depends on the background theory, and below we give a few examples (more background on the technical notions can be found for instance in Beckert's survey [5]):

Example 1 (Pure first-order logic). In the empty theory, closing a branch $\vdash \Gamma$ is done by finding a literal l and its negation $\overline{l}$ in Γ or, if meta-variables were used, by finding a pair l and $\overline{l'}$ and a substitution σ for meta-variables that *unifies* l and l'. Such a *first-order unifier* σ may be produced by the sole analysis of $\vdash \Gamma$, or by the simultaneous analysis of the branches that need to be closed. Since the latter problem is still decidable, a global management of unification constraints is sometimes preferred, avoiding the propagation of unifiers from branch to branch.

Example 2 (First-order logic with equality). When adding equality, closing a branch $\vdash \Gamma$ is done by finding in Γ either an equality $t{=}u$ such that $\Gamma_E \models_E t{=}u$, or a pair of literals $p(t_1, \ldots, t_n)$ and $\overline{p}(u_1, \ldots, u_n)$ such that $\Gamma_E \models_E t_1 = u_1 \wedge \cdots \wedge t_n = u_n$, where Γ_E is the set of all equalities $a = b$ such that $a \neq b$ is in Γ, and $\models_E$ is entailment in the theory of equality. *Congruence closure* can be used to check this entailment. If meta-variables were used, then a substitution σ for meta-variables has to be found such that e.g. $\sigma(\Gamma_E) \models_E \sigma(t) = \sigma(u)$, a problem known as *rigid E-unification.* While this problem is decidable, finding a substitution that simultaneously closes several open branches (*simultaneous rigid E-unification*) is undecidable. A natural way to use rigid E-unification is to produce a *stream* of substitutions from the analysis of one branch and propagate them into the other branches; if at some point we have difficulties closing one of these, we can try the next substitution in the stream.

The idea of producing streams of substitutions at the leaves of branches (advocated by Giese [8]) can be taken further:

Example 3 (Theories with ground decidability). Any theory whose ground validity predicate is decidable has a semi-decision procedure that "handles" meta-variables: to close a branch $\vdash \Gamma$ with meta-variables, enumerate as a stream all substitutions to ground terms (i.e. terms whose variables are all eigenvariables), and filter out of it all substitutions σ such that $\not\models \sigma(\Gamma)_{\mathsf{lit}}$. Stream productivity -and therefore decidability- may thus be lost, but completeness of proof search in first-order logic already requires fairness of strategies with e.g. iterative deepening methods, which may as well include the computation of streams.

While this mostly seems an impractical theoretical remark, heuristics can be used (e.g. first trying those ground terms that are already present in the problem) that are not far from what is implemented in SMT-solvers (like *triggers* [6]).

The enumeration strategy can also be theory-driven, and also make use of substitutions to non-ground terms: An interesting instance of this is higher-order logic expressed as a first-order theory, λ-terms being encoded as first-order terms using De Bruijn's indices, and $\beta\eta$-equivalence being expressed with first-order axioms. Similarly to Example 1, closing a branch with meta-variables requires solving (higher-order) unification problems, whose (semi-decision) algorithms can be seen as complete but optimised enumeration techniques.

All of the above examples use substitutions of meta-variables as the output of a successful branch closure, forming *total background reasoners* for the tableaux of [5]. But by letting successful branch closures produce a more general notion of theory-specific *constraints*, we also cover examples such as:

Example 4 (Theories with quantifier elimination). When a theory satisfies quantifier elimination (such as linear arithmetic), the provability of arbitrary formulae can be reduced to the provability of quantifier-free formulae. This reduction can be done with the same proof-search methodology as for the previous examples, provided successful branch closures produce other kinds of data-structures. For instance with p an uninterpreted predicate symbol, $l(x,y) := 3x \leq 2y \leq 3x{+}1$ and $l'(x,y) := 99 \leq 3y{+}2x \leq 101$, the foreground reasoner will turn the sequent

$$\vdash (\exists xy(p(x,y) \wedge l(x,y))) \vee (\exists x'y'(\overline{p}(x',y') \wedge l'(x',y'))$$

into a tree with 4 branches, with meta-variables $?X$, $?X'$, $?Y$, and $?Y'$:

$$\begin{array}{ll} \vdash p(?X,?Y),\overline{p}(?X',?Y') & \vdash l(?X,?Y),\overline{p}(?X',?Y') \\ \vdash p(?X,?Y),l'(?X',?Y') & \vdash l(?X,?Y),l'(?X',?Y') \end{array}$$

While it is clear that the background reasoner will close the top-left leaf by producing the substitution identifying $?X$ with $?X'$, $?Y$ with $?Y'$, it is hard to see how the analysis of any of the other branches could produce, on its own and not after a lengthy enumeration, a substitution that is, or may be refined into, the unique integer solution $?X \mapsto 15, ?Y \mapsto 23$. Hence the need for branches to communicate to other branches more appropriate data-structures than substitutions, like *constraints* (in this case, arithmetic ones).

In the rest of this paper, all of the above examples are instances of an abstract notion of theory module that comes with its own system of constraints.

3 Constraint Structures

Meta-variables (denoted $?X$, $?Y$, etc) can be thought of as place-holders for yet-to-come instantiations. Delayed though these may be, they must respect the freshness conditions from System LK_1, so dependencies between meta-variables and eigenvariables must be recorded during proof search.

While Skolem symbols are a convenient implementation of such dependencies when the theory reasoner is unification-based (*occurs check* ruling out incorrect instantiations for free), we record them in a data-structure, called *domain*, attached to each sequent. Two operations are used on domains: adding to a domain d a fresh eigenvariable x (resp. meta-variable $?X$) results in a new domain $d;\mathrm{x}$ (resp. $d;?X$). The use of the notation always implicitly assumes x (resp. $?X$) to be fresh for d. An *initial domain* d_0 is also used before proof search introduces fresh eigenvariables and meta-variables.[5]

Definition 1 (Terms, formulae with meta-variables). *A* term *(resp.* formula*) of domain d is a term (resp. formula) whose variables (resp. free variables) are all eigenvariables or meta-variables declared in d. A term (resp. formula) is* ground *if it contains no meta-variables. Given a domain d, we define T_d to be the set of ground terms of domain d. A* context *of domain d is a multiset of formulae of domain d. In the rest of this paper, a* free variable *(of domain d) means either an eigenvariable or a meta-variable (declared in d).*

In this setting, the axiom rule of system LK_1 is adapted to the presence of meta-variables in literals, so as to produce theory-specific *constraints* on (yet-to-come) instantiations. We characterise the abstract structure that they form:

Definition 2 (Constraint structures). *A* constraint structure *is:*

- *a family of sets $(\Psi_d)_d$, indexed by domains and satisfying $\Psi_{d;\mathrm{x}} = \Psi_d$ for all domains d and eigenvariables x; elements of Ψ_d are called* constraints of domain d*, and are denoted σ, σ', etc.*
- *a family of mappings from $\Psi_{d;?X}$ to Ψ_d for all domains d and meta-variables $?X$, called* projections*, mapping constraints $\sigma \in \Psi_{d;?X}$ to constraints $\sigma_\downarrow \in \Psi_d$.*

A meet constraint structure *is a constraint structure $(\Psi_d)_d$ with a binary operator $(\sigma, \sigma') \mapsto \sigma \wedge \sigma'$ on each set Ψ_d.*

A lift constraint structure *is a constraint structure $(\Psi_d)_d$ with a map $\sigma \mapsto \sigma^\uparrow$ from Ψ_d to $\Psi_{d;?X}$ for all domains d and meta-variables $?X$.*

Intuitively, each mapping from $\Psi_{d;?X}$ to Ψ_d projects a constraint concerning the meta-variables declared in $(d;?X)$ to a constraint on the meta-variables in d. Different constraints can be used for different theories:

Example 5. 1. In Examples 1 and 2, it is natural to take Ψ_d to be the set whose elements are either $\bot$ (to represent the unsatisfiable constraint) or a substitution σ for the meta-variables in d.[6] Projecting a substitution from $\Psi_{d;?X}$ is just erasing its entry for $?X$. The meet of two substitutions is their most general unifier, and the lift of $\sigma \in \Psi_d$ into $\Psi_{d;?X}$ is $\sigma, ?X \mapsto ?X$.

[5] For instance, a domain may be implemented as a pair $(\Phi; \Delta)$, where Φ is the set of declared eigenvariables and Δ maps every declared meta-variable to the set of eigenvariables on which it is authorised to depend. With this implementation, $(\Phi; \Delta); \mathrm{x} := (\Phi, \mathrm{x}; \Delta)$ and $(\Phi; \Delta); ?X := (\Phi; \Delta, ?X \mapsto \Phi)$. We also set $d_0 = (\Phi_0, \emptyset)$, with Φ_0 already containing enough eigenvariables so as to prove e.g. $\exists x(p(x) \vee \overline{p}(x))$.

[6] Technically, the term $\sigma(?X)$, if defined, features only eigenvariables among those authorised for $?X$ by d, and meta-variables outside d or mapped to themselves by σ.

2. In Example 3, the default constraint structure would restrict the above to substitutions that map meta-variables to either themselves or to ground terms, unless a particular theory-specific enumeration mechanism could make use of non-ground terms (such as higher-order unification).
3. In Example 4, $\Psi_{d;\mathrm{x}} = \Psi_d$ for any d, and we take Ψ_{d_0} (resp. $\Psi_{d;?X}$) to be the set of quantifier-free formulae of domain d_0 (resp. $d;?X$). Quantifier elimination provides projections, the meet operator is conjunction and the lift is identity.

4 A system for proof search with constraints

In the rest of this section $(\Psi_d)_d$ denotes a fixed meet constraint structure.

4.1 The constraint-producing sequent calculus $\mathsf{LK}_1^?$

This sequent calculus is parameterised by a background theory reasoner that can handle meta-variables. The reasoner is modelled by a *constraint-producing predicate* that generalises the ground validity predicate used in System LK_1.

Definition 3 ($\mathsf{LK}_1^?$ sequent calculus).
A constraint-producing predicate *is a family of relations* $(\models^d)_d$*, indexed by domains d, relating sets $\mathcal{A}$ of literals of domain d with constraints σ in Ψ_d; when it holds, we write* $\models^d \mathcal{A} \to \sigma$*.*

Given such a predicate $(\models^d)_d$*, the* constraint-producing sequent calculus $\mathsf{LK}_1^?$ *manipulates sequents of the form $\vdash^d \Gamma \to \sigma$, where Γ is a context and σ is a constraint, both of domain d. Its rules are presented in Fig. 2.*

$$\frac{}{\vdash^d \Gamma \to \sigma}\ \models^d \Gamma_{\mathsf{lit}} \to \sigma \qquad \frac{\vdash^d \Gamma, A \to \sigma \qquad \vdash^d \Gamma, B \to \sigma'}{\vdash^d \Gamma, A \wedge B \to \sigma \wedge \sigma'} \qquad \frac{\vdash^d \Gamma, A, B \to \sigma}{\vdash^d \Gamma, A \vee B \to \sigma}$$

$$\frac{\vdash^{d;?X} \Gamma, A\,[x := ?X]\,, \exists x A \to \sigma}{\vdash^d \Gamma, \exists x A \to \sigma_{\downarrow}}\ \text{where } ?X \text{ is a fresh meta-variable} \qquad \frac{\vdash^{d;\mathrm{x}} \Gamma, A\,[x := \mathrm{x}] \to \sigma}{\vdash^d \Gamma, \forall x A \to \sigma}\ \text{where x is a fresh eigenvariable}$$

Figure 2. The constraint-producing sequent calculus $\mathsf{LK}_1^?$

In terms of process, a sequent $\vdash^d \Gamma \to \sigma$ displays the inputs Γ, d and the output σ of proof search, which starts building a proof tree, in system $\mathsf{LK}_1^?$, from the root. The sequent at the root would typically be of the form $\vdash^{d_0} \Gamma \to \sigma$, with $\sigma \in \Psi_{d_0}$ to be produced as output. The constraints are produced at the leaves, and propagated back down towards the root.

Example 6. In Examples 1, 2, 3, the constraint-producing predicate $\models^d \mathcal{A} \to \sigma$ holds if, respectively, σ is the most general unifier of two dual literals in $\mathcal{A}$, σ is an output of rigid E-unification on $\mathcal{A}$, σ is a ground substitution for which $\sigma(\mathcal{A})$ is $\mathcal{T}$-inconsistent. In Example 4, $\models^d \mathcal{A} \to \sigma$ holds if the quantifier-free formula σ

(of appropriate domain) implies $\mathcal{A}$ (as a disjunction). For our specific example, which also involves uninterpreted predicate symbols, proof search in system $\mathsf{LK}_1^?$ builds a tree

$$\dfrac{\dfrac{\dfrac{\begin{array}{cc}\vdash^d p(?X,?Y),\overline{p}(?X',?Y') \to \sigma_1 & \vdash^d l(?X,?Y),\overline{p}(?X',?Y') \to \sigma_2 \\ \vdash^d p(?X,?Y),l'(?X',?Y') \to \sigma_3 & \vdash^d l(?X,?Y),l'(?X',?Y') \to \sigma_4\end{array}}{\dots}}{\vdash^d (p(?X,?Y) \wedge l(?X,?Y)),(\overline{p}(?X',?Y') \wedge l'(?X',?Y')) \to \sigma}}{\dfrac{\dots}{\vdash^{d_0} (\exists xy(p(x,y) \wedge l(x,y))) \vee (\exists x'y'(\overline{p}(x',y') \wedge l'(x',y'))) \to \sigma_{\downarrow\downarrow\downarrow\downarrow}}}$$

where $d := ?X; ?Y; ?X'; ?Y'$, the background reasoner produces $\sigma_1 := \{?X = ?X'; ?Y = ?Y'\}$, $\sigma_2 := \{3?X \leq 2?Y \leq 3?X+1\}$, $\sigma_3 := \{99 \leq 3?Y'+2?X' \leq 101\}$, and $\sigma_4 := \sigma_2$ ($\sigma_4 := \sigma_3$ also works); then $\sigma := (\sigma_1 \wedge \sigma_2) \wedge (\sigma_3 \wedge \sigma_4)$ and finally $\sigma_{\downarrow\downarrow\downarrow\downarrow}$, obtained by quantifier elimination from σ, is the trivially true formula.

System $\mathsf{LK}_1^?$ is very close to Rümmer's PresPred$_S^C$ System [14], but using abstract constraints instead of linear arithmetic constraints. Using $\mathsf{LK}_1^?$ with the constraint structure of Example 5.3 implements Rümmer's suggestion [14] to eliminate quantifiers along the propagation of constraints down to the root.

4.2 Instantiations and Compatibility with Constraints

Notice that, in system $\mathsf{LK}_1^?$, no instantiation for meta-variables is actually ever produced. Instantiations would only come up when reconstructing, from an $\mathsf{LK}_1^?$ proof, a proof in the original calculus LK_1. So as to relate constraints to actual instantiations, we formalise what it means for an instantiation to satisfy, or be compatible with, a constraint of domain d. Such an instantiation should provide, for each meta-variable, a term that at least respects the eigenvariable dependencies specified in d, as formalised in Definition 4. 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 5 some minimal axioms. We list these axioms, along with the rest of this paper's axiomatisation, in Fig. 4 on page 13.

Definition 4 (Instantiation).
The set of instantiations of domain d*, denoted* Σ_d*, is the set of mappings from meta-variables to ground terms defined by induction on* d *as follows:*

$$\Sigma_{d_0} = \emptyset \qquad \Sigma_{d;\mathrm{x}} = \Sigma_d \qquad \Sigma_{d;?X} = \{\rho, ?X \mapsto t \mid t \in T_d, \rho \in \Sigma_d\}$$

For a term t *(resp. a formula* A*, a context* Γ*) of domain* d *and an instantiation* $\rho \in \Sigma_d$*, we denote by* $\rho(t)$ *(resp.* $\rho(A)$*,* $\rho(\Gamma)$*) the result of substituting in* t *(resp.* A*,* Γ*) each meta-variable* $?X$ *in* d *by its image through* ρ*.*

Definition 5 (Compatibility relation). *A* compatibility relation *is a (family of) relation(s) between instantiations* $\rho \in \Sigma_d$ *and constraints* $\sigma \in \Psi_d$ *for each domain* d*, denoted* $\rho \epsilon \sigma$*, that satisfies Axiom* Proj *of Fig. 4.*

If the constraint structure is a meet constraint structure, we say that the compatibility relation distributes *over* $\wedge$ *if it satisfies Axiom* Meet *of Fig. 4.*

Another ingredient we need to relate the two sequent calculi is a mechanism for producing instantiations. We formalise a *witness builder* which maps every constraint of $\Psi_{d;?X}$ to a function, which outputs an "appropriate" instantiation for $?X$ when given as input an instantiation of domain d:

Definition 6 (Witness). *A* witness builder *for a compatibility relation* ϵ *is a (family of) function(s) that maps every* $\sigma \in \Psi_{d;?X}$ *to* $f_\sigma \in \Sigma_d \to T_d$*, for every domain* d *and every meta-variable* $?X$*, and that satisfies Axiom* Wit *of Fig. 4.*

Example 7. For the constraint structure of Example 5.1, we can define: $\rho\epsilon\sigma$ if ρ is a (ground) instance of substitution σ. Given $\sigma \in \Psi_{d;?X}$ and $\rho\epsilon\sigma_\downarrow$, we have $\rho = \rho' \circ \sigma_\downarrow$, and we can take $f_\sigma(\rho)$ to be any instance of $\rho'(\sigma(?X))$ in Σ_d.

In the particular case of Example 5.2, $\rho\epsilon\sigma$ if ρ coincides with σ (on every meta-variable not mapped to itself by σ). To define $f_\sigma(\rho)$, note that $\rho'(\sigma(?X))$ is either ground or it is $?X$ itself (in which case any term in Σ_d works as $f_\sigma(\rho)$).

For the constraint structure of Example 5.3, we can take: $\rho\epsilon F$ if the ground formula $\rho(F)$ is valid in the theory. From a formula $F \in \Psi_{d;?X}$ and an instantiation ρ, the term $f_F(\rho)$ should represent an existential witness for the formula $\rho(F)$, which features $?X$ as the only meta-variable. In the general case, we might need to resort to a Hilbert-style choice operator to construct the witness: $\epsilon(\exists x((\rho,?X \mapsto x)(F))$. For instance in the case of linear arithmetic, $\rho(F)$ corresponds to a disjunction of systems of linear constraints, involving $?X$ and the eigenvariables $\overrightarrow{Y}$ of d. Expressing how $?X$ functionally depends on $\overrightarrow{Y}$ to build a solution of one of the systems, may require extending the syntax of terms. But note that proof search in $\mathsf{LK}_1^?$ does not require implementing a witness builder.

A meet constraint structure can also be defined by taking constraints to be (theory-specific kinds of) sets of instantiations: the compatibility relation ϵ is just set membership, set intersection provides a meet operator and the projection of a constraint is obtained by removing the appropriate entry in every instantiation belonging to the constraint. Witness building would still be theory-specific.

To relate Systems $\mathsf{LK}_1^?$ and LK_1, we relate constraint-producing predicates to ground validity ones: intuitively, the instantiations that turn a set $\mathcal{A}$ of literals of domain d into a valid set of (ground) literals should coincide with the instantiations that are compatible with some constraint produced for $\mathcal{A}$ (a similar condition appears in Theorem 55 of [5] with $\mathcal{T}$-refuters instead of constraints):

Definition 7 (Relating predicates).
For a compatibility relation ϵ*, we say that a constraint-producing predicate* $(\models^d)_d$ relates to *a ground validity predicate* $\models$ *if they satify Axiom* PG *of Fig. 4.*

A constraint-producing predicate may allow several constraints to close a given leaf (finitely many for Example 1, possibly infinitely many for Examples 2 and 3, just one for Example 4). So in general our foreground reasoner expects a *stream* of constraints to be produced at a leaf, corresponding to the (possibly infinite) union in axiom PG: each one of them is sufficient to close the branch. The first one is tried, and if it later proves unsuitable, the next one in the stream can be tried, following Giese's suggestion [8] of using streams of instantiations.

4.3 Soundness and Completeness

System $\mathsf{LK}_1^?$ can be proved equivalent to System LK_1, from the axioms in the top half of Fig. 4 [13]. To state this equivalence, assume that we have a compatibility relation that distributes over $\wedge$, equipped with a witness builder, plus a constraint-producing predicate $(\models^d)_d$ related to a ground validity predicate $\models$.

Theorem 1 (Soundness and completeness of $\mathsf{LK}_1^?$).
For all contexts Γ of domain d:
If $\vdash^d \Gamma \rightarrow \sigma$ is derivable in $\mathsf{LK}_1^?$, then for all $\rho\epsilon\sigma$, $\vdash \rho(\Gamma)$ is derivable in LK_1.
For all $\rho \in \Sigma_d$, if $\vdash \rho(\Gamma)$ is derivable in LK_1, then there exists $\sigma \in \Psi_d$ such that $\vdash^d \Gamma \rightarrow \sigma$ is derivable in $\mathsf{LK}_1^?$ and $\rho\epsilon\sigma$.

We will usually start proof search with the domain d_0, so as to build a proof tree whose root is of the form $\vdash^{d_0} \Gamma \rightarrow \sigma$ for some constraint $\sigma \in \Psi_{d_0}$. Since the only instantiation in Σ_{d_0} is $\emptyset$, and since $\emptyset(\Gamma) = \Gamma$, soundness and completeness for domain d_0 can be rewritten as follows:

Corollary 1 (Soundness and completeness for the initial domain).
There exists $\sigma \in \Psi_{d_0}$ such that $\vdash^{d_0} \Gamma \rightarrow \sigma$ is derivable in $\mathsf{LK}_1^?$ and $\emptyset\epsilon\sigma$, if and only if $\vdash \Gamma$ is derivable in LK_1.

5 Sequentialising

The soundness and completeness properties of System $\mathsf{LK}_1^?$ 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, branching on conjunctions 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 satisfiable. This section develops a variant of System $\mathsf{LK}_1^?$ to support this sequentialisation of branches, much closer than System $\mathsf{LK}_1^?$ to the free variable tableaux with theory reasoning [5].

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

5.1 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 $(\models\!\!\!\rightarrow^d)_d$ is used that now takes an extra argument: the input constraint.

Definition 8 ($\mathsf{LK}_1^{?\rangle}$ sequent calculus).
A constraint-refining predicate *is a family of relations $(\models\!\!\!\rightarrow^d)_d$, indexed by domains d, relating sets $\mathcal{A}$ of literals of domain d with pairs of constraints σ and σ' in Ψ_d; when it holds, we write $\sigma \rightarrow\models\!\!\!\rightarrow^d \mathcal{A} \rightarrow \sigma'$.*

Given such a predicate $(\models\!\rangle^d)_d$, *the* constraint-refining sequent calculus, *denoted* $\mathsf{LK}_1^{?\rangle}$, *manipulates sequents of the form* $\sigma \rightarrow\!\vdash^d \Gamma \rightarrow \sigma'$, *where* Γ *is a context and* σ *and* σ' *are constraints, all of domain* d. *Its rules are presented in Fig. 3.*

$$\frac{}{\sigma \rightarrow\!\vdash^d \Gamma \rightarrow \sigma'}\ \sigma \rightarrow\!\models\!\rangle^d \Gamma_{\mathsf{lit}} \rightarrow \sigma' \qquad \frac{\sigma \rightarrow\!\vdash^d \Gamma, A, B \rightarrow \sigma'}{\sigma \rightarrow\!\vdash^d \Gamma, A \vee B \rightarrow \sigma'}$$

$$\frac{\sigma \rightarrow\!\vdash^d \Gamma, A_i \rightarrow \sigma'' \qquad \sigma'' \rightarrow\!\vdash^d \Gamma, A_{1-i} \rightarrow \sigma'}{\sigma \rightarrow\!\vdash^d \Gamma, A_0 \wedge A_1 \rightarrow \sigma'}\ i \in \{0,1\}$$

$$\frac{\sigma^{\uparrow} \rightarrow\!\vdash^{d;?X} \Gamma, A\,[x :=?X]\,, \exists x A \rightarrow \sigma'}{\sigma \rightarrow\!\vdash^d \Gamma, \exists x A \rightarrow \sigma'_{\downarrow}} \qquad \frac{\sigma \rightarrow\!\vdash^{d;\mathrm{x}} \Gamma, A\,[x := \mathrm{x}] \rightarrow \sigma'}{\sigma \rightarrow\!\vdash^d \Gamma, \forall x A \rightarrow \sigma'}$$

where $?X$ is a fresh meta-variable   where x is a fresh eigenvariable

Figure 3. The sequent calculus with sequential delayed instantiation $\mathsf{LK}_1^{?\rangle}$

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

Example 8. In Examples 1, 2, 3, constraints are simply substitutions, and the constraint-refining predicate $\sigma \rightarrow\!\models\!\rangle^d \mathcal{A} \rightarrow \sigma'$ is taken to hold if the constraint-producing predicate $\models^d \sigma(\mathcal{A}) \rightarrow \sigma'$ (as given in Example 6) holds. Here we recover the standard behaviour of free variable tableaux (with or without theory [5]) where the substitutions used to close branches are applied to the literals on the remaining branches. Of course in both cases, an implementation may apply the substitution lazily. In Example 4, the constraint-refining predicate $\sigma \rightarrow\!\models\!\rangle^d \mathcal{A} \rightarrow \sigma'$ is taken to hold if $\models^d (\sigma \wedge \mathcal{A}) \rightarrow \sigma'$ holds. Proof search in $\mathsf{LK}_1^{?\rangle}$ builds, for our specific example and a trivially true constraint σ_0, the proof-tree

$$\cfrac{\cfrac{\cfrac{\begin{array}{cc}\sigma_0 \rightarrow\!\vdash^d p(?X,?Y), \overline{p}(?X',?Y') \rightarrow \sigma_1' & \sigma_1' \rightarrow\!\vdash^d l(?X,?Y), \overline{p}(?X',?Y') \rightarrow \sigma_2' \\ \sigma_2' \rightarrow\!\vdash^d p(?X,?Y), l'(?X',?Y') \rightarrow \sigma_3' & \sigma_3' \rightarrow\!\vdash^d l(?X,?Y), l'(?X',?Y') \rightarrow \sigma'\end{array}}{\cdots}}{\sigma_0 \rightarrow\!\vdash^d (p(?X,?Y) \wedge l(?X,?Y)), (\overline{p}(?X',?Y') \wedge l'(?X',?Y')) \rightarrow \sigma'}}{\cfrac{\cdots}{\sigma_0 \rightarrow\!\vdash^{d_0} (\exists xy(p(x,y) \wedge l(x,y))) \vee (\exists x'y'(\overline{p}(x',y') \wedge l'(x',y'))) \rightarrow \sigma'_{\downarrow\downarrow\downarrow\downarrow}}}$$

similar to that of Example 6, where $\sigma_1' := \sigma_1$, $\sigma_2' := \sigma_1' \wedge \sigma_2$, $\sigma_3' := \sigma_2' \wedge \sigma_3$ and $\sigma' := \sigma_3'$, projected by quantifier elimination to the trivially true formula $\sigma'_{\downarrow\downarrow\downarrow\downarrow}$.

5.2 Soundness and Completeness

We now relate system $\mathsf{LK}_1^{?\rangle}$ to system $\mathsf{LK}_1^{?}$. 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 $\mathsf{LK}_1^{?}$ to LK_1, since we are not (yet) trying to relate system $\mathsf{LK}_1^{?\rangle}$ to LK_1. In the next section however, we will combine the two steps.

Definition 9 (Decency). *When $\leqslant$ (resp. $\wedge$, P) is a family of pre-orders (resp. binary operators, predicates) over each Ψ_d, we say that $(\leqslant, \wedge, P)$ is* decent *if the following axioms hold:*

D1 $\forall \sigma, \sigma' \in \Psi_d,\ \sigma \wedge \sigma'$ *is a greatest lower bound of* σ *and* σ' *for* $\leqslant$
D2 $\forall \sigma \in \Psi_d\, \forall \sigma', \sigma'' \in \Psi_{d;?X},\ \sigma'' \simeq \sigma^{\uparrow} \wedge \sigma' \ \Rightarrow\ \sigma''_{\downarrow} \ \simeq\ \sigma \wedge \sigma'_{\downarrow}$
P1 $\forall \sigma \in \Psi_{d;?X},\ P(\sigma) \Leftrightarrow P(\sigma_{\downarrow})$ *P2* $\forall \sigma, \sigma' \in \Psi_d,\ \begin{cases} P(\sigma) \\ \sigma \leqslant \sigma' \end{cases} \Rightarrow P(\sigma')$

where $\simeq$ denotes the equivalence relation generated by $\leqslant$.

Notice that this makes $({}^{\Psi_d}/_{\simeq}, \wedge)$ a meet-semilattice that could equally be defined by the associativity, commutativity, and idempotency of $\wedge$.

Definition 10 (Relating constraint-producing/refining predicates).
Given a family of binary operators $\wedge$ and a family of predicates P, we say that a constraint-refining predicate $(\models\!\!\!\!\!\!\rightarrow^d)_d$ relates to *a constraint-producing predicate $(\models^d)_d$ if, for all domains d, all sets $\mathcal{A}$ of literals of domain d and all $\sigma \in \Psi_d$,*

$$A1 \quad \forall \sigma' \in \Psi_d, \quad \sigma \rightarrow\!\models\!\!\!\!\!\!\rightarrow^d \mathcal{A} \rightarrow \sigma' \Rightarrow\ \exists \sigma'' \in \Psi_d, \begin{cases} \sigma' \simeq \sigma \wedge \sigma'' \\ P(\sigma \wedge \sigma'') \\ \models^d \mathcal{A} \rightarrow \sigma'' \end{cases}$$

$$A2 \quad \forall \sigma' \in \Psi_d, \quad \begin{cases} P(\sigma \wedge \sigma') \\ \models^d \mathcal{A} \rightarrow \sigma' \end{cases} \Rightarrow\ \exists \sigma'' \in \Psi_d, \begin{cases} \sigma'' \simeq \sigma \wedge \sigma' \\ \sigma \rightarrow\!\models\!\!\!\!\!\!\rightarrow^d \mathcal{A} \rightarrow \sigma'' \end{cases}$$

In the rest of this sub-section, we assume that we have a decent triple $(\leqslant, \wedge, P)$, and a constraint-refining predicate $(\models\!\!\!\!\!\!\rightarrow^d)_d$ that relates to a contraint-producing predicate $(\models^d)_d$. In this paper we only use two predicates P, allowing us to develop two variants of each theorem, with a compact presentation: $P(\sigma)$ is always "true", and $P(\sigma)$ is "σ is satisfiable", both of which satisfy P1 and P2.

System $\mathsf{LK}_1^{?\rangle}$ can then be proved sound with respect to System $\mathsf{LK}_1^{?}$ [13]:

Theorem 2 (Soundness of $\mathsf{LK}_1^{?\rangle}$).
If $\sigma \rightarrow\vdash^d \Gamma \rightarrow \sigma'$ is derivable in $\mathsf{LK}_1^{?\rangle}$, then there exists $\sigma'' \in \Psi_d$ such that $\sigma' \simeq \sigma \wedge \sigma''$, $P(\sigma \wedge \sigma'')$ and $\vdash^d \Gamma \rightarrow \sigma''$ is derivable in $\mathsf{LK}_1^{?}$.

Notice that the statement for soundness of Theorem 2 is merely a generalisation of axiom R2P where the reference to $\models^d$ and $\models\!\!\!\!\!\!\rightarrow^d$ have respectively been replaced by derivability in $\mathsf{LK}_1^{?}$ and $\mathsf{LK}_1^{?\rangle}$.

A natural statement for completeness of $\mathsf{LK}_1^{?\rangle}$ w.r.t. $\mathsf{LK}_1^{?}$ comes as the symmetric generalisation of axiom P2R:

Theorem 3 (Weak completeness of $\mathsf{LK}_1^{?\rangle}$). *If $\vdash^d \Gamma \rightarrow \sigma'$ is derivable in $\mathsf{LK}_1^{?}$, then for all $\sigma \in \Psi_d$ such that $P(\sigma \wedge \sigma')$, there exists $\sigma'' \in \Psi_d$ such that $\sigma'' \simeq \sigma \wedge \sigma'$ and $\sigma \rightarrow\vdash^d \Gamma \rightarrow \sigma''$ is derivable in $\mathsf{LK}_1^{?\rangle}$.*

This statement can be proved, but it fails to capture an important aspect of system $\mathsf{LK}_1^{?\rangle}$: the order in which proof search treats branches should not matter for completeness. But the above statement concludes that there *exists* a sequentialisation of branches that leads to a complete proof tree in $\mathsf{LK}_1^{?\rangle}$, so the proof-search procedure should either guess it or investigate all possibilities. We therefore proved [13] the stronger statement of completeness (below) 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.

Theorem 4 (Strong completeness of $\mathsf{LK}_1^{?\rangle}$).
If $\vdash^d \Gamma \to \sigma'$ is derivable in $\mathsf{LK}_1^{?}$, then for all $\sigma \in \Psi_d$ such that $P(\sigma \wedge \sigma')$, and for all sequentialisations r of branches, there exists $\sigma'' \in \Psi_d$ such that $\sigma'' \simeq \sigma \wedge \sigma'$ and $\sigma \to\!\vdash^d \Gamma \to \sigma''$ is derivable in $\mathsf{LK}_1^{?\rangle}$ with a proof tree that follows r.

6 Relating $\mathsf{LK}_1^{?\rangle}$ to LK_1

Now we combine the two steps: from LK_1 to $\mathsf{LK}_1^{?}$ and from $\mathsf{LK}_1^{?}$ to $\mathsf{LK}_1^{?\rangle}$, so as to relate $\mathsf{LK}_1^{?\rangle}$ to LK_1. For this we aggregate (and consequently simplify) the axioms that we used for the first step with those that we used for the second step.

Definition 11 (Compatibility-based pre-order). *Assume we have a family of compatibility relations ϵ for a constraint structure $(\Psi_d)_d$. We define the following pre-order on each Ψ_d:*

$$\forall \sigma, \sigma' \in \Psi_d,\ \sigma \leqslant_\epsilon \sigma' \Leftrightarrow \{\rho \in \Sigma_d \mid \rho \epsilon \sigma\} \subseteq \{\rho \in \Sigma_d \mid \rho \epsilon \sigma'\}$$

and let $\simeq_\epsilon$ denote the symmetric closure of $\leqslant_\epsilon$.

We now assume that we have a lift constraint structure and a constraint-refining predicate $(\models\!\!\!\dashv^d)_d$ used to define $\mathsf{LK}_1^{?\rangle}$, and the existence of

- a binary operator $\wedge$
- a compatibility relation ϵ that distributes over $\wedge$ (Proj and Meet in Fig. 4)
- a binding operator for ϵ (Wit in Fig. 4)
- a constraint-producing predicate $(\models^d)_d$ that relates to $\models$ (PG in Fig. 4)
- a predicate P

satisfying the axioms of Fig. 4. These entail decency [13]:

Lemma 1. *Given the axioms of Fig. 4, $(\leqslant_\epsilon, \wedge, P)$ is decent.*

Hence, we have soundness and completeness of $\mathsf{LK}_1^{?\rangle}$ w.r.t. LK_1 on the empty domain, as a straightforward consequence of Corollary 1 and Theorems 2 and 4:

Theorem 5 (Soundness and completeness on the empty domain).
If $\sigma \to\!\vdash^{d_0} \Gamma \to \sigma'$ is derivable in $\mathsf{LK}_1^{?\rangle}$ and $\emptyset \epsilon \sigma'$, then $\vdash \Gamma$ is derivable in LK_1. In particular when P is the predicate "being satisfiable", if $\sigma \to\!\vdash^{d_0} \Gamma \to \sigma'$ is derivable in $\mathsf{LK}_1^{?\rangle}$, then $\vdash \Gamma$ is derivable in LK_1.

Assume P is always true or is "being satisfiable". If $\vdash \Gamma$ is derivable in LK_1, then for all $\sigma \in \Psi_{d_0}$ such that $\emptyset \epsilon \sigma$ and for all sequentialisations r, there exists $\sigma' \in \Psi_{d_0}$ such that $\emptyset \epsilon \sigma'$ and $\sigma \to\!\vdash^{d_0} \Gamma \to \sigma'$ is derivable in $\mathsf{LK}_1^{?\rangle}$ with a proof tree that follows r.

Remark 1 (Soundness of $\mathsf{LK}_1^{?\rangle}$*).* Soundness of $\mathsf{LK}_1^{?\rangle}$ on an arbitrary domain is a direct consequence of Theorem 1 and Theorem 2: If $\sigma \rightarrow\!\vdash^d \Gamma \rightarrow \sigma'$ is derivable in $\mathsf{LK}_1^{?\rangle}$, then $P(\sigma')$ holds and for all $\rho\epsilon\sigma'$, $\vdash \rho(\Gamma)$ is derivable in LK_1. For the sake of brevity, we omit the general statement of completeness on an arbitrary domain, which is quite long to write.

As we shall see in Sect. 7, it is useful to have a "top element" $\top$ in Ψ_{d_0} with $\emptyset\epsilon\top$, which we feed to a proof-search procedure based on $\mathsf{LK}_1^{?\rangle}$, as the initial input constraint σ mentioned in the soundness and completeness theorems.

$$\begin{array}{|lll|}
\hline
\mathsf{Proj} & \forall\sigma\in\Psi_{d;?X},\ \forall t\forall\rho, & (\rho,?X\mapsto t)\,\epsilon\,\sigma \Rightarrow \rho\epsilon\sigma_{\downarrow} \\
\mathsf{Wit} & \forall\sigma\in\Psi_{d;?X},\ \forall\rho, & \rho\epsilon\sigma_{\downarrow} \Rightarrow (\rho,?X\mapsto f_{\sigma}(\rho))\,\epsilon\,\sigma \\
\mathsf{Meet} & \forall\sigma\sigma'\in\Psi_d,\ \forall\rho, & \begin{cases}\rho\epsilon\sigma\\ \rho\epsilon\sigma'\end{cases} \Leftrightarrow \rho\epsilon\,(\sigma\wedge\sigma') \\
\mathsf{PG} & \forall l,\forall\mathcal{A}, & \{\rho \mid \models\rho\,(\mathcal{A})\} = \bigcup_{\{\sigma\mid\ \models^l\mathcal{A}\rightarrow\sigma\}} \{\rho\mid\rho\epsilon\sigma\} \\
\hline
\end{array}$$

$$\begin{array}{|lll|}
\hline
\mathsf{Lift} & \forall\sigma\in\Psi_d,\forall\sigma'\in\Psi_{d;?X},\forall\rho, & (\rho,?X\mapsto f_{\sigma'}(\rho))\epsilon\sigma^{\uparrow} \Leftrightarrow \rho\epsilon\sigma \\
P_1 & \forall\sigma\in\Psi_{d;?X}, & P(\sigma)\Leftrightarrow P(\sigma_{\downarrow}) \\
P_2 & \forall\sigma\sigma'\in\Psi_d, & \begin{cases}P(\sigma)\\ \sigma\leqslant_{\epsilon}\sigma'\end{cases} \Rightarrow P(\sigma') \\
\mathsf{R2P} & \forall d,\forall\mathcal{A},\forall\sigma,\sigma'\in\Psi_d, & \sigma\rightarrow\mathrel{\models\!\!\rangle}^d\mathcal{A}\rightarrow\sigma' \Rightarrow \exists\sigma''\in\Psi_d,\begin{cases}\sigma'\simeq_{\epsilon}\sigma\wedge\sigma''\\ P(\sigma\wedge\sigma'')\\ \models^d\mathcal{A}\rightarrow\sigma''\end{cases} \\
\mathsf{P2R} & \forall d,\forall\mathcal{A},\forall\sigma,\sigma'\in\Psi_d, & \begin{cases}P(\sigma\wedge\sigma')\\ \models^d\mathcal{A}\rightarrow\sigma'\end{cases} \Rightarrow \exists\sigma''\in\Psi_d,\begin{cases}\sigma''\simeq_{\epsilon}\sigma\wedge\sigma'\\ \sigma\rightarrow\mathrel{\models\!\!\rangle}^d\mathcal{A}\rightarrow\sigma''\end{cases} \\
\hline
\end{array}$$

Figure 4. Full Axiomatisation

7 Implementation

Psyche is a platform for proof search, where a *kernel* offers an API for programming various search strategies as *plugins*, while guaranteeing the correctness of the search output [10]. Its architecture extensively uses OCaml's system of modules and functors. In order to modularly support theory-specific reasoning (in presence of quantifiers), the axiomatisation proposed in the previous sections was used to identify the signature and the specifications of *theory components*. In version 2.0 of Psyche [12], the kernel implements (the *focused* version of) System $\mathsf{LK}_1^{?\rangle}$, and a theory component is required to provide the implementation of the concepts

```
module type Theory = sig
 module Constraint: sig
   type t
   val topconstraint:t
   val proj : t -> t
   val lift : t -> t
   val meet : t -> t -> t option
   ...
 end
 val consistency :
  ASet.t -> (ASet.t,Constraint.t) stream
end
```

Theory component signature in Psyche 2.0

developed in the previous sections, as shown in the module type above. It provides a lift constraint structure in the form of a module `Constraint`, with a type for constraints, the projection and lift maps, as well as a top constraint (always satisfied) with which proof search will start. We also require a meet operation: While the theory of complete proofs in $\mathsf{LK}_1^{?\rangle}$ 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 $\mathcal{S}$, with input constraint σ_0 and output constraint σ_1; at some point the procedure may have to search again for a proof of $\mathcal{S}$ but with a different input constraint σ_0'. We can check whether the first proof can be re-used by simply checking whether $\sigma_0' \wedge \sigma_1$ is satisfiable. The `meet` function 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 `consistency`, which implements the constraint-refining predicate; `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* (see e.g. [10]).

While our axiomatisation immediately yields the specification for theory components, it does not provide instances and so far, the only (non-ground) instance implemented in Psyche is that of pure first-order logic (based on unification).

8 Related Works and Further Work

The sequent calculi developed in this paper for theory reasoning in presence of quantifiers, are akin to the free variable tableaux of [5] for total theory reasoning. But they use abstract constraints, instead of substitutions, and our foreground reasoner is able to propagate them across branches while being ignorant of their nature. This allows new theories to be treated by the framework, such as those satisfying quantifier elimination, like linear arithmetic. In this particular case, the asymmetric treatment of $\mathsf{LK}_1^{?\rangle}$ formalises an improvement, in the view of an effective implementation, over System PresPred$_S^C$ [14] for LIA. A novel point of our paper is to show that the propagation of substitutions in tableaux and the propagation of linear arithmetic constraints follow the same pattern, by describing them as two instances of an abstract constraint propagation mechanism.

Constraints have been integrated to various tableaux calculi: In the nomenclature proposed in Giese and Hähnle's survey [9], our approach is closest to *constrained formula tableaux* or *constrained branch tableaux* which propagate constraints between branches (rather than *constrained tableaux* which have a global management of constraints). But the *tableaux* calculi cited by [9] in these categories are for specific theories and logics (pure classical logic, equality, linear temporal logic or bunched implications), in contrast to our generic approach.

When classes of theories are generically integrated to automated reasoning with the use of constraints, as for the Model Evolution Calculus [3], these are usually described as first-order formulae over a particular theory's signature (as it is the case in [1,14] for LIA). Our abstract data-structures for constraints could be viewed as the semantic counter-part of such a syntactic representation, whose atomic construction steps are costless but which may incur expensive satisfiability checks by the background reasoner. Our semantic view of constraints, as shown in Section 7, more directly supports theory-tuned implementations where e.g. the meet and projection operations involve computation. Our specifications for theory-specific computation also seems less demanding than deciding the satisfiability of any constraint made of atoms (over the theory's signature), conjunction, negation, and existential quantification [3].

The semantic approach to constraints was explored by a rich literature in (Concurrent) Constraint Programming [15], but the applicability of constraint systems to programming usually leads to more demanding axioms as well (requiring e.g. complete lattices) and to a global management of constraints (with a global store that is reminiscent of *constrained tableaux*). Our local management of constraints allows for more 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.

In the case of ground theory reasoning, the field of SMT-solving has evolved powerful techniques for combining theories (see e.g. the unifying approach of [7]). A natural question is whether similar techniques can be developed in presence of quantifiers, combining constraint-producing or constraint-refining procedures. We did not provide such techniques here, but we believe our modular and abstract approach could be a first step towards that end, with our axiomatisation identifying what properties should be sought when engineering such techniques, i.e. serving as a correctness criterion.

Finally, SMT-solvers usually adopt a heuristic approach for handling quantifiers, often involving incomplete mechanisms, with slimmer theoretical foundations than for their ground reasoning core. A notable exception is a formalisation of *triggers* mechanisms by Dross et al. [6], which we hope to view as particular instances of our constraint systems. Moreover, the way in which triggers control the breaking of quantifiers appears as the kind of structured proof-search mechanisms that PSYCHE can specify (based on focusing).

Acknowledgements. This research was supported by ANR projects PSI and ALCOCLAN, as well as by DARPA under agreement number FA8750-12-C-0284. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, or the U.S. Government.

References

1. P. Baumgartner, A. Fuchs, and C. Tinelli. ME(LIA) – Model Evolution With Linear Integer Arithmetic Constraints. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Proc. of the the 15th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'08)*, volume 5330 of *LNCS*, pages 258–273. Springer-Verlag, Nov. 2008. 2, 15

2. P. Baumgartner, U. Furbach, and U. Petermann. A unified approach to theory reasoning. Technical report, Inst. für Informatik, Univ., 1992. 1
3. P. Baumgartner and C. Tinelli. Model evolution with equality modulo built-in theories. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Proc. of the 23rd Int. Conf. on Automated Deduction (CADE'11)*, volume 6803 of *LNCS*, pages 85–100. Springer-Verlag, July 2011. 14, 15
4. B. Beckert. Chapter 8: Rigid *E*-unification. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume I: Foundations. Calculi and Methods, pages 265–289. Kluwer Academic Publishers, 1998. 2
5. B. Beckert. Equality and other theories. In *Handbook of Tableau Methods*, pages 197–254. Kluwer Academic Publishers, 1999. 1, 2, 3, 4, 8, 9, 10, 14
6. C. Dross, S. Conchon, J. Kanig, and A. Paskevich. Reasoning with triggers. In P. Fontaine and A. Goel, editors, *10th Int. Work. on Satisfiability Modulo Theories, SMT 2012*, volume 20 of *EPiC Series*, pages 22–31. EasyChair, June 2012. 4, 15
7. H. Ganzinger, H. RueB, and N. Shankar. Modularity and refinement in inference systems. Technical Report SRI-CSL-04-02, SRI, 2004. 15
8. M. Giese. Proof search without backtracking using instance streams, position paper. In P. Baumgartner and H. Zhang, editors, *3rd Int. Work. on First-Order Theorem Proving (FTP), St. Andrews, Scotland, TR 5/2000 Univ. of Koblenz*, pages 227–228, 2000. 3, 8
9. M. Giese and R. Hähnle. Tableaux + constraints. In M. C. Mayer and F. Pirri, editors, *Proc. of the 16th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux'03)*, volume 2796 of *LNCS*, pages 37–42. Springer-Verlag, Sept. 2003. 14
10. 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, *Proc. of the 22nd Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux'13)*, volume 8123 of *LNCS*, pages 149–156. Springer-Verlag, Sept. 2013. 13, 14
11. 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. 1
12. Psyche: the Proof-Search factorY for Collaborative HEuristics. 2, 13
13. D. Rouhling, M. Farooque, S. Graham-Lengrand, J.-M. Notin, and A. Mahboubi. Axiomatic constraint systems for proof search modulo theories. Technical report, Laboratoire d'informatique de l'École Polytechnique - CNRS, Microsoft Research - INRIA Joint Centre, Parsifal & TypiCal - INRIA Saclay, France, Dec. 2014. 9, 11, 12
14. P. Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Proc. of the the 15th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'08)*, volume 5330 of *LNCS*, pages 274–289. Springer-Verlag, Nov. 2008. 2, 7, 14, 15
15. 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 333–352. ACM Press, Jan. 1991. 15
16. M. E. Stickel. Automated deduction by theory resolution. *J. of Automated Reasoning*, 1(4):333–355, 1985. 1