J. Goubault–Larrecq

Higher–Order Automata, Pushdown systems, and Set Constraints

Research Report LSV–01–9, Nov. 2001

Laboratoire Spécification et Vérification

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

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

http://www.lsv.ens–cachan.fr/Publis/
Research Report LSV–01–9, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, Nov. 2001

Higher-Order Automata, Pushdown Systems, and 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 automata, pushdown systems, and set constraints in increasing order of expressiveness, on simply-typed λ-terms. We show that recognizability of ground terms, and testing emptiness of higher-order automata and pushdown systems, as well as the satisfiability of higher-order set constraints, are decidable in non-deterministic double exponential time. This uses a first-order clause format on shallow higher-order patterns, and automated deduction techniques based on ordered resolution with splitting.

1 Introduction

The point of this note is to adapt the construction of [10], which defines a view of tree automata, pushdown systems and set constraints as particular sets of clauses to the case of simply-typed λ-terms modulo $\beta\eta$-conversions. The value is that it yields a particularly natural notion of automata on the latter higher-order terms, which in addition has the following properties: 1. recognizing a λ-terms modulo $\beta\eta$ is decidable; 2. deciding the emptiness of a higher-order tree automaton, resp. a pushdown system, is decidable; similarly, the satisfiability of higher-order set constraints is decidable. The idea is to use the same clausal format as in [10], except terms are replaced by Miller's higher-order patterns [14], limited at depth one—the so-called *shallow patterns*.

We give a few preliminary definitions in Section 2, then introduce shallow higher-order patterns in Section 3, and prove a few properties on the most general unifiers of shallow unifiers that we shall need later on. In Section 4, we recall a few mostly well-known results on the completeness of ordered resolution with splitting, adapted here to the case of a first-order logic with higher-order patterns in clause format. Restricting clauses to use only shallow patterns (as introduced in Section 3), subject to a few other technical conditions, allows us to define natural notions of higher-order automata, of higher-order pushdown systems, and of higher-order set constraints, in increasing order of expressiveness, in Section 5. Using the ordered resolution plus splitting format of Section 4, we deduce that the satisfiability of higher-order set constraints, and therefore also the emptiness of higher-order pushdown systems (including higher-order automata), is decidable in non-deterministic double exponential time. As a consequence, testing whether a given ground λ-term is in the language of a higher-order pushdown system is also decidable.

2 Preliminaries

Recall that *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 $Var_\forall$ 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 $Var_\exists$ 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, ..., of type τ on the signature Σ* is defined inductively by the rules:

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

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

We 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 all existential. (This does not restrict generality in the sequel, as we can always add λs in front in order to bind all universal variables.) A λ-term is *ground* iff it 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 [3]. We consider the following rewrite rules:

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

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.

It is well-known that $\to_\beta$, $\to_\eta$, $\to_{\beta\eta}$ terminate on simply-typed terms [9]. 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 called *flexible*, otherwise it is *rigid*.

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

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

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}, …, x_p are fresh universal variables of types τ_{n+1}, …, τ_p respectively, and t_1 has type τ'_1, …, 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 [11] 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 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 [11] 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 [14].

For convenience, we shall adopt Snyder and Gallier's convention [15] 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 $s_{\pi(1)} s_{\pi(2)} \cdots s_{\pi(k)}$.

3 Unification of Shallow Patterns

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 pattern of the form $\lambda\overline{x}_m \cdot h\overline{u}_n$, where for every i, $1 \leq i \leq n$, $\lambda\overline{x}_m \cdot u_i$ is a variable pattern. Call the latter* rigid shallow patterns.

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 $\{1,\dots,k\}$ to $\{1,\dots,m\}$ (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 $u[y_1 := x_{\pi(1)},\dots,y_k := x_{\pi(k)}]$ and $u[y_1 := x_{\pi'(1)},\dots,y_k := x_{\pi'(k)}]$ coincide. However, since u is β-normal already, $u[y_1 := x_{\pi(1)},\dots,y_k := x_{\pi(k)}] = u[y_1 := x_{\pi'(1)},\dots,y_k := x_{\pi'(k)}]$. 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

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

  where $y_{i_1},\dots,y_{i_\ell}$ 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

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

  while doing the same with t yields

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

  which are equal precisely because $\pi(i_1) = \pi'(i_1)$, …, $\pi(i_\ell) = \pi'(i_\ell)$.
- 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 $\{1,\dots,k\}$, resp. $\{1,\dots,k'\}$ to $\{1,\dots,m\}$. 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 $u[y_1 := x_{\pi(1)},\dots,y_k := x_{\pi(k)}] \approx_\beta u'[y'_1 := x_{\pi'(1)},\dots,y'_{k'} := x_{\pi'(k)}]$, hence $u[y_1 := x_{\pi(1)},\dots,y_k := x_{\pi(k)}] = u'[y'_1 := x_{\pi'(1)},\dots,y'_{k'} := x_{\pi'(k')}]$. Since π is one-to-one, it obtains $u = u'[y'_1 := x_{\pi'(1)},\dots,y'_{k'} := x_{\pi'(k')}][x_{\pi(1)} := y_1,\dots,x_{\pi(k)} := y_k]$. 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

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

  where j_1, …, j_ℓ 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

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

□

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 [12].) 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. □

4 Ordered Resolution in First-Order Logic with 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 [13]. 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 [1], 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. 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 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* [5] 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.

Given an ordering $>$ on η-long β-normal atoms, we say that it 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 $B\sigma$. *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 (usually infinite) set S_0 of ground instances of clauses in S is unsatisfiable. However it is easy to see that 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 $\Box$ from S_1. This can then be lifted to a corresponding ordered resolution deduction of $\Box$ from S. $\Box$

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 [13]. In our decision procedures, we shall use an additional rule to split clauses into their blocks:

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

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 $\Box$. 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 $S \cup \{B_1\}, \ldots, S \cup \{B_k\}$. 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).

There is a folk theorem that says that splitting can be applied eagerly. That is, we may use the resolution rule on just those branches that contain only blocks without losing completeness.

5 Higher-Order Automata, Pushdown Systems and Set Constraints

We define higher-order automata, higher-order pushdown systems, and higher-order set constraints as particular sets of clauses. The idea dates back to Fribourg and Veloso-Peixoto [8], and to Podelski and Charatonik [6].

5.1 Definitions

We consider clauses build 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, this simplifies to $P_1(X_1), \ldots, P_k(X_k) \supset P(h(X'_1, \ldots, X'_{k'}))$. In the special case $P_1(X_1), \ldots, P_k(X_k) \supset P(h(X_1, \ldots, X_k))$, 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. In case some X'_i occurs twice as an argument of h, we get *tree automata with equality constraints between brothers* [4]. In case some X'_i does not occur on the left-hand side of the implication, then this is a don't care. 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 [7].

Sets of clauses of the form (8) will be called *higher-order automata*. Note that the versatility offered by the one-to-one mappings $\pi_1, \ldots, \pi_k$, and those hidden in $u_1, \ldots, u_n$ allows us to permute and drop some bound arguments, which we could not do at the first order.

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

$$P_1(\lambda \overline{y}^1_{n_1} \cdot X \overline{y}^1_{|\pi_1}), \ldots, P_k(\lambda \overline{y}^k_{n_k} \cdot X \overline{y}^k_{|\pi_k}) \supset P(\lambda \overline{y}_n \cdot X \overline{y}_{|\pi}) \tag{9}$$

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 $P_1, \ldots, P_k$ 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 (9) 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:

$$P(\lambda \overline{x}_m \cdot h\overline{u}_n) \supset P_1(\lambda \overline{y}_n \cdot X\overline{y}_{|\pi}) \tag{10}$$

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*

$$\neg P_1(t_1) \vee \ldots \vee \neg P_m(t_m) \vee P_{m+1}(t_{m+1}) \vee \ldots \vee P_n(t_n) \tag{11}$$

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* $\lambda \overline{x}_m \cdot h\overline{u}_n$*, and containing 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 any finite set of Horn automatic clauses. Finite sets of (not necessarily Horn) automatic clauses are called* higher-order set constraints*.*

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

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

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. $\overline{e}$ 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 some term $f(t_1, \ldots, t_i, \ldots, t_n)$ is in e.

The *elementary constraints* are $e_1 \subseteq e_2$, where e_1 and e_2 are set expressions. *Set constraints* K are finite conjunctions of elementary constraints. It is easy to translate the

semantics of set constraints into sets of automatic (first-order) clauses by first associating with each set expression e occurring as sub-expression of expressions in K a distinct unary predicate P_e, and writing out the definition of P_e as clauses as shown in Figure 1. Then, for each inclusion $e_1 \subseteq e_2$, we generate the clause $P_{e_1}(X) \supset P_{e_2}(X)$. Note that this translation does not use every feature that are available in automatic clauses, even at first-order; notably, this does not use any equality constraint between brothers. It should be clear that any set constraint is satisfiable if and only if the corresponding translation is satisfiable.

e	definition
ξ	–
0	$\neg P_0(X)$
1	$P_1(X)$
$e_1 \cap e_2$	$P_{e_1}(X), P_{e_2}(X) \supset P_e(X)$
	$P_e(X) \supset P_{e_1}(X)$
	$P_e(X) \supset P_{e_2}(X)$
$e_1 \cup e_2$	$P_{e_1}(X) \supset P_e(X)$
	$P_{e_2}(X) \supset P_e(X)$
	$P_e(X) \supset P_{e_1}(X) \vee P_{e_2}(X)$
$\overline{e}_1$	$P_{e_1}(X) \vee P_e(X)$
	$\neg P_{e_1}(X) \vee \neg P_e(X)$
$f(e_1, \ldots, e_n)$	$P_{e_1}(X_1), \ldots, P_{e_n}(X_n) \supset P_e(f(X_1, \ldots, X_n))$
	$P_e(f(X_1, \ldots, X_n)) \supset P_{e_1}(X_1)$
	$\ldots$
	$P_e(f(X_1, \ldots, X_n)) \supset P_{e_n}(X_n)$
	$\neg P_e(g(X_1, \ldots, X_m))$ (for all $g \neq f$)
$f_i^{-1}(e_1)$	$P_e(f(X_1, \ldots, X_n)) \supset P_{e_1}(X_i)$
	$P_{e_1}(X_i) \supset P_e(f(X_1, \ldots, X_n))$

Fig. 1. Set constraints as first-order automatic clauses

5.2 Deciding Satisfiability of Higher-Order Set Constraints

We now show that the satisfiability of higher-order 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 $\lambda\overline{x}_m.X\overline{x}_{|\pi}$ of the same type, with head X. (This is the natural extension of the subterm ordering in the first-order case.) Take $s > t$ if and only if, for every well-typed substitution σ such that $s\sigma$ and $t\sigma$ are ground, the depth of the η-long β-normal form of $s\sigma$ is greater than that of $t\sigma$, for example. We define the *depth* if $\lambda\overline{x}_m \cdot h\overline{t}_n$ as 1 plus the maximum depth of t_i, $1 \leq i \leq n$ (1 if $n = 0$). In the sequel, we fix such an ordering $>$, and do ordered resolution w.r.t. $>$, as defined in Section 4.

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 (rigid path condition), 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 case a single ϵ-block, e.g. already in the first-order case, resolving on $P_1(X_1), P_2(X_2) \supset P(f(X_1, X_2))$ and $P(f(X_1, X_2)) \supset P_3(X_1)$ yields $P_1(X_1), P_2(X_2) \supset P_3(X_1)$.

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 count the number of ϵ-blocks. Up to renaming, there is only one choice for the free variable X. In variable patterns $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$, there is no choice as to the names of bound variables, up to α-renaming. If X has arity k, then there are $m!/(m-k)!$ choices for the one-to-one function π from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$. This is always at most $m!/(\lfloor m/2 \rfloor)!$, which is bounded above by m^m. Since $m \leq \alpha$, we have at most

α^α variable patterns with head X as arguments to predicate symbols. So we can choose our atoms from a set of at most $p\alpha^\alpha$ atoms. Therefore, there are at most $4^{p\alpha^\alpha}$ 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 α^α choices of u_i for each variable head, and at most n possible variable heads for u_i. So there are at most $(m+k)n^n\alpha^{n\alpha}$ choices for a rigid shallow pattern on the given fixed variables. Since $n \leq \alpha$, there are at most $(m+k)\alpha^{\alpha+\alpha^2}$ possible patterns on the given fixed variables. Since $m \leq \alpha$, this is at most $(\alpha+k)\alpha^{\alpha+\alpha^2}$. So the number of atoms of the form $P(\lambda\overline{x}_m \cdot h\overline{u}_n)$ is at most $p(\alpha+k)\alpha^{\alpha+\alpha^2}$. Similarly to the previous case, the number of atoms with variable patterns as arguments is at most $p\alpha^\alpha$, times n (number of possible heads), i.e., at most $np\alpha^\alpha$. So, up to the name of free variables, complex clauses are build from at most $p\alpha\alpha^\alpha + p(\alpha+k)\alpha^{\alpha+\alpha^2} \leq 2p(\alpha+k)\alpha^{\alpha+\alpha^2}$ atoms. There are therefore at most $16^{p(\alpha+k)\alpha^{\alpha+\alpha^2}}$ complex clauses.

To conclude, there are only finitely many automatic clauses, namely at most

$$4^{p\alpha^\alpha} + 16^{p(\alpha+k)\alpha^{\alpha+\alpha^2}}$$

□

This is exponential in the number of predicates and constants, and doubly exponential in the maximal arity α of types. An interesting subcase is when all variable patterns $\lambda\overline{x}_m \cdot Xx_{|\pi}$ are such that π is monotonic. We call the resulting higher-order set constraints *non-permuting*. Then, given that X has arity k, there are at most $\binom{m}{k}$ strictly increasing functions π, and this is at most $m^{m/2} \leq \alpha^{\alpha/2}$, yielding at most

$$4^{p\alpha^{\alpha/2}} + 16^{p(\alpha+k)\alpha^{\alpha+\alpha^2/2}}$$

clauses. This is still doubly exponential. If in variable patterns we always have $k = m$ and π is the identity function, then we can only choose among p atoms in ϵ blocks, once the name of the unique variable head is fixed. If additionally variable heads of arguments to rigid shallow pattern always occur in the same order in any given complex clause, then the only choice in rigid shallow patterns of complex clauses is in the predicate symbol and the rigid head, allowing for at most $p(\alpha+k)$ possibilities. Therefore, in this case, we have at most

$$4^p + 4^{p+p(\alpha+k)}$$

clauses. This is only singly exponential, and includes the case of first-order set constraints.

It follows that:

Theorem 1 (Decidability). *The satisfiability of higher-order set constraints is decidable in non-deterministic double exponential time.*

If α is bounded above by a constant (or even by $\sqrt{\log(p^i k^j)}$ for some $i, j \geq 0$), then this is decidable in non-deterministic single exponential time. It is well-known that the problem is NEXPTIME-hard, since it contains the problem of satisfiability of first-order set constraints [2].

5.3 Deciding Emptiness of Higher-Order Pushdown Systems

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 $\cap_{i\in\mathcal{I}} I_i$ 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 of unary predicates $P_1, \ldots, P_n$ (the* final states*), the* language *defined by $(S; P_1, \ldots, P_n)$ is the set of ground terms t such that $P_i(t)$ is in the least model of S.*

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

Proof. We claim that this language is non-empty if and only if S plus the clauses $\neg P_1(X_1), \ldots, \neg P_n(X_n)$, is unsatisfiable. The proposition will then follow from Theorem 1.

If S plus the clauses $\neg P_1(X_1), \ldots, \neg P_n(X_n)$ is satisfiable, then let I be some model of it. Clearly, since I satisfies the clauses $\neg P_1(X_1), \ldots, \neg P_n(X_n)$, there is no term t such that any of $P_1(t), \ldots, P_n(t)$ is in I. Since I is a model of S, too, it contains its smallest model. Therefore the smallest model contains no $P_i(t)$, $1 \leq i \leq n$: the language of $(S; P_1, \ldots, P_n)$ is empty.

Conversely, if S plus the clauses $\neg P_1(X_1), \ldots, \neg P_n(X_n)$ is unsatisfiable, in particular the smallest model I of S does not satisfy it. So there is some i, $1 \leq i \leq n$, such that I does not satisfy $\neg P_i(X_i)$. Therefore there is a ground term t such that $I \models P_i(t)$. So the language of $(S; P_1, \ldots, P_n)$ contains t, and is therefore not empty. □

Lemma 6. *Given a ground λ-term t, there is a satisfiable higher-order pushdown system S and a unary predicate Q such that the language of $(S; Q)$ is exactly $\{t\}$.*

Proof. By induction on the depth of t, taking t to be η-long β-normal. Write t as $\lambda\overline{x}_m \cdot h\overline{s}_n$, where h is a constant or among $\overline{x}_m$. For each i, $1 \leq i \leq n$, by induction hypothesis, there is a satisfiable higher-order pushdown system S_i and a unary predicate Q_i such that the language of $(S_i; Q_i)$ is exactly $\{\lambda\overline{x}_m \cdot s_i\}$. Without loss of generality, assume that no two S_i's share any predicate symbol. Then create a fresh unary predicate symbol Q, and let S be $\bigcup_{i=1}^{n} S_i$ plus the clauses:

$$\begin{aligned} Q_1(X_1), \ldots, Q_n(X_n) &\supset Q(\lambda\overline{x}_m \cdot h(X_1\overline{x}_m)\ldots(X_n\overline{x}_m)) \\ Q(\lambda\overline{x}_m \cdot h(X_1\overline{x}_m)\ldots(X_n\overline{x}_m)) &\supset Q_1(X_1) \\ &\cdots \\ Q(\lambda\overline{x}_m \cdot h(X_1\overline{x}_m)\ldots(X_n\overline{x}_m)) &\supset Q_n(X_n) \end{aligned}$$

The result is then clear. □

Proposition 3. *Given a satisfiable higher-order pushdown system S, and final states $P_1, \ldots, P_n$, it is decidable whether the ground λ-term t is in the language of $(S; P_1, \ldots, P_n)$.*

Proof. Without loss of generality, we may assume that the arity of each P_i is the type of t. (All other predicates contribute nothing in recognizing the term t.) Now use Lemma 6 to create a new clause set S' such that the language of $(S'; Q)$ is exactly t. Without loss of generality, we may assume that the set of predicates in S' is disjoint from that of S. Then build the set $S'' \hat{=} S \cup S' \cup \{\neg Q(X) \vee \neg P_1(X), \ldots, \neg Q(X) \vee \neg P_n(X)\}$. We claim that this clause set is unsatisfiable if and only if t is in the language of $(S; P_1, \ldots, P_n)$.

If S'' is satisfiable, then let I be a model of S''. Since $I \models S'$, the only ground term $Q(t')$ in I is $Q(t)$. Since I is a model of $\neg Q(X) \vee \neg P_i(X)$, for every i, $1 \leq i \leq n$, then $I \models \neg Q(t) \vee \neg P_i(t)$, so since $Q(t) \in I$, $P_i(t)$ is not in I. Since no $P_i(t)$ is in I, and I is a model of S, no $P_i(t)$ is in the least model of S either, so t is not in the language of $(S; P_1, \ldots, P_n)$.

Conversely, if S'' is unsatisfiable, then build a Herbrand interpretation I as follows. On the predicate symbols that comprise S, take the least model of S. (More precisely, for every predicate symbol P in S, let the ground atoms $P(t)$ in I be those that are in the least model of S.) On the predicate symbols that comprise S', take any of its models, which exist by Lemma 6. Since I does not satisfy S'', but satisfies both S and S', there must be an i, $1 \leq i \leq n$, such that $I \not\models \neg Q(X) \vee \neg P_i(X)$. So there is a ground term t' such that $I \models Q(t')$ and $I \models P_i(t')$. By Lemma 6, t' must be t. So $I \models P_i(t)$. Therefore t is in the language of $(S; P_1, \ldots, P_n)$. □

All decision procedures work in non-deterministic double exponential time, but are almost surely less complex. We conjecture that the test for emptiness of higher-order pushdown systems is in deterministic double exponential time, as well as language membership.

References

1. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof*. Computer Science and Applied Mathematics. Academic Press, 1986.
2. 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.
3. 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.
4. B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In A. Finkel and M. Jantzen, editors, *Proceedings of Symposion on Theoretical Aspects of Computer Science (STACS '92)*, volume 577 of *LNCS*, pages 161–172, Berlin, Germany, Feb. 1992. Springer.
5. C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Computer Science Classics. Academic Press, 1973.
6. W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *TACAS'98*, pages 358–375. Springer Verlag LNCS 1384, 1998.
7. 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.
8. L. Fribourg and M. Veloso Peixoto. Automates concurrents à contraintes. *Technique et Science Informatique*, 13(6):837–866, 1994.

9. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.
10. J. Goubault-Larrecq. Examen de démonstration automatique. `http://www.lsv.ens-cachan.fr/~goubault/autoans.ps`, Apr. 2001.
11. G. P. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
12. G. P. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Université Paris VII, 1976. Thèse d'état.
13. W. H. Joyner Jr. Resolution strategies as decision procedures. *Journal of the ACM*, 23(3):398–417, July 1976.
14. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
15. 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.