J. Goubault–Larrecq

Higher–Order Positive Set Constraints

Research Report LSV–02–6, Jul. 2002

Laboratoire Spécification et Vérification

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

http://www.lsv.ens−cachan.fr/Publis/
Research Report LSV−02−6, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, Jul. 2002

This is the long version of the paper with same title in *Proc. CSL'2002.*

Higher-Order Positive Set Constraints*

Jean Goubault-Larrecq

LSV/CNRS UMR 8643, ENS Cachan
61, av. du président-Wilson
94235 Cachan Cedex, France

Abstract. We introduce a natural notion of positive set constraints on simply-typed λ-terms. We show that satisfiability of these so-called positive higher-order set constraints is decidable in 2-NEXPTIME. We explore a number of subcases solvable in 2-DEXPTIME, among which higher-order definite set constraints, a.k.a., emptiness of higher-order pushdown processes. This uses a first-order clause format on so-called shallow higher-order patterns, and automated deduction techniques based on ordered resolution with splitting. This technique is then applied to the task of approximating success sets for a restricted subset of λ-Prolog, à la Frühwirth et al.

1 Introduction

It is well-known that a certain form of positive set constraints are subsets of the monadic class [3]. In turn, the monadic class can be decided by resolution [18]. More precisely, ordered resolution with splitting decides the satisfiability of positive set constraints in NEXPTIME, and this is optimal [19].

The point of this paper is to note that a similar construction adapts directly to define notions of positive set constraints for typed λ-terms up to $\beta\eta$-conversion. In particular we show that the satisfiability of positive higher-order set constraints is decidable. This hinges on the use of a clausal format with terms replaced by higher-order patterns [20], limited to depth one—the so-called *shallow patterns*. A natural application is computing upper approximations of success sets for a restricted class of λ-Prolog programs (*descriptive typing*), following a construction of [12].

The paper is organized as follows. We give a few preliminary definitions in Section 2 on λ-terms and Miller's higher-order patterns, including our *shallow* patterns: these will be the terms that we shall allow in clauses defining positive higher-order set constraints. We recall and adapt the form of ordered resolution we shall use in Section 3. The meat of the paper is Section 4, where we introduce higher-order automata, higher-order pushdown systems, and higher-order positive set constraints. We show that the latter are decidable in 2-NEXPTIME, and investigate several special cases of lower complexity. We apply this technique to typing a restricted class of λ-Prolog programs in Section 5, and conclude in Section 6.

* Partially supported by the ACI VERNAM, the RNTL project EVA and the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions".

Related Work. Set constraints traditionally denote relations between sets of ground first-order terms. Their main application is set-based analysis and type inference for functional, imperative and logic programming languages. See [22] for a recent survey—this is a very active field, and it would be too long to list all relevant papers. Even the number of variants of set constraints is daunting: definite, co-definite, positive or negative set constraints, with or without projection notably. The great majority is decidable in NEXPTIME, most of them are NEXPTIME-complete, while some of them, e.g. definite set constraints are DEXPTIME-complete [8]. Set constraints have even been generalized to deal with sets of terms modulo some equational theories [7], where decidability is obtained in the case of linear, shallow equational first-order theories. Our work can be seen as one addition to this category of set constraints, dealing with the theory of $\beta\eta$-equality in the typed λ-calculus—a theory that is far from shallow.

One particular relevant piece of work is [3], which shows that positive set constraints (with a limited form of projection, and with equality) are in close correspondence to monadic first-order formulas. We take this as a starting point to define a higher-order analogue. (In particular, we won't care to define a syntax resembling set constraints for the higher-order case, and will be content with just a clausal format.)

To decide clausal forms representing positive higher-order set constraints, we shall-make extensive use of resolution theorem proving techniques. A comprehensive reference is the handbook [23]. Using resolution to decide subclasses of first-order logic formulas was pioneered by Joyner [18] and by Maslov, see [10]. Standard refinements of resolution used in this area are hyperresolution and ordered refinements.

2 Preliminaries

Simple types, or types for short in this paper, are given by the grammar $\tau ::= b | \tau \to \tau$ where b ranges over a non-empty collection of *base types*. A *signature* Σ is a map from so-called *constants* $a, b, c, \ldots$ to types. A signature is *finite* iff its domain is finite. Fix a countably infinite set $\mathit{Var}_\forall$ of *universal variables* $x, y, z, \ldots$, each equipped with a unique type (let $\tau(x)$ be the type of x). Also, fix a countably infinite set $\mathit{Var}_\exists$ of *existential variables* $X, Y, Z, \ldots$, each equipped with a unique type (let $\tau(X)$ be the type of X). The set $T_\tau(\Sigma)$ of *preterms* s, t, u, $\ldots$, *of type* τ *on the signature* Σ is defined inductively by the rules:

$$\frac{\tau(X) = \tau}{X \in T_\tau(\Sigma)} \quad \frac{\tau(x) = \tau}{x \in T_\tau(\Sigma)} \quad \frac{\Sigma(c) = \tau}{c \in T_\tau(\Sigma)}$$

$$\frac{s \in T_{\tau_1 \to \tau_2}(\Sigma) \quad t \in T_{\tau_1}(\Sigma)}{st \in T_{\tau_2}(\Sigma)} \quad \frac{\tau(x) = \tau_1 \quad t \in T_{\tau_2}(\Sigma)}{\lambda x \cdot t \in T_{\tau_1 \to \tau_2}(\Sigma)}$$

Abbreviate $(\ldots((st_1)t_2)\ldots t_n)$ as $st_1 \ldots t_n$, and $\lambda x_1 \cdot \lambda x_2 \cdot \ldots \lambda x_n \cdot t$ as $\lambda x_1, \ldots, x_n \cdot s$.

The set of λ-*terms* of type τ is the set of all preterms in $T_\tau(\Sigma)$ whose free variables are existential. (This does not restrict generality, as we can always add λs in front in order to bind all universal variables.) A *ground* λ-term has no free existential variable.

All λ-terms that are α-equivalent (i.e., differ only in the name of bound variables) will be dealt with as though they were equal, using Barendregt's naming convention [4].

We consider the following rewrite rules:

$$(\beta)\ (\lambda x \cdot s)t \to s[x := t] \qquad\qquad (\eta)\ \lambda x \cdot tx \to t\ (x \text{ not free in } t)$$

where $s[x := t]$ denotes the standard capture-avoiding substitution. We write $\to_\beta$, $\to_\eta$, $\to_{\beta\eta}$ the corresponding one-step rewrite relations; if $\to$ is a rewrite relation, we write $\to^*$ its reflexive-transitive closure, $\to^+$ its transitive closure. We write $\approx_\beta$, $\approx_\eta$, $\approx_{\beta\eta}$ for the appropriate congruences.

The relations $\to_\beta$, $\to_\eta$, $\to_{\beta\eta}$ terminate on simply-typed terms [14]. Moreover, any (β)-normal preterm if of the form $\lambda x_1, \ldots, x_n \cdot h\, t_1 \ldots t_m$, where the *head* h is a constant, an existential variable or one of $x_1, \ldots, x_n$, and $t_1, \ldots, t_m$ are (β)-normal. If h is an existential variable, then $\lambda x_1, \ldots, x_n \cdot h\, t_1 \ldots t_m$ is *flexible*, otherwise it is *rigid*.

Define the η-long normal form $\eta_\tau[t]$ of any (β)-normal preterm $t \in T_\tau(\Sigma)$ by

$$\eta_\tau[\lambda x_1, \ldots, x_n \cdot h\, t_1 \ldots t_m] \mathrel{\hat{=}} \lambda x_1, \ldots, x_n, x_{n+1}, \ldots, x_p \cdot \\ h\ \eta_{\tau'_1}[t_1] \ldots \eta_{\tau'_m}[t_m]\ \eta_{\tau_{n+1}}[x_{n+1}] \ldots \eta_{\tau_p}[x_p]$$

where $\tau = \tau_1 \to \ldots \to \tau_n \to \tau_{n+1} \to \ldots \to \tau_p \to b$, b a base type, $x_{n+1}, \ldots, x_p$ are fresh universal variables of types $\tau_{n+1}, \ldots, \tau_p$ respectively, and t_1 has type $\tau'_1, \ldots, t_m$ has type τ'_m. This is a correct definition by double induction on the size of terms first, on the structure of types second. Then it is well-known [16] that any two λ-terms of the same type are $\beta\eta$-equal if and only if they have identical η-long β-normal forms; and that if s is η-long β-normal, and σ is a substitution mapping variables to η-long β-normal terms of the same types, then the η-long β-normal form of $s\sigma$ is its β-normal form (no need to perform η-expansion). This allows us to reason on η-long β-normal forms only, reasoning up to (β)-reduction and ignoring the (η)-rule entirely. From now on, we shall even abuse language and take terms to denote their η-long β-normal forms. In particular, when we talk about a variable x of type τ, we really mean $\eta_\tau[x]$.

Higher-order unification [16] is the following problem: given two λ-terms of the same type, find whether there is a substitution σ mapping variables to λ-terms of the same types such that $s\sigma \approx_{\beta\eta} t\sigma$. By the above remarks, taking s and t to be η-long β-normal, and restricting ourselves to substitutions mapping variables to η-long β-normal terms, it is equivalent to ask that $s\sigma$ and $t\sigma$ have the same β-normal form.

Miller's *patterns* are λ-terms where existential variables are only applied to distinct universal variables. For example, $\lambda x_1, x_2, x_3 \cdot X\, x_3 x_1$ is a pattern, but $\lambda x_1, x_2, x_3 \cdot X\,(X\, x_3 x_1)$ and $\lambda x_1, x_2, x_3 \cdot X\, x_1 x_1 x_2$ are not. It is well-known that higher-order unification of patterns is decidable in polynomial time, and that there is a most general unifier (*mgu*) if any unifier exists at all [20].

For convenience, we shall adopt Snyder and Gallier's convention [24] that $\overline{s}_m$ abbreviates the sequence $s_1 s_2 \ldots s_m$, or $s_1, s_2, \ldots, s_m$ depending on context. If π is a one-to-one mapping from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$, write $\overline{s}_{|\pi}$ the sequence $s_{\pi(1)} s_{\pi(2)} \ldots s_{\pi(k)}$.

To define higher-order automata, we shall need patterns that are not too deep:

Definition 1. *A* variable pattern *is a λ-term of the form $\lambda \overline{x}_m \cdot X \overline{x}_{|\pi}$, where π is a one-to-one mapping from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$.*

A shallow pattern *is either a variable pattern, or a* rigid shallow pattern*, i.e., a pattern of the form* $\lambda\overline{x}_m \cdot h\overline{u}_n$*, with* h *rigid, and where for every* i*,* $1 \leq i \leq n$*,* $\lambda\overline{x}_m \cdot u_i$ *is a variable pattern.*

The value of shallow patterns is given by Lemma 2 below.

Lemma 1. *Let* E *be a finite set of pairs* (s_i, t_i) *of terms of the same type,* $1 \leq i \leq n$*. If every* s_i *and every* t_i *is a variable pattern, then the simultaneous mgu of each pair, if any, maps variables to variable patterns.*

Proof. By induction on n. This is obvious if $n = 0$. Otherwise, consider s_1 and t_1. The simutaneous mgu of E is the composite of that, σ', of $\{(s_2, t_2), \ldots, (s_n, t_n)\}$ with that of $s_1\sigma'$ and $t_1\sigma'$. By induction hypothesis, σ'_1 maps variables to variable patterns. In particular, $s_1\sigma'$ and $t_1\sigma'$ are variable patterns. It therefore remains to show that the mgu of two variable patterns, if any, maps variables to variable patterns. We will then conclude by noticing that any composite of substitutions mapping variables to variable patterns again maps variables to variable patterns.

So let s and t be two variable patterns. Then either they have the same head or they do not.

- If they do have the same head, i.e., $s = \lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ and $t = \lambda\overline{x}_m \cdot X\overline{x}_{|\pi'}$, where π and π' are two one-to-one mappings from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$ (with the same k, since this is the same X is both cases). If σ is any unifier, then it must map X to some term of the form $\lambda\overline{y}_k \cdot u$, where the β-normal forms of $u[y_1 := x_{\pi(1)}, \ldots, y_k := x_{\pi(k)}]$ and $u[y_1 := x_{\pi'(1)}, \ldots, y_k := x_{\pi'(k)}]$ coincide. However, since u is β-normal already, $u[y_1 := x_{\pi(1)}, \ldots, y_k := x_{\pi(k)}] = u[y_1 := x_{\pi'(1)}, \ldots, y_k := x_{\pi'(k)}]$. Then it is easy to see that for every i, $1 \leq i \leq k$, if y_i is free in u, this implies that $x_{\pi(i)} = x_{\pi'(i)}$, hence that $\pi(i) = \pi'(i)$. In clear, the only free universal variables in u are the y_i's, $1 \leq i \leq k$, such that $\pi(i) = \pi'(i)$. Therefore every unifier of s and t must be an instance of

$$[X := \lambda\overline{y}_k \cdot Y y_{i_1} \ldots y_{i_\ell}] \tag{1}$$

  where $y_{i_1}, \ldots, y_{i_\ell}$ is the subsequence of those y_i's such that $\pi(i) = \pi'(i)$, $1 \leq i \leq k$, and Y is a fresh existential variable of the right type.
  Conversely, it is easy to see that (1) unifies s and t, so it is an mgu. Indeed, instantianting s by (1) and reducing, we get

$$\lambda\overline{x}_m \cdot Y y_{\pi(i_1)} \ldots y_{\pi(i_\ell)}$$

  while doing the same with t yields

$$\lambda\overline{x}_m \cdot Y y_{\pi'(i_1)} \ldots y_{\pi'(i_\ell)}$$

  which are equal precisely because $\pi(i_1) = \pi'(i_1), \ldots, \pi(i_\ell) = \pi'(i_\ell)$.
- If s and t have different heads, i.e., $s = \lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ and $t = \lambda\overline{x}_m \cdot X'\overline{x}_{|\pi'}$, where π and π' are two one-to-one mappings from $\{1, \ldots, k\}$, resp. $\{1, \ldots, k'\}$ to $\{1, \ldots, m\}$. If σ is any unifier, then it maps X to some $\lambda\overline{y}_k \cdot u$ and X' to some $\lambda\overline{y}'_{k'} \cdot u'$, such that $u[y_1 := x_{\pi(1)}, \ldots, y_k := x_{\pi(k)}] \approx_\beta u'[y'_1 := x_{\pi'(1)}, \ldots, y'_{k'} :=$

$x_{\pi'(k)}]$, hence $u[y_1 := x_{\pi(1)}, \ldots, y_k := x_{\pi(k)}] = u'[y'_1 := x_{\pi'(1)}, \ldots, y'_{k'} := x_{\pi'(k')}]$. Since π is one-to-one, it obtains $u = u'[y'_1 := x_{\pi'(1)}, \ldots, y'_{k'} := x_{\pi'(k')}][x_{\pi(1)} := y_1, \ldots, x_{\pi(k)} := y_k]$. Since the free universal variables of u are among $\overline{y}_k$, it must be that for every i', $1 \leq i' \leq k'$, such that $y'_{i'}$ is free in u', $x_{\pi'(i')}$ must equal some $x_{\pi(i)}$, $1 \leq i \leq k$. That is, the free universal variables of u' are of the form $y'_{\pi'^{-1}(j)}$ where $j \in \operatorname{Im}\pi \cap \operatorname{Im}\pi'$. Symmetrically, the free universal variables of u are of the form $y_{\pi^{-1}(j)}$ where $j \in \operatorname{Im}\pi \cap \operatorname{Im}\pi'$. So every unifier of s and t must be an instance of

$$[X := \lambda\overline{y}_k \cdot Y y_{\pi^{-1}(j_1)} \ldots y_{\pi^{-1}(j_\ell)}, X' := \lambda\overline{y}'_{k'} \cdot Y y'_{\pi'^{-1}(j_1)} \ldots y'_{\pi'^{-1}(j_\ell)}] \quad (2)$$

where $j_1, \ldots, j_\ell$ is the sequence of integers that comprise $\operatorname{Im}\pi \cap \operatorname{Im}\pi'$, and Y is a fresh existential variable of the right type. (The instance is obtained by $[Y := \lambda x_{j_1}, \ldots, x_{j_\ell} \cdot u[y_{\pi^{-1}(j_1)} := x_{j_1}, \ldots, y_{\pi^{-1}(j_\ell)} := x_{j_\ell}]$, or equivalently $[Y := \lambda x_{j_1}, \ldots, x_{j_\ell} \cdot u'[y'_{\pi'^{-1}(j_1)} := x_{j_1}, \ldots, y'_{\pi'^{-1}(j_\ell)} := x_{j_\ell}]$, since these two are equal.)

Conversely, it is easy to check that (2) unifies s and t, so that it is an mgu. Indeed, instantiating s by (2) and reducing, we get

$$\begin{aligned} &\lambda\overline{x}_m \cdot Y y_{\pi^{-1}(j_1)} \ldots y_{\pi^{-1}(j_\ell)}[y_1 := x_{\pi(1)}, \ldots, y_k := x_{\pi(k)}] \\ &= \lambda\overline{x}_m \cdot Y x_{\pi(\pi^{-1}(j_1))} \ldots x_{\pi(\pi^{-1}(j_\ell))} \\ &= \lambda\overline{x}_m \cdot Y x_{j_1} \ldots x_{j_\ell} \end{aligned}$$

Similarly, instantiating t by (2) and reducing, we get

$$\begin{aligned} &\lambda\overline{x}_m \cdot Y y'_{\pi'^{-1}(j_1)} \ldots y'_{\pi'^{-1}(j_\ell)}[y'_1 := x_{\pi'(1)}, \ldots, y'_{k'} := x_{\pi'(k')}] \\ &= \lambda\overline{x}_m \cdot Y x_{\pi'(\pi'^{-1}(j_1))} \ldots x_{\pi'(\pi'^{-1}(j_\ell))} \\ &= \lambda\overline{x}_m \cdot Y x_{j_1} \ldots x_{j_\ell} \end{aligned}$$

□

For example, the mgu of $(\lambda x_1, x_2, x_3 \cdot X x_3 x_1, \lambda x_1, x_2, x_3 \cdot X x_2 x_1)$ is $[X := \lambda y_1, y_2 \cdot Y y_2]$ where Y is a fresh free existential variable, and the mgu of $(\lambda x_1, x_2, x_3 \cdot X x_3 x_1, \lambda x_1, x_2, x_3 \cdot X' x_1 x_2)$ (where X and X' are distinct) is $[X := \lambda y_1, y_2 \cdot Y y_2, X' := \lambda y_1, y_2 \cdot Y y_1]$.

Lemma 2. *The mgus, if any, of two shallow patterns s and t are substitutions mapping variables to shallow patterns. Moreover, if both s and t are variable patterns, or both of them are rigid shallow patterns, then the mgus maps variables to variable patterns.*

Proof. Consider three cases:

- Both s and t are variable patterns. This is dealt with by Lemma 1.
- s and t are rigid shallow patterns. If s and t are unifiable, then they must have the same heads, so write s as $\lambda\overline{x}_m \cdot h\overline{u}_n$, t as $\lambda\overline{x}_m \cdot h\overline{v}_n$. Then any unifier of s and t must unify $\lambda\overline{x}_m \cdot u_1$ with $\lambda\overline{x}_m \cdot v_1$, …, $\lambda\overline{x}_m \cdot u_n$ with $\lambda\overline{x}_m \cdot v_n$. Conversely any simultaneous unifier of the latter unify s and t. Since every $\lambda\overline{x}_m \cdot u_i$ and every $\lambda\overline{x}_m \cdot v_i$, $1 \leq i \leq n$, is a variable pattern, we conclude by Lemma 1.

– s is a variable pattern, and t is a rigid shallow pattern. (The symmetric case is analogous.) Write s as $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$, where π is a one-to-one mapping from $\{1,\ldots,k\}$ to $\{1,\ldots,m\}$, and t as $\lambda\overline{x}_m \cdot h\overline{u}_n$.
If s and t unify, then let $s\sigma \approx_\beta t\sigma$. Let $\lambda\overline{y}_k \cdot u$ be $X\sigma$. Then $\lambda\overline{x}_m \cdot u[y_1 := x_{\pi(1)},\ldots,y_k := x_{\pi(k)}]$ is obtained by reduction from $s\sigma$, and is clearly β-normal. On the other hand, the β-normal form of $t\sigma$ is $\lambda\overline{x}_m \cdot h\overline{v}_n$, where v_i is the β-normal form of $u_i\sigma$, $1 \leq i \leq n$, so $u[y_1 := x_{\pi(1)},\ldots,y_k := x_{\pi(k)}] = h\overline{v}_n$. Since π is one-to-one, u must be $h\overline{v}_n[x_{\pi(1)} := y_1,\ldots,x_{\pi(k)} := y_k]$. Since the only free universal variables of u are among $\overline{x}_m$, the head h must either be a constant or some $x_{\pi(j)}$, $1 \leq j \leq k$ (so if it is not, then unification fails). In particular, σ must be an instance of

$$[X := \lambda\overline{y}_k \cdot h'(Y_1\overline{y}_k)\ldots(Y_n\overline{y}_k)] \tag{3}$$

where $h' \hat{=} h$ if h is a constant, $h' \hat{=} y_j$ if $h = x_{\pi(j)}$, and Y_1, …, Y_n are distinct fresh existential variables of the appropriate types. (We have made a slight abuse of language, e.g., we have written $Y_1\overline{y}_k$ instead of its η-long normal form $\lambda\overline{z}^1_{k_1} \cdot Y_1\overline{y}_k\overline{z}^1_{k_1}$.)
Then σ must be the composite of (3) with some unifier σ' of $\lambda\overline{x}_m \cdot h(Y_1\overline{x}_{|\pi})\ldots(Y_n\overline{x}_{|\pi})$ and t. (We use here the fact that if s and t unify, then X cannot occur free in t, by the *rigid path* argument [17].) By the previous case, any mgu of the latter maps variables to variable patterns. We obtain the mgus of s and t by precomposing with (3), which always yields a substitution mapping variables to shallow terms: X is mapped to a rigid shallow term, and the other existential variables are mapped to variable patterns. □

For example, the mgu of $(\lambda x_1 \cdot x_1(\lambda x_2, x_3 \cdot Xx_3x_1)(\lambda x_2, x_3 \cdot Xx_3x_1), \lambda x_1 \cdot x_1(\lambda x_2, x_3 \cdot Xx_2x_1)(\lambda x_2, x_3 \cdot X'x_1x_2))$ is $[X := \lambda y_1, y_2 \cdot Yy_2, X' := \lambda y_1, y_2 \cdot Yy_1]$.

3 Ordered Resolution in a First-Order Logic of Higher-Order Patterns

The technical tool we shall use in the sequel is resolution in a first-order logic with higher-order patterns. This is in the spirit of Joyner [18]. Although it would be possible to define a Tarskian semantics for this logic (use domains of individuals indexed by types, forming a Henkin applicative structure [2], then build a Tarskian semantics for first-order formulas atop these domains), we shall only be interested in Herbrand semantics here, where the domain of individuals of type τ is the set of ground λ-terms of type τ, up to $\beta\eta$-conversion—alternatively, the set of η-long β-normal forms of type τ. In fact, we will only consider clausal formats, where existential quantifiers are absent and universal quantifiers are implicit.

As far as syntax is concerned, fix a set of *predicate symbols* P, Q, R, …, each coming with an *arity*, which is a sequence of types. (Sometimes we shall call arity, ambiguously, just the number of arguments to predicates, constants or variables.) The *atoms* A, B, …, are $P(t_1,\ldots,t_n)$, where P is a predicate symbol of arity $\tau_1,\ldots,\tau_n$, t_1 is a λ-term of type τ_1, …, t_n is a λ-term of type τ_n. *Literals* L are either atoms A or negations of atoms $\neg A$. We also write $+A$ for A, $-A$ for $\neg A$. *Clauses* C are finite

disjunctions $L_1 \vee \ldots \vee L_p$ of literals. *Clause sets* S are conjunctions of clauses (possibly infinite, although our interest is in finite ones).

The semantics is as follows. Let the *Herbrand universe of type* τ, D_τ, be the set of all ground η-long normal forms of type τ. A *Herbrand interpretation* I is just a set of ground atoms. Herbrand interpretations are ordered by inclusion. A *valuation* ρ, giving values to each variable, is a substitution mapping each variable of type τ to a ground term of type τ (that is, in D_τ). The *value* of a term t under ρ is $t\rho$. We define the satisfaction relations $\models$ by:

$$I, \rho \models P(t_1, \ldots, t_n) \text{ iff } P(t_1\rho, \ldots, t_n\rho) \in I \tag{4}$$

$$I, \rho \models \neg A \text{ iff } I, \rho \not\models A \tag{5}$$

$$I \models L_1 \vee \ldots \vee L_p \text{ iff for every } \rho, \text{ for some } i,\ 1 \leq i \leq p,\ I, \rho \models L_i \tag{6}$$

$$I \models S \text{ iff for every } C \text{ in } S,\ I \models C \tag{7}$$

We say that a clause set S is *satisfiable* if and only if $I \models S$ for some Herbrand interpretation I. It is *unsatisfiable* otherwise.

Let us now restrict the set of terms we consider to higher-order patterns, so that every pair s, t of terms has exactly one mgu, as soon as they unify. Denote this mgu by $\mathrm{mgu}(s, t)$. Define the *resolution rule* [6] by:

$$\text{(Binary resolution)}\ \frac{C \vee A \quad \neg A' \vee C'}{(C \vee C')\sigma}\ \sigma = \mathrm{mgu}(A, A')$$

$$\text{(Factoring)}\ \frac{C \vee L \vee L'}{(C \vee L)\sigma}\ \sigma = \mathrm{mgu}(L, L')$$

where it is understood that, in binary resolution, the clauses $C \vee A$ and $\neg A' \vee C'$ are renamed so that they have no common free existential variable, and in factoring, two literals L and L' unify provided they have the same signs and the underlying atoms unify. Resolution is a *sound* deduction calculus, in the sense that if we can derive the empty clause $\Box$ from S by resolution, then S is unsatisfiable. In fact, every conclusion of the rules above is logically implied by the premises.

An ordering $>$ on η-long β-normal atoms is *stable* if and only if $A > B$ implies that the β-normal form of $A\sigma$ is greater than, in the sense of $>$, to the β-normal form of $B\sigma$, for every well-typed substitution σ mapping variables to η-long β-normal forms. *Ordered resolution* is the refinement of resolution where: in binary resolution, A is a $>$-maximal atom in $C \vee A$, A' is a $>$-maximal atom in $\neg A' \vee C'$; in factoring, letting $L \hat{=} \pm A$, then A is $>$-maximal in $C \vee L \vee L'$. The following is standard.

Proposition 1 (Completeness). *Ordered resolution w.r.t. $>$ is* complete*, provided that $>$ is stable. That is, given a finite set S of clauses, S is unsatisfiable if and only if the empty clause $\Box$ can be derived from S by ordered resolution.*

Proof. The if direction is soundness. Conversely, assume S unsatisfiable. Then by construction the (in general infinite) set S_0 of ground instances of clauses in S is unsatisfiable. This is equivalent to the fact that S_0 is propositionally unsatisfiable. By the compactness of propositional logic, S_0 contains a finite unsatisfiable subset S_1. Since

propositional ordered resolution is complete, there is a propositional ordered resolution deduction of $\square$ from S_1. This can then be lifted to a corresponding ordered resolution deduction of $\square$ from S. $\square$

Similarly, every form of, say, hyper-resolution is complete.

Given a clause C, we say that C is a *block* if and only if every pair of atoms in C has a common free existential variable. We can always write clauses as a disjunction $B_1 \vee \ldots \vee B_k$ of non-empty blocks that pairwise do not share any free existential variable. Moreover, this decomposition is unique [18].

In our decision procedures, we shall use an additional rule to split clauses into their blocks (on the right),

$$\text{(Splitting)} \quad \frac{C \vee C'}{C \mid C'}$$

where C and C' do not share any free existential variable, and are non-empty. This means that we shall split the current set of clauses in 2 sets, adding C to the first, and C' to the second. In other words, we define a tableau calculus in the following way. A *branch* is a finite clause set, and a *tableau* is a finite set of branches. A branch is *closed* if and only if it contains the empty clause $\square$. A tableau is *closed* if and only if all its branches are closed. We read a tableau as the disjunction of its branches.

As far as deduction is concerned, our tableau rules are as follows. We may either add a new clause to some branch by using resolution on this branch, or use splitting to replace some branch S of the tableau such that S contains $B_1 \vee \ldots \vee B_k$ by k branches $S \cup \{B_1\}$, ..., $S \cup \{B_k\}$. We write $T \Longrightarrow T'$ if we can go from tableau T to T' by applying one of these rules. This calculus is clearly sound, in the sense that if $T \Longrightarrow T'$ then T implies T'. So, if we can close some tableau by these deduction rules, then it is unsatisfiable: no Herbrand interpretation satisfies any of its branches. It is also clear that this tableau calculus is complete, even under some stable ordering restriction, because already the calculus without splitting is (Proposition 1).

Splitting can in fact be applied eagerly. That is, we may use the resolution rule on just those branches that contain only blocks without losing completeness. While this is well-known (see e.g., [10]), let us say quickly that the reason is that, just like Joyner's less efficient rule of *condensation*, ordered resolution, even in our higher-order setting, is complete by *semantic trees* [18]. If S is unsatisfiable, there is a finite closed semantic tree based on a finite subset of the Herbrand universe (the set of closed atoms); completeness of ordered resolution follows because this closed semantic tree is finite and can be shrinked by adding ordered resolvents between clauses that fail at leaves of the tree (see [6] for details). Since splitting replaces a failed clause C by subclauses that are failed at or above C, completeness is preserved.

4 Higher-Order Automata, Pushdown Systems and Positive Set Constraints

We define higher-order automata, higher-order pushdown systems, and higher-order positive set constraints as particular sets of clauses. The idea can be traced to [12].

We consider clauses built from unary predicate symbols and shallow patterns. Consider first Horn clauses of the form:

$$P_1(\lambda \overline{y}^1_{n_1} \cdot X_1 \overline{y}^1_{|\pi_1}), \ldots, P_k(\lambda \overline{y}^k_{n_k} \cdot X_k \overline{y}^k_{|\pi_k}) \supset P(\lambda \overline{x}_m \cdot h \overline{u}_n) \tag{8}$$

where $\overline{u}_n$ are variable patterns, h is rigid, and every X_i, $1 \leq i \leq k$, is free in $P(\lambda \overline{x}_m \cdot h \overline{u}_n)$. In the first order case, i.e. when $n_1 = \ldots = n_k = 0$, and if each u_i is X_i, (8) simplifies to

$$P_1(X_1), \ldots, P_k(X_k) \supset P(h(X_1, \ldots, X_k)) \tag{9}$$

This is a transition of a tree automaton: if t_1 is recognized at state P_1, ..., and t_n is recognized at state P_k, then $h(t_1, \ldots, t_k)$ is recognized at state P.

It seems that people familiar with classical automata theory are puzzled by this definition, and in particular by the fact that no definition of a *run* of a term against an automaton is given; we invite the puzzled reader to check that *positive hyper-resolution* derivations [6] (which are also unit derivations in the case of Horn clauses) are exactly *bottom-up runs* [13]: for every ground term $P(t)$, the positive hyper-resolution derivations of the unit clause $P(t)$ are exactly the runs of t that abut to state P, against the given tree automaton, considered bottom-up. On the other hand, *negative hyper-resolution* derivations are exactly the top-down runs. The theory of resolution theorem proving enables us to replace any complete deduction procedure (positive, negative hyper-resolution) by any other complete procedure; it seems that ordered resolution is the most powerful refinement of resolution in many practical cases.

Returning to clauses of the form (9), in case some X'_i occurs twice as an argument of h, we get *tree automata with equality constraints between brothers* [5]. The higher-order case enables us to write two arguments to h with the same head X'_i, but with permuted arguments to X'_i; for example, we may write transitions such as:

$$P_1(X_1) \supset P(\lambda x_1, x_2 \cdot h(X_1 x_1 x_2)(X_1 x_2 x_1))$$

which means that to be recognized at t, the term $\lambda x_1, x_2 \cdot h t_1 t_2$ should be such that $t_1 = t_2[x_1 := x_2, x_2 := x_1]$. This properly generalizes equality constraints.

In case some X'_i does not occur on the left-hand side of the implication, then X'_i is a don't care: any term of the same type as X'_i can instantiate X'_i. In the first-order case, it is always possible to describe the set of all terms by an automaton, and this would provide no added expressive power. In the higher-order case, these don't cares are a proper extension of tree automata, since there is no automaton recognizing all (ground) terms of a given type [9].

Sets of clauses of the form (8) will be called *higher-order automata*.

These can be enriched by, say, Horn clauses of the form:

$$P_1(\lambda \overline{y}^1_{n_1} \cdot X \overline{y}^1_{|\pi_1}), \ldots, P_k(\lambda \overline{y}^k_{n_k} \cdot X \overline{y}^k_{|\pi_k}) \supset P(\lambda \overline{y}_n \cdot X \overline{y}_{|\pi}) \tag{10}$$

with the same variable X in each atom. This corresponds to clauses of the form $P_1(X), \ldots, P_k(X) \supset P(X)$ in the first-order case. If $k = 1$, these are ϵ-transitions ("every term recognized at state P_1 must be recognized at state P, too"). If $k \geq 2$, we get conjunctive transitions ("if a term is recognized at states P_1, ..., P_k simultaneously, then it must be recognized at P"). Disjunctive transitions are handled naturally by having several ϵ-transitions reach the same state.

Sets of clauses of the form (8) or (10) will be called *alternating higher-order automata*. Notice again that the use of one-to-one mappings that shuffle bound variables around allows us to do a few more tricks than just intersections and unions.

Third, we may also consider Horn clauses of the form:

$$P(\lambda\overline{x}_m \cdot h\overline{u}_n) \supset P_1(\lambda\overline{y}_n \cdot X\overline{y}_{|\pi}) \tag{11}$$

where X is free in the rigid shallow term $\lambda\overline{x}_m \cdot h\overline{u}_n$. In the first-order case, this would simplify to $P(h(X_1, \ldots, X_n)) \supset P_1(X_i)$: this is a *pushdown transition*, which allows us to state that if some functional term $h(X_1, \ldots, X_n)$ is recognized at state P, then its ith argument must be recognized at state P_1. Again, the use of bound variables allows us to state slightly more in the higher-order case.

In general, we consider clauses of the following form:

Definition 2. *An* automatic clause *is any clause of the form*

$$\neg P_1(t_1) \vee \ldots \vee \neg P_m(t_m) \vee P_{m+1}(t_{m+1}) \vee \ldots \vee P_n(t_n) \tag{12}$$

where $0 \leq m \leq n$*, and* t_i*,* $1 \leq i \leq n$*, are shallow patterns such that:*

(i) *if every* t_i *is a variable pattern, then they all have the same head, say* X*;*

(ii) *otherwise, all the* t_i*'s that are not variable patterns are rigid shallow patterns* $\lambda\overline{x_i}_m \cdot h_i\overline{u_i}_n$*, which contain every free existential variable in the clause.*

In the first case, we call the clause an ϵ-block*. In the second case, it is a* complex clause*.*

A higher-order pushdown system *is a finite set of Horn automatic clauses. Finite sets of (non-Horn) automatic clauses are called* higher-order positive set constraints*.*

The reason why finite sets of automatic clauses are called higher-order set constraints is by analogy with the first-order case [3]. (Ordinary, first-order) set constraints are defined as follows. Let the *set expressions* be defined by the grammar:

$$e ::= \xi \mid 0 \mid 1 \mid e \cap e \mid e \cup e \mid \complement e \mid f(e_1, \ldots, e_n) \mid f_i^{-1}(e)$$

where f ranges over all function symbols (of arity n), and ξ ranges over a set of so-called *set variables*. In expressions of the form $f_i^{-1}(e)$, we require $1 \leq i \leq n$. Each set expression is interpreted, under a valuation that maps each set variable to a set of ground terms, as a set of ground terms. $\complement e$ denotes the complement of e, $f(e_1, \ldots, e_n)$ denotes the set of terms $f(t_1, \ldots, t_n)$ where t_1 is in e_1, ..., t_n is in e_n, and $f_i^{-1}(e)$ denotes the set of terms t_i such that $f(t_1, \ldots, t_i, \ldots, t_n)$ is in e for some terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$.

The *elementary constraints* are of the forms listed in the first column of the table on the right. Their translation as clauses is given in the second column—this may in fact be taken as the semantics of set constraints.

Set constraint	Automatic clause
$\xi \subseteq \eta$	$-\xi(X) \vee +\eta(X)$
$\xi \subseteq \eta \cup \zeta$	$-\xi(X) \vee +\eta(X) \vee +\zeta(X)$
$\xi \cap \eta \subseteq \zeta$	$-\xi(X) \vee -\eta(X) \vee +\zeta(X)$
$\xi \subseteq \complement\eta$	$-\xi(X) \vee -\eta(X)$
$\complement\xi \subseteq \eta$	$+\xi(X) \vee +\eta(X)$
$\xi \subseteq f(\xi_1, \ldots, \xi_n)$	$-\xi(f(X_1, \ldots, X_n)) \vee +\xi_1(X_1)$ $\ldots$ $-\xi(f(X_1, \ldots, X_n)) \vee +\xi_n(X_n)$ $-\xi(g(X_1, \ldots, X_m))$ (for all $g \neq f$)
$f(\xi_1, \ldots, \xi_n) \subseteq \xi$	$\bigvee_{i=1}^{n} -\xi_i(X_i) \vee +\xi(f(X_1, \ldots, X_n))$
$f_i^{-1}(\xi) \subseteq \eta$	$-\xi(f(X_1, \ldots, X_n)) \vee +\eta(X_i)$

This format of set constraints is *positive*—only inclusions $\subseteq$ can be dealt with, not negated inclusions $\not\subseteq$—and handles projections f_i^{-1} only partially—constraints $\xi \subseteq f_i^{-1}(\eta)$ require an extension of our format. This is just as in the first-order case investigated in [3]. Dealing with negative constraints and projections can be done by adding special constraints expressing that some variables ξ must be non-empty. This can be dealt with in a resolution format by considering clauses with additional *rigid existential* variables, which can be instantiated only once, just like the variables used in V-resolution [6] or in ordinary free-variable tableaux [11]; this will be treated elsewhere.

Finally, note that (higher-order) pushdown processes are just the higher-order analogue of *definite* set constraints.

4.1 Deciding Satisfiability of Higher-Order Positive Set Constraints

We now show that the satisfiability of higher-order positive set constraints is decidable.

To this end, we first need a stable ordering $>$ on shallow patterns such that any rigid shallow pattern s with X free in it is strictly greater than any variable pattern $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ with head X. (This is the natural extension of the subterm ordering in the first-order case.) Take $s > t$ if and only if the *rigid depth* $d(s)$ is greater than $d(t)$, where rigid depth is defined by: $d(\lambda\overline{x}_m \cdot h\overline{t}_n) = 1 + \max_{1 \leq i \leq n} d(t_i)$ if h is rigid (the maximum being 0 in case $n = 0$), and zero if h is a free existential variable. In the sequel, we shall do ordered resolution w.r.t. $>$, as defined in Section 3.

Lemma 3. *Every factor of an automatic clause is an automatic clause.*

Proof. Consider the clause $C \vee P(t) \vee P(t')$, and its factor $C\sigma \vee P(t\sigma)$, where $\sigma \hat{=} \operatorname{mgu}(t, t')$. (The case $C \vee \neg P(t) \vee \neg P(t')$ is entirely analogous.) If one of t, t' is a variable pattern, say with head X, and the other is a rigid shallow pattern, then by condition (ii) X is free in the rigid shallow term, hence this case is impossible (by Huet's *rigid path argument* [17]), as already noticed in the proof of Lemma 2. So by Lemma 2 σ maps variables to variable patterns. Therefore, the factor is an ϵ-block if the original clause was, and it is a complex clause otherwise. □

Lemma 4. *Every ordered binary resolvent of automatic clauses is either an automatic clause or a disjunction of ϵ-blocks that pairwise do not share free variables.*

Proof. Consider two automatic clauses $C \vee P(t)$ and $\neg P(t') \vee C'$.

If t and t' are both variable patterns, or both rigid shallow patterns, then the mgu σ if any of t and t' maps variables to variable patterns by Lemma 2. If C or C' contains any non-variable pattern at all, then it is easy to check that $(C \vee C')\sigma$ is a complex clause. Otherwise, $(C \vee C')\sigma$ is a disjunction of literals $\pm P(u)$ where u is a variable pattern, hence can be written as a disjunction of ϵ-blocks that pairwise do not share free variables. It may be the case that we do not have just a single ϵ-block, e.g. already in the first-order case, resolving on $-P_1(X_1) \vee -P_2(X_2) \vee +P(f(X_1, X_2))$ and $-P(f(X_1, X_2)) \vee +P_3(X_1)$ yields $-P_1(X_1) \vee -P_2(X_2) \vee +P_3(X_1)$.

If t is a variable pattern $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ and t' is a rigid shallow pattern $\lambda\overline{x}_m \cdot h\overline{u}_n$, then the mgu σ, if any, of t and t' maps X to some rigid shallow pattern $\lambda\overline{x}_m \cdot h\overline{v}_n$, and the free variables in C' to variable patterns, as shown in the proof of Lemma 2. Examining carefully this proof reveals that, additionally, each free variable of t' occurs as the head of some v_i, $1 \leq i \leq n$. Since $\neg A' \vee C'$ is a complex clause, by (ii) $X\sigma$ not only has head h, but also contains the heads of every v_i, $1 \leq i \leq n$, therefore every free variable of $C'\sigma$. Moreover, since t is a variable pattern, by the ordering condition every atom in C has a variable pattern as argument, so by (i) C is an ϵ-block. It follows that every literal of $C\sigma$ is of the form $\pm P(t)$ with t some rigid shallow pattern with head h containing every free variable of $C'\sigma$. So, if C is not empty or if C' contains some rigid shallow pattern, then the resolvent $(C \vee C')\sigma$ is a complex clause. Otherwise, it is trivially a disjunction of ϵ-blocks that pairwise do not share free variables, as above. The case where t' is a variable pattern and t is a rigid shallow pattern is analogous. □

Lemma 5. *Up to renaming of free existential variables, there are only finitely many automatic clauses on any given finite set of predicate symbols and constants.*

Proof. Let p be the number of predicate symbols, k the number of constants. There is an upper bound α on the number n such that $\tau_1 \to \ldots \to \tau_n \to b$ is a subtype of the arity of predicate symbols, or of the types of constants.

Let us first compute an upper bound $\psi(\alpha)$ on the number $\psi_X(m)$ of variable patterns $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ with head X. Letting X apply to at most k arguments, $\psi_X(m)$ is the number of one-to-one functions from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$, namely $m!/(m-k)!$. This is always at most $m!/(\lfloor m/2 \rfloor)! \leq m^m \leq \alpha^\alpha$. That is, we may take $\psi(\alpha) = \alpha^\alpha$.

Let us count the number of ϵ-blocks. Up to renaming, there is only one choice for the free variable X. We may choose each atom amongst $p\psi(\alpha)$ at most, so there are at most $4^{p\psi(\alpha)}$ ϵ-blocks.

Let us now count the number of complex clauses. Consider any rigid shallow pattern $\lambda\overline{x}_m \cdot h\overline{u}_n$, $n \leq \alpha$. Fix the names of free variables. There are $m + k$ choices for h, and for each argument u_i, there are at most $\psi(\alpha)$ choices of u_i for each variable head, and at most n possible variable heads for u_i. So there are at most $(m+k)(n\psi(\alpha))^n \leq (\alpha+k)(\alpha\psi(\alpha))^\alpha$ choices for a rigid shallow pattern on the given fixed variables. So the number of atoms of the form $P(\lambda\overline{x}_m \cdot h\overline{u}_n)$ is at most $p(\alpha+k)(\alpha\psi(\alpha))^\alpha$. Similarly to the previous case, the number of atoms with variable patterns as arguments is at most $p\psi(\alpha)$, times n (number of possible heads), i.e., at most $np\psi(\alpha)$. So, up to the name of free variables, complex clauses are build from at most $p\alpha\psi(\alpha) + p(\alpha+k)(\alpha\psi(\alpha))^\alpha \leq 2p(\alpha+k)(\alpha\psi(\alpha))^\alpha$ atoms. There are therefore at most $16^{p(\alpha+k)(\alpha\psi(\alpha))^\alpha}$ complex clauses.

So there are only finitely many automatic clauses: their number is at most doubly exponential in α, and simply exponential in p and k, namely $4^{p\psi(\alpha)} + 16^{p(\alpha+k)(\alpha\psi(\alpha))^\alpha}$. □

Because η-long forms have size greater than or equal to p, k and α, it follows that:

Theorem 1 (Decidability). *The satisfiability of higher-order positive set constraints is decidable in 2-NEXPTIME.*

4.2 Subcases of smaller complexity

A first slightly less complex subcase is that of *extended unary* higher-order positive set constraints, i.e., when all automatic clauses in S have at most one free existential variable. This includes the case of *unary* higher-order positive set constraints, where all rigid heads have arity at most 1. In turn this generalizes the case of unary set constraints [1] to the higher-order case.

Theorem 2. *The satisfiability of extended unary higher-order positive set constraints is decidable in* deterministic *double exponential time (2-DEXPTIME).*

Proof. Ordered resolution only produces clauses with at most one free existential variable again. Then splitting never occurs. □

Another remarkable subcase is that of *alternation-free* higher-order positive set constraints: this is defined as the case where every ϵ-block has at most 2 literals, and in every complex clause, every free existential variable X occurs at most once in some rigid shallow pattern, and at most once in some variable pattern. For example,

$$-P(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}) \vee +Q(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi'}) \tag{13}$$

$$-P_1(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_1}) \vee -P_2(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_2}) \tag{14}$$

$$\begin{aligned} -P_1(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_1}) \vee -P_2(\lambda\overline{y}_{m_2} \cdot X_2\overline{y}_{|\pi_2}) \\ \vee +Q(\lambda\overline{z}_m \cdot h(\lambda\overline{x}'_{m'_1} \cdot X_1\overline{z}_m\overline{x}'_{m'_1})(\lambda\overline{y}'_{m'_2} \cdot X_2\overline{z}_m\overline{y}'_{m'_2})) \end{aligned} \tag{15}$$

with $X_1 \neq X_2$, are alternation-free, but the following are not:

$$P_1(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_1}), P_2(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_2}) \supset Q(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi'}) \tag{16}$$

$$-P_1(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_1}) \vee -P_2(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_2}) \vee +Q(\lambda\overline{z}_m \cdot h(\lambda\overline{x}'_{m'_1} \cdot X_1\overline{z}_m\overline{x}'_{m'_1}) \tag{17}$$

$$-P_1(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_1}) \vee +Q(\lambda\overline{z}_m \cdot h(\lambda\overline{x}'_{m'_1} \cdot X_1\overline{z}_m\overline{x}'_{m'_1})(\lambda\overline{y}'_{m'_1} \cdot X_1\overline{z}_m\overline{y}'_{m'_1}) \tag{18}$$

Theorem 3. *The satisfiability of alternation-free higher-order positive set constraints is decidable in* deterministic *double exponential time (2-DEXPTIME).*

Proof. We first show that every clause that we get by resolution with eager splitting is alternation-free. Resolving two ϵ-blocks of at most 2 literals yields again an ϵ-block of

at most 2 literals. When we resolve two complex clauses, alternation-freeness implies that these clauses are of the form:

$$\pm_1 P_1(\lambda \overline{x}^1_{n_1} \cdot X_1 \overline{x}^1_{|\pi_1}) \vee \ldots \vee \pm_k P_k(\lambda \overline{x}^k_{n_k} \cdot X_k \overline{x}^k_{|\pi_k}) \vee + P(\lambda \overline{x}_m \cdot h \overline{u}_n) \quad (19)$$

$$\pm'_1 P'_1(\lambda \overline{y}^1_{n'_1} \cdot X'_1 \overline{y}^1_{|\pi'_1}) \vee \ldots \vee \pm'_{k'} P'_{k'}(\lambda \overline{y}^{k'}_{n'_{k'}} \cdot X'_{k'} \overline{y}^{k'}_{|\pi'_{k'}}) \vee - P(\lambda \overline{x}_m \cdot h \overline{u}'_n) \quad (20)$$

Then the mgu σ of the rigid shallow patterns $\lambda \overline{x}_m \cdot h \overline{u}_n$ and $\lambda \overline{x}_m \cdot h \overline{u}'_n$ maps free variables X of the first clause to variable patterns, in such a way that no two free variables are mapped to variable patterns with the same head. The resolvent then splits as ϵ-blocks with at most two literals ($\pm_i P_i(\ldots) \vee \pm'_j P'_j(\ldots)$ if $X_i\sigma$ has head X'_j, $\pm_i P_i(\ldots)$ if $X_i\sigma$ has a head that is none of the X'_js, $\pm'_j P'_j(\ldots)$ if X'_j is the head of no $X_i\sigma$).

Finally, when we resolve an ϵ-block C with a complex clause, say (19) (the case (20) is symmetric) then either C is of the form $-P(\lambda \overline{x}_m \cdot X \overline{x}_{|\pi})$, so the resolvent splits as unit clauses $\pm_i P_i(\lambda \overline{x}^i_{n_i} \cdot X_i \ldots)$; or C is a 2-literal ϵ-block, then the resolvent is again an alternation-free complex clause.

Now splitting only produces ϵ-blocks with at most two literals, and there are only exponentially many of them. Provided we remove subsumed clauses [6], this means that every branch of the tableau only splits exponential many times. Moreover, every split produces only polynomially many clauses. Implementing this double exponential time procedure with exponentially many splits by backtracking on a deterministic machine then produces a 2-DEXPTIME algorithm. □

A final 2-DEXPTIME subcase is the Horn case, which we explore in the next section. We believe that all the upper bounds we have given in this paper are tight.

4.3 Deciding Emptiness of Higher-Order Pushdown Systems, Or: the Horn Case

Let S be a higher-order pushdown system. Since S is a set of Horn clauses, if S has a model—a Herbrand interpretation I such that $I \models S$—then it has a least one. The argument is standard: if $(I_i)_{i \in \mathcal{I}}$ is a family of models, then $\bigcap_{i \in \mathcal{I}} I_i$ is a model again. Notice that if S is a set of *definite clauses*—with exactly one positive atom—, then S is satisfiable: the Herbrand interpretation containing every ground term is a model.

Definition 3 (Language). *Given a satisfiable higher-order pushdown system S, and a finite set F of unary predicates $P_1, \ldots, P_n$ (the* final states*), the* language $\mathcal{L}(S; F)$ *defined by S and F is the set of ground terms t such that $P_i(t)$ is in the least model of S for some i, $1 \leq i \leq n$.*

This language can be generated as the set of unit clauses that are ground instances of unit clauses $P_i(t)$ obtained by positive hyper-resolution (equivalently, by Prolog's T_P operator). Recall that positive hyper-resolution derivations are just bottom-up runs of the automaton S.

It is easy to see that $\mathcal{L}(S; F)$ is empty if and only if S plus the clauses $-P_1(X)$, $\ldots$, $-P_n(X)$ is satisfiable (where X stands for its η-long form): if it is satisfiable, then its least model satisfies $-P_i(X)$ for every i, hence cannot contain any ground term of the form $P_i(t)$, $1 \leq i \leq n$. Conversely, if $\mathcal{L}(S; F)$ is empty then no atom $P_i(t)$ is in its least Herbrand model, i.e., this model satisfies all the clauses $-P_i(X)$.

Theorem 4. *The satisfiability of sets of Horn automatic higher-order clauses is decidable in 2-DEXPTIME.*

Proof. Recall that a *negative* clause is a non-empty clause containing no atom of sign $+$ (i.e., its head is false). Let S be any fixed set of non-negative Horn automatic higher-order clauses. This is satisfiable: let I be its least Herbrand model, and let S^- be the set of all negative ϵ-blocks C (on the given signature) such that $S \cup \{C\}$ is unsatisfiable—equivalently, such that $I \not\models C$. We may compute S^- by noticing first that any splitting in any derivation from $S \cup \{C\}$, $C \in S^-$, must produce one non-negative ϵ-block C_0 (possibly $\Box$), plus negative ϵ-blocks $C_1, \ldots, C_n$. Branches stemming from C_i, $1 \leq i \leq n$, will be unsatisfiable iff $I \not\models C_i$, iff $C_i \in S^-$ (or $C \in S^-$). Consider then the S'-*splitting* rule, for each set S' of negative ϵ-blocks: this derives C_0 when splitting would derive the non-negative ϵ-block C_0 and the negative ϵ-blocks $C_1, \ldots, C_n$, provided the latter are in S'; otherwise it does not apply. Let $F(S')$ be the set of negative ϵ-blocks C that are either in S' or such that ordered resolution with S'-splitting derives $\Box$ from $S \cup \{C\}$. The discussion above shows that S^- is a fixpoint of F. In fact it is easy to see that S^- is the least fixpoint of F.

Let N be an upper bound on the number of Horn automatic clauses (this is doubly exponential). Since S^- contains at most N clauses, this fixpoint can be computed in at most N calls to F. F tests whether $S \cup \{C\}$ is unsatisfiable for at most N clauses C. Then this test is deterministic since S'-splitting is, and proceeds by generating at most N clauses. Hence computing S^- can be done in time $O(N^3)$.

Now given any set S_0 of Horn automatic higher-order clauses, by the same token ordered resolution with S^--splitting is complete, where S^- is computed as above (in a preprocessing step) from the subset S of non-negative clauses in S_0. This takes additional time $O(N)$. □

Corollary 1. *Given a satisfiable higher-order pushdown system S, and states P_1, …, P_n, it is decidable in 2-DEXPTIME whether the language of $(S; P_1, \ldots, P_n)$ is empty.*

Again, recall that this is only simply exponential in the number p of predicate symbols and k of constants, when α is fixed.

In the first-order case every pushdown process is equivalent to (recognizes the same language) as an ordinary tree automaton that we may compute in exponential time. Analogously, every satisfiable set S of Horn automatic clauses is equivalent to an *up-tree automaton*, that is, a set of *up-clauses*, of the form (8) or of the form $+P(\lambda\overline{x}^m \cdot X\overline{x}_{|\pi})$: saturate S by ordered resolution with S^--splitting as in the proof of Theorem 4, getting a set S' of clauses, then keep only the up-clauses in S'. This rests on the fact that whenever there is a positive hyperresolution derivation of the unit clause $P(t)$ from S', there is a positive hyperresolution derivation of some unit clause $P(s)$ using only *up-clauses* from S', with t an instance of s. (Exercise, using induction on the length of the derivation. Hint: take the first resolution step with some non-up-clause C_2; this must be preceded by a resolution step with some up-clause C_1; then this sequence of two steps may be replaced by one resolution step with a resolvent of C_1 and C_2, and this step derives a more general unit clause in general.)

5 Application: Towards Typing λ-Prolog Programs

Following [12], a natural use of our higher-order format is in computing upper approximations of success sets (*descriptive types*) of λ-Prolog programs [21]. This in fact works also for sets of non-Horn clauses over higher-order terms, but we don't consider this here. On the other hand, we consider for simplicity only a restricted subset of λ-Prolog programs, consisting of Horn clauses instead of general hereditary Harrop formulas, and using only rigid heads. (The fact that our typing discipline is simple types is inessential, since higher-order patterns are actually type-independent [20].) We are confident that the case of general λ-Prolog programs can be reduced to this simpler case, drawing inspiration from early Prolog implementations of λ-Prolog to define a translation from λ-Prolog to Horn clauses operating over typed λ-terms. However, existential quantifications cause some headaches, and probably require some approximation already. We prefer to leave this subject for future work.

Consequently, consider any set S of clauses, i.e., finite disjunctions of atoms $\pm P(t_1, \ldots, t_k)$, where P is a constant predicate of arity $\tau_1, \ldots, \tau_k$, and t_i is any λ-term of type τ_i, $1 \leq i \leq k$. Recall that, in this case at least, the success set of a logic program is its least Herbrand model.

We first make every predicate unary. Let o be a fresh base type. For every predicate P of arity $\tau_1, \ldots, \tau_k$ with $k \neq 1$, create a fresh constant f_P of type $\tau_1 \to \ldots \to \tau_k \to o$, and a fresh unary predicate $\tilde{P}$ of arity o. Then replace every atomic formula $P(u_1, \ldots, u_k)$ by $\tilde{P}(f_P(u_1, \ldots, u_k))$. Clearly Herbrand models of the original set of clauses are in one-to-one correspondence with Herbrand models of the transformed set.

We now define a series of transformations on sets of clauses with only unary predicates. While there is a term t that is not a shallow pattern in some clause $C = (C_0 \vee \pm P(t))$ of S:

1. if t is of the form $\lambda \overline{x}_m \cdot h \overline{u}_n$ with h rigid, x_i of type τ'_i, $1 \leq i \leq m$, and u_j of type τ''_j, $1 \leq j \leq n$, create n fresh unary predicates P_j and n free variables X_j of respective types $\tau'_1 \to \ldots \to \tau'_m \to \tau''_j$, $1 \leq j \leq n$, replace C by the $1 + n$ clauses:

$$C_0 \vee \pm P_1(\lambda \overline{x}_m \cdot u_1) \vee \ldots \vee \pm P_n(\lambda \overline{x}_m \cdot u_n) \tag{21}$$

$$\pm P(t') \vee \mp P_j(X_j) \qquad (1 \leq j \leq n) \tag{22}$$

   where t' is the rigid shallow pattern $\lambda \overline{x}_m \cdot h(X_1 \overline{x}_m) \ldots (X_n \overline{x}_m)$, and $\mp$ is the sign opposite to $\pm$ (each occurrence of $\pm$ denoting the *same* sign; recall also that we write terms up to η-expansion for brevity and clarity);
2. if t is of the form $\lambda \overline{x}_m \cdot X \overline{u}_n$ with X a free variable, let $x_{\pi(1)}, \ldots, x_{\pi(k)}$ be the free variables in the sequence $\overline{u}_n$, $1 \leq \pi(1) < \ldots < \pi(k) \leq m$, create a fresh variable Y, then replace C by the clause:

$$C_0 \vee \pm P(\lambda \overline{x}_m \cdot Y \overline{x}_{|\pi}) \tag{23}$$

If S' is obtained as above, write $S \rightsquigarrow S'$. The $\rightsquigarrow$ relation terminates: define a measure of atoms A by $\mu(P(t)) = 0$ if t is a shallow pattern, $\mu(P(t)) = d(t) + 1$, the rigid depth of t plus 1, otherwise; define $\mu(\pm_1 A_1 \vee \ldots \vee \pm_n A_n) = \sum_{i=1}^{n} \mu(A_i)$. Then $S \rightsquigarrow S'$ implies that the multiset of all $\mu(C)$, $C \in S$, is greater than that of all $\mu(C)$, $C \in S'$.

It is easy to check that if $S \rightsquigarrow S'$, then S' implies S: in case 1 this is because clause C is a (non-ordered) resolvent of the $1 + n$ generated clauses, in case 2 this is because C is an instance of clause (23). The interested reader may check that we can in fact improve slightly on items 1 and 2 above: in item 1 notably, we may produce any set of clauses that together produce C as a resolvent (in particular, we may take $X_i = X_j$ when $u_i = u_j$). The distinctive feature of this process compared to the first-order case is item 2, which involves some unavoidable loss of precision; the corresponding case in first-order Prolog programs would be when t is just a variable X and not a shallow pattern already, which is impossible.

Given any $\rightsquigarrow$-normal form S of a given set of clauses. S may fail to be a higher-order pushdown process: there might be a clause C in S of the form $C_0 \vee L_1 \vee L_2$, where L_1 and L_2 are two literals with distinct but non-disjoint sets of free existential variables. Then replace C by $C_0 \vee L_1 \vee C'_0 \vee L'_2$, where $C'_0 \vee L'_2$ is a renamed version of $C_0 \vee L_2$ whose free existential variables are not free in L_1. This process terminates, and results in a set of Horn clauses that are variable-disjoint disjunctions of automatic clauses. By a slight extension of the remark at the end of Section 4.3, this can be converted to a higher-order up-tree automaton (consisting only of up-clauses) in doubly exponential time. As in the first-order case, this automaton is a good candidate for a descriptive type of the values of free existential variables in succeeding goals.

6 Conclusion

We have defined a natural extension of positive set constraints to the case of higher-order terms. While the main idea is elementary (extend ordered resolution techniques used for the monadic class [18] to a higher-order analogue), one of the subtleties of the approach is to define a restriction of higher-order terms not only to a subcase where unification is decidable (Miller's patterns fit well), but also where applying most general unifiers to produce resolvents will only produce finitely many clauses.

A nice feature of this approach is that it yields as a by-product a natural notion of higher-order automata (up-automata), and a natural notion of approximation (typing) for the Horn, rigid-head subset of all λ-Prolog programs. This is in the line of [12].

We do not claim that any of these ideas alone is novel, however combining them appears to be new. On the other hand, this suggests numerous extensions. First, to other equational theories $\mathcal{E}$—just use $\mathcal{E}$-unification; this might be deceiving, however, and $\beta\eta$-equality of λ-terms is the only meaningful example that we know of where just ordered resolution with splitting provides a decision procedure—see [15] to realize how formidable a challenge just the case of associativity and commutativity is. A second and easier extension is to deal with higher-order analogues of other decidable classes of first-order formulas: just replace the shallow terms of [18] by our shallow patterns. This is probably easy; due to its relationship with set constraints and descriptive typing of logic programs, the higher-order analogue of the monadic class we have dealt with here is certainly the most useful such class.

References

1. A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In *CSL'93*, pages 1–17. Springer-Verlag LNCS 832, 1993.
2. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof*. Computer Science and Applied Mathematics. Academic Press, 1986.
3. L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83. IEEE Computer Society Press, 1993.
4. H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, Amsterdam, 1984.
5. B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In A. Finkel and M. Jantzen, editors, *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS '92)*, volume 577 of *LNCS*, pages 161–172, Berlin, Germany, Feb. 1992. Springer.
6. C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Computer Science Classics. Academic Press, 1973.
7. W. Charatonik. Set constraints in some equational theories. *Information and Computation*, 142(1):40–75, 1998.
8. W. Charatonik and A. Podelski. Set constraints with intersection. In G. Winskel, editor, *LICS'97*, pages 362–372, 1997.
9. H. Comon and Y. Jurski. Higher-order matching and tree automata. In M. Nielsen and W. Thomas, editors, *11th Workshop on Computer Science Logic (CSL'97)*, pages 157–176, Aarhus, 1997. Springer-Verlag LNCS 1414.
10. C. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. *Resolution Decision Procedures*, chapter 25, pages 1791–1849. Volume II of Robinson and Voronkov [23], 2001.
11. M. C. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, 1990.
12. T. Frühwirth, E. Shapiro, M. Y. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *LICS'91*, 1991.
13. F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer Verlag, 1997.
14. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.
15. J. Goubault-Larrecq and K. N. Verma. Alternating two-way AC-tree automata. In *CSL'02*, 2002. Submitted.
16. G. P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
17. G. P. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Université Paris VII, 1976. Thèse d'état.
18. W. H. Joyner Jr. Resolution strategies as decision procedures. *Journal of the ACM*, 23(3):398–417, July 1976.
19. H. R. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computers and System Sciences*, 21:317–353, 1980.
20. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
21. G. Nadathur and D. Miller. An overview of λ-Prolog. In R. Kowalski and K. Bowen, editors, *5th Intl. Conf. Logic Programming*, pages 810–827. MIT Press, 1988.
22. L. Pacholski and A. Podelski. Sets constraints—a pearl in research on constraints. In G. Smolka, editor, *3rd Intl. Conf. Principles and Practice of Constraint Programming (CP'97)*. Springer Verlag LNCS 1330, 1997.

23. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. North-Holland, 2001.
24. W. Snyder and J. Gallier. Higher order unification revisited: Complete sets of tranformations. *Journal of Symbolic Computation*, 8(1 & 2):101–140, 1989. Special issue on unification. Part two.