

E-generalization using grammars

Jochen Burghardt

Institute for Computer Architecture and Software Technology, Berlin, Germany

Received 11 January 2003

Available online 14 March 2005

Abstract

We extend the notion of anti-unification to cover equational theories and present a method based on regular tree grammars to compute a finite representation of *E*-generalization sets. We present a framework to combine Inductive Logic Programming and *E*-generalization that includes an extension of Plotkin's *lgg* theorem to the equational case. We demonstrate the potential power of *E*-generalization by three example applications: computation of suggestions for auxiliary lemmas in equational inductive proofs, computation of construction laws for given term sequences, and learning of screen editor command sequences.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Equational theory; Generalization; Inductive logic programming

1. Introduction

Many learning techniques in the field of symbolic Artificial Intelligence are based on adopting the features common to the given examples, called *selective induction* in the classification of [14], for example. Syntactical *anti-unification* reflects these abstraction techniques in the theoretically elegant domain of term algebras.

In this article, we propose an extension, called *E*-anti-unification or *E*-generalization, which also provides a way of coping with the well-known problem of representation change [12,29]. It allows us to perform abstraction while modeling equivalent representations using appropriate equations between terms. This means that all equivalent representations are considered simultaneously in the abstraction process. Abstraction becomes insensitive to representation changes.

In 1970, Plotkin and Reynolds [30,31,33] introduced the notion of (syntactical) anti-unification of terms as the dual operation to unification: while the latter computes the most general common specialization of the given terms, if it exists, the former computes the most special generalization of them, which always exists and is unique up to renaming. For example, using the usual 0- s representation of natural numbers and abbreviating $s(s(0))$ to $s^2(0)$, the terms $0 * 0$ and $s^2(0) * s^2(0)$ anti-unify to $x * x$, retaining the common function symbol $*$ as well as the equality of its arguments.

While extensions of unification to equational theories and classes of them have been investigated [17,20,36], anti-unification has long been neglected in this respect, except for the theory of associativity and commutativity [32] and so-called *commutative theories* [1]. For an arbitrary equational theory E , the set of all E -generalizations of given terms is usually infinite. Heinz [2,22] presented a specially tailored algorithm that uses regular tree grammars to compute a finite representation of this set, provided E leads to regular congruence classes. However, this work has never been internationally published. In this paper, we try to make up for this neglect, giving an improved presentation using standard grammar algorithms only, and adding some new theoretical results and applications (Sections 4, 5.3 below).

In general, E -anti-unification provides a means to find correspondences that are only detectable using an equational theory as background knowledge. By way of a simple example, consider the terms 0 and $s^4(0)$. Anti-unifying them purely syntactically, without considering an equational theory, we obtain the term y , which indicates that there is no common structure. If, however, we consider the usual defining equations for $(+)$ and $(*)$, see Fig. 1 (left), the terms may be rewritten nondeterministically as shown in Fig. 1 (right), and then syntactically anti-unified to $x * x$ as one possible result. In other words, it is recognized that both terms are quadratic numbers.

Expressed in predicate logic, this means we can learn a definition $p(x * x)$ from the examples $p(0)$ and $p(s^4(0))$. Other possible results are $p(s^4(0) * x)$, and, less meaningfully, $p(x + y * z)$. The closed representation of generalization sets by grammars allows us to filter out generalizations with certain properties that are undesirable in a given application context.

After some formal definitions in Section 2, we introduce our method of E -generalization based on regular tree grammars in Section 3 and briefly discuss extensions to more sophisticated grammar formalisms. As a first step toward integrating E -generalization into Inductive Logic Programming (ILP), we provide, in Section 4, theorems for learning determinate or nondeterminate predicate definitions using atoms or clauses. In Section 5, we present applications of determinate atom learning in different areas, including induc-

1.	$x + 0 = x$	0	$=_E$	$0 * 0$
2.	$x + s(y) = s(x + y)$	$s^4(0)$	$=_E$	$s^2(0) * s^2(0)$
3.	$x * 0 = 0$	<u>syn. anti-un.</u>		<u>syn. anti-un.</u>
4.	$x * s(y) = x * y + x$	<u>y</u>		<u>$x * x$</u>

Fig. 1. Equations defining $(+)$ and $(*)$ (left). E -generalization of 0 and $s^4(0)$ (right).

tive equational theorem-proving, learning of series-construction laws and user support for learning advanced screen-editor commands. Section 6 draws some conclusions.

2. Definitions

We assume familiarity with the classical definitions of terms, substitutions [13], and Horn clauses [24]. The cardinality of a finite set S is denoted by $\#S$. A signature Σ is a set of function symbols f , each of which has a fixed arity; if some f is nullary, we call it a constant. Let \mathcal{V} be an infinite set of variables. $\mathcal{T}_{\mathcal{V}}$ denotes the set of all terms over Σ and a given $V \subseteq \mathcal{V}$. For a term t , $\text{var}(t)$ denotes the set of variables occurring in t ; if it is empty, we call t a ground term. We call a term linear if each variable occurs at most once in it.

By $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, or $\{x_i \mapsto t_i \mid 1 \leq i \leq n\}$, we denote a substitution that maps each variable x_i to the term t_i . We call it ground if all t_i are ground. We use the postfix notation $t\sigma$ for application of σ to t , and $\sigma\tau$ for the composition of σ (to be applied first) and τ (second). The domain of σ is denoted by $\text{dom } \sigma$. A term t is called an instance of a term t' if $t = t'\sigma$ for some substitution σ . In this case, we call t more special than t' , and t' more general than t . We call t a renaming of t' if $\sigma : \mathcal{V} \rightarrow \mathcal{V}$ is a bijection.

A term t is called a syntactical generalization of terms t_1 and t_2 , if there exist substitutions σ_1 and σ_2 such that $t\sigma_1 = t_1$ and $t\sigma_2 = t_2$. In this case, t is called the most specific syntactical generalization of t_1 and t_2 , if for each syntactical generalization t' of t_1 and t_2 there exists a substitution σ' such that $t = t'\sigma'$. The most specific syntactical generalization of two terms is unique up to renaming; we also call it their syntactical anti-unifier [30].

An equational theory E is a finite set of equations between terms. $(=_E)$ denotes the smallest congruence relation that contains all equations of E . Define $[t]_E = \{t' \in \mathcal{T}_{\mathcal{V}} \mid t' =_E t\}$ to be the congruence class of t in the algebra of ground terms. The congruence class of a term is usually infinite; for example, using the equational theory from Fig. 1, we have $[0]_E = \{0, 0 * s(0), 0 + 0 * 0, \dots\}$. Let $[t]_E^\sigma = \{t' \in \mathcal{T}_{\text{dom } \sigma} \mid t'\sigma =_E t\}$ denote the set of all terms congruent to t under σ .

A congruence relation $(=_1)$ is said to be a refinement of another congruence relation $(=_2)$, if $\forall t, t' \in \mathcal{T}_{\mathcal{V}}: t =_1 t' \Rightarrow t =_2 t'$. In Section 5.1, we need the definition $t_1 \equiv_E t_2$ if $t_1\sigma =_E t_2\sigma$ for all ground substitutions σ with $\text{var}(t_1) \cup \text{var}(t_2) \subseteq \text{dom } \sigma$; this is equivalent to the equality of t_1 and t_2 being inductively provable [13, Section 3.2].

We call an n -ary function symbol f a constructor if n functions π_1^f, \dots, π_n^f exist such that $\forall t, t_1, \dots, t_n: (\bigwedge_{i=1}^n \pi_i^f(t) =_E t_i) \leftrightarrow t =_E f(t_1, \dots, t_n)$. The π_i^f are called selectors associated to f . As usual, we assume additionally that $f(s_1, \dots, s_n) \neq_E g(t_1, \dots, t_m) \neq_E x$ for any two constructors $f \neq g$, any variable x and arbitrary terms s_i, t_j . On this assumption, some constants can also be called constructors. No selector can be a constructor. If f is a constructor, then $[f(t_1, \dots, t_n)]_E = f([t_1]_E, \dots, [t_n]_E)$.

A term t is called a constructor term if it is built from constructors and variables only. Let t and t' be constructor terms. If $t\sigma_1 =_E t\sigma_2$, then $\forall x \in \text{var}(t): x\sigma_1 =_E x\sigma_2$. If $t\sigma =_E t'$, then $t\sigma' = t'$ for some σ' such that $x\sigma'$ is a constructor term and $x\sigma' =_E x\sigma$ for each $x \in \mathcal{V}$.

A (nondeterministic) regular tree grammar [10,37] is a triple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R} \rangle$. Σ is a signature, \mathcal{N} is a finite set of nonterminal symbols and \mathcal{R} is a finite set of rules

of the form $N ::= f_1(N_{11}, \dots, N_{1n_1}) \mid \dots \mid f_m(N_{m1}, \dots, N_{mn_m})$ or, abbreviated, $N ::= \bigcup_{i=1}^m f_i(N_{i1}, \dots, N_{in_i})$. Each $f_i(N_{i1}, \dots, N_{in_i})$ is called an alternative of the rule. We assume that for each nonterminal $N \in \mathcal{N}$, there is exactly one defining rule in \mathcal{R} with N as its left-hand side.

Given a grammar \mathcal{G} and a nonterminal $N \in \mathcal{N}$, the language $\mathcal{L}_{\mathcal{G}}(N)$ produced by N is defined in the usual way as the set of all ground terms derivable from N as the start symbol. We omit the index \mathcal{G} if it is clear from the context. We denote the total number of alternatives in \mathcal{G} by $\#\mathcal{G}$.

In Section 4, we will use the following predicate logic definitions. To simplify notation, we sometimes assume all predicate symbols to be unary. An n -ary predicate p' can be simulated by a unary p using an n -ary tupling constructor symbol and defining $p(\langle t_1, \dots, t_n \rangle) \Leftrightarrow p'(t_1, \dots, t_n)$. An n -ary predicate p is called determinate wrt some background theory B if there is some k such that wlog each of the arguments $k+1, \dots, n$ has only one possible binding, given the bindings of the arguments $1, \dots, k$ [25, Section 5.6.1]. The background theory B may be used to define p , hence p 's determinacy depends on B . Similar to the above, we sometimes write $p(t_1, \dots, t_n)$ as a binary predicate $p(\langle t_1, \dots, t_k \rangle, \langle t_{k+1}, \dots, t_n \rangle)$ to reflect the two classes of arguments. For a binary determinate predicate p , the relation $\{\langle s, t \rangle \mid s, t \in \mathcal{T}_{\emptyset} \wedge B \models p(s, t)\}$ corresponds to a function g . We sometimes assume that g is defined by equations from a given E , i.e., that $B \models (p(s, t) \Leftrightarrow g(s) =_E t)$.

A literal has the form $p(t)$ or $\neg p(t)$, where p is a predicate symbol and t is a term. We consider a negation to be part of the predicate symbol. We say that the literals L_1 and L_2 fit if both have the same predicate symbol, including negation. We extend $(=_E)$ to literals by defining $p(t_1) =_E p(t_2)$ if $t_1 =_E t_2$. For example, $(\neg \text{divides}(\langle 1+1, 5 \rangle)) =_E (\neg \text{divides}(\langle 2, 5 \rangle))$ if $1+1 =_E 2$.

A clause is a finite set $C = \{L_1, \dots, L_n\}$ of literals, with the meaning $C \Leftrightarrow L_1 \vee \dots \vee L_n$. We consider only nonredundant clauses, i.e., clauses that do not contain congruent literals. For example, $\{p(x+0), p(x)\}$ is redundant if $x+0 =_E x$. We write $C_1 \subseteq_E C_2$ if $\forall L_1 \in C_1 \exists L_2 \in C_2: L_1 =_E L_2$; if C_2 is nonredundant, L_2 is uniquely determined by L_1 .

We say that C_1 E -subsumes C_2 if $C_1 \sigma \subseteq_E C_2$ for some σ . In this case, the conjunction of E and C_1 implies C_2 ; however, there are other cases in which $E \wedge C_1 \models C_2$ but C_1 does not E -subsume C_2 . For example, $\{\neg p(x), p(f(x))\}$ implies, but does not subsume, $\{\neg p(x), p(f(f(x)))\}$, even for an empty E .

A Horn clause is a clause $\{p_0(t_0), \neg p_1(t_1), \dots, \neg p_n(t_n)\}$ with exactly one positive literal. It is also written as $p_0(t_0) \leftarrow p_1(t_1) \wedge \dots \wedge p_n(t_n)$. We call $p_0(t_0)$ the head literal, and $p_i(t_i)$ a body literal for $i = 1, \dots, n$. Like [25, Section 2.1], we call the Horn clause constrained if $\text{var}(t_0) \supseteq \text{var}(t_1, \dots, t_n)$.

We call a Horn clause $p_0(s_0, t_0) \leftarrow \bigwedge_{i=1}^n p_i(s_i, x_i) \wedge \bigwedge_{i=1}^m q_i(t_i)$ semi-determinate wrt some background theory B if all p_i are determinate wrt B , all variables x_i are distinct and do not occur in s_0 , $\text{var}(s_i) \subseteq \text{var}(s_0) \cup \{x_1, \dots, x_{i-1}\}$, and $\text{var}(t_1, \dots, t_m) \subseteq \text{var}(s_0, x_1, \dots, x_n)$. Semi-determinacy for clauses is a slight extension of determinacy defined by [25, Section 5.6.1], as it additionally permits arbitrary predicates q_i . On the other hand, [25] permits $x_i = x_j$ for $i \neq j$; however, $p_i(s_i, x_i) \wedge p_j(s_j, x_i)$ can be equivalently transformed into $p_i(s_i, x_i) \wedge p_j(s_j, x_j) \wedge x_i =_E x_j$.

3. E -generalization

We treat the problem of E -generalization of ground terms by standard algorithms on regular tree grammars. Here, we also give a rational reconstruction of the original approach from [22], who provided monolithic specially tailored algorithms for E -anti-unification. We confine ourselves to E -generalization of two terms. All methods work similarly for the simultaneous E -generalization of n terms.

3.1. The core method

Definition 1 (E -generalization). For an equational theory E , a term t is called an E -generalization, or E -anti-unifier, of terms t_1 and t_2 if there exist substitutions σ_1 and σ_2 such that $t\sigma_1 =_E t_1$ and $t\sigma_2 =_E t_2$. In Fig. 1 (right), we had $t_1 = 0$, $t_2 = s^4(0)$, $t = x * x$, $\sigma_1 = \{x \mapsto 0\}$, and $\sigma_2 = \{x \mapsto s^2(0)\}$.

As in unification, a most special E -generalization of arbitrary terms does not normally exist. A set $G \subseteq \mathcal{T}_V$ is called a set of E -generalizations of t_1 and t_2 if each member is an E -generalization of t_1 and t_2 . Such a G is called complete if, for each E -generalization t of t_1 and t_2 , G contains an instance of t .

As a first step towards computing E -generalization sets, let us weaken Definition 1 by *fixing* the substitutions σ_1 and σ_2 . We will see below, in Sections 4 and 5, that the weakened definition has important applications in its own right.

Definition 2 (Constrained E -generalization). Given two terms t_1, t_2 , a variable set V , two substitutions σ_1, σ_2 with $\text{dom } \sigma_1 = \text{dom } \sigma_2 = V$ and an equational theory E , define the set of E -generalizations of t_1 and t_2 wrt σ_1 and σ_2 as $\{t \in \mathcal{T}_V \mid t\sigma_1 =_E t_1 \wedge t\sigma_2 =_E t_2\}$. This set equals $[t_1]_E^{\sigma_1} \cap [t_2]_E^{\sigma_2}$.

If we can represent the congruence class $[t_1]_E$ and $[t_2]_E$ as some regular tree language $\mathcal{L}_{\mathcal{G}_1}(N_1)$ and $\mathcal{L}_{\mathcal{G}_2}(N_2)$, respectively, we can immediately compute the set of constrained E -generalizations $[t_1]_E^{\sigma_1} \cap [t_2]_E^{\sigma_2}$: The set of regular tree languages is closed wrt union, intersection and complement, as well as under inverse tree homomorphisms, which cover

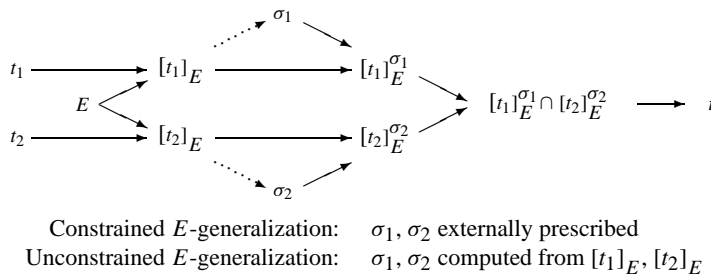


Fig. 2. E -generalization using tree grammars.

substitution application as a special case. Fig. 2 gives an overview of our method for computing the *constrained* set of E -generalizations of t_1 and t_2 wrt σ_1 and σ_2 according to Definition 2:

- From t_i and E , obtain a grammar for the congruence class $[t_i]_E$, if one exists; the discussion of this issue is postponed to Section 3.2 below.
- Apply the inverse substitution σ_i to the grammar for $[t_i]_E$ to get a grammar for $[t_i]_E^{\sigma_i}$, using some standard algorithm, e.g., that from [10, Theorem 7 in Section 1.4]. This algorithm takes time $\mathcal{O}(\#\mathcal{N} \cdot \text{size}(\sigma_i))$ for inverse substitution application, where $\text{size}(\sigma_i) = \sum_{x \in \text{dom } \sigma_i} \text{size}(x\sigma_i)$ is the total number of function symbols occurring in σ_i .
- Compute a grammar for the intersection $[t_1]_E^{\sigma_1} \cap [t_2]_E^{\sigma_2}$, using the product-automaton construction, e.g., from [10, Section 1.3], which takes time $\mathcal{O}(\#\mathcal{N}_1 \cdot \#\mathcal{N}_2)$.
- Each member t of the resulting tree language is an actual E -generalization of t_1 and t_2 . The question of enumerating that language is discussed later on.

Based on this result, we show how to compute the set of *unconstrained* E -generalizations of t_1 and t_2 according to Definition 1, where no σ_i is given. It is sufficient to compute two fixed *universal* substitutions τ_1 and τ_2 from the grammars for $[t_1]_E$ and $[t_2]_E$ and to let them play the role of σ_1 and σ_2 in the above method (cf. the dotted vectors in Fig. 2). Intuitively, we introduce one variable for each pair of congruence classes and map them to a kind of normalform member of the first and second class by τ_1 and τ_2 , respectively.

We give below a general construction that also accounts for *auxiliary* nonterminals not representing a congruence class, and state the universality of τ_1, τ_2 in a formal way. For the sake of technical simplicity, we assume that $[t_1]_E$ and $[t_2]_E$ share the same grammar \mathcal{G} ; this can easily be achieved by using the disjoint union of the grammars for $[t_1]_E$ and $[t_2]_E$.

Definition 3 (*Normal form*). Let an arbitrary tree grammar $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R} \rangle$ be given. A non-empty set of nonterminals $\mathbf{N} \subseteq \mathcal{N}$ is called *maximal* if $\bigcap_{N \in \mathbf{N}} \mathcal{L}(N) \neq \{\}$, but $\bigcap_{N \in \mathbf{N}'} \mathcal{L}(N) = \{\}$ for each $\mathbf{N}' \supsetneq \mathbf{N}$. Define $\mathcal{N}_{\max} = \{\mathbf{N} \subseteq \mathcal{N} \mid \mathbf{N} \neq \{\}, \mathbf{N} \text{ maximal}\}$. Choose some arbitrary but fixed

- maximal $\mathbf{N}(t) \supseteq \{N \in \mathcal{N} \mid t \in \mathcal{L}(N)\}$ for each $t \in \bigcup_{N \in \mathcal{N}} \mathcal{L}(N)$,
- maximal $\mathbf{N}(t)$ for each $t \in \mathcal{T}_{\mathcal{V}} \setminus \bigcup_{N \in \mathcal{N}} \mathcal{L}(N)$,
- ground term $\mathbf{t}(\mathbf{N}) \in \bigcap_{N \in \mathbf{N}} \mathcal{L}(N)$ for each $\mathbf{N} \in \mathcal{N}_{\max}$.

The mappings $\mathbf{N}(\cdot)$ and $\mathbf{t}(\cdot)$ can be effectively computed from \mathcal{G} . We abbreviate $\bar{t} = \mathbf{t}(\mathbf{N}(t))$; this is a kind of normalform of t . Each term not in any $\mathcal{L}(N)$, in particular each nonground term, is mapped to some arbitrary ground term, the choice of which does not matter. For a substitution $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, define $\bar{\sigma} = \{x_1 \mapsto \bar{t}_1, \dots, x_n \mapsto \bar{t}_n\}$. We always have $x\bar{\sigma} = \overline{x\sigma}$.

Lemma 4 (Substitution normalization). *For all $N \in \mathcal{N}$, $t \in \mathcal{T}_{\mathcal{V}}$, and σ ,*

- (1) $t \in \mathcal{L}(N) \Rightarrow \bar{t} \in \mathcal{L}(N)$, and
- (2) $t\sigma \in \mathcal{L}(N) \Rightarrow t\bar{\sigma} \in \mathcal{L}(N)$.

Proof. From the definition of $N(\cdot)$ and $t(\cdot)$, we get $t \in \mathcal{L}(N) \Rightarrow N \in N(t)$ and $N \in N \Rightarrow t(N) \in \mathcal{L}(N)$, respectively.

(1) Hence, $t \in \mathcal{L}(N) \Rightarrow N \in N(t) \Rightarrow \bar{t} \in \mathcal{L}(N)$.

(2) Induction on the structure of t :

- If $t = x \in \mathcal{V}$ and $x\sigma \in \mathcal{L}(N)$, then $x\bar{\sigma} = \overline{x\sigma} \in \mathcal{L}(N)$ by 1.
- Assuming $N ::= \dots f(N_{11}, \dots, N_{1n}) \mid \dots \mid f(N_{m1}, \dots, N_{mn}) \dots$, we have

$$\begin{aligned} f(t_1, \dots, t_n) \sigma &\in \mathcal{L}(N) \\ \Rightarrow \exists i \leq m \forall j \leq n: t_j \sigma &\in \mathcal{L}(N_{ij}) && \text{by Definition } \mathcal{L}(\cdot) \\ \Rightarrow \exists i \leq m \forall j \leq n: t_j \bar{\sigma} &\in \mathcal{L}(N_{ij}) && \text{by Induction Hypothesis} \\ \Rightarrow f(t_1, \dots, t_n) \bar{\sigma} &\in \mathcal{L}(N) && \text{by Definition } \mathcal{L}(\cdot) \quad \square \end{aligned}$$

Lemma 5 (Universal substitutions). *For each grammar \mathcal{G} , we can effectively compute two substitutions τ_1, τ_2 that are universal for \mathcal{G} in the following sense. For any two substitutions σ_1, σ_2 , a substitution σ exists such that for $i = 1, 2$, we have $\forall t \in \mathcal{T}_{\text{dom } \sigma_1 \cap \text{dom } \sigma_2} \forall N \in \mathcal{N}: t\sigma_i \in \mathcal{L}(N) \Rightarrow t\sigma\tau_i \in \mathcal{L}(N)$.*

Proof. Let $v(N_1, N_2)$ be a new distinct variable for each $N_1, N_2 \in \mathcal{N}_{\text{max}}$. Define $\tau_i = \{v(N_1, N_2) \mapsto t(N_i) \mid N_1, N_2 \in \mathcal{N}_{\text{max}}\}$ for $i = 1, 2$. Given σ_1 and σ_2 , let $\sigma = \{x \mapsto v(N(x\sigma_1), N(x\sigma_2)) \mid x \in \text{dom } \sigma_1 \cap \text{dom } \sigma_2\}$. Then $\sigma\tau_i$ and $\bar{\sigma}\bar{\tau}_i$ coincide on $\text{var}(t)$, and hence $t\sigma_i \in \mathcal{L}(N) \Rightarrow t\sigma\tau_i \in \mathcal{L}(N)$ by Lemma 4.2. \square

Example 6. We apply Lemma 5 to the grammar \mathcal{G} consisting of the topmost three rules in Fig. 3. The result will be used in Example 8 to compute some set of E -generalizations. We have $\mathcal{N}_{\text{max}} = \{\{N_0, N_t\}, \{N_1, N_t\}\}$, since, e.g., $0 \in \mathcal{L}(N_0) \cap \mathcal{L}(N_t)$ and $s(0) \in \mathcal{L}(N_1) \cap \mathcal{L}(N_t)$, while $\mathcal{L}(N_0) \cap \mathcal{L}(N_1) = \{\}$. We choose

$$N(t) = \begin{cases} \{N_0, N_t\} & \text{if } t \in \mathcal{L}(N_0), \\ \{N_1, N_t\} & \text{else,} \end{cases} \quad \text{and} \quad \begin{cases} t(\{N_0, N_t\}) = 0, \\ t(\{N_1, N_t\}) = s(0). \end{cases}$$

We abbreviate, e.g., $v(\{N_0, N_t\}, \{N_1, N_t\})$ to v_{01} . This way, we obtain

$$\begin{aligned} \tau_1 &= \{v_{00} \mapsto 0, v_{01} \mapsto 0, v_{10} \mapsto s(0), v_{11} \mapsto s(0)\} \quad \text{and} \\ \tau_2 &= \{v_{00} \mapsto 0, v_{01} \mapsto s(0), v_{10} \mapsto 0, v_{11} \mapsto s(0)\}. \end{aligned}$$

Given $t = x * y$, $\sigma_1 = \{x \mapsto 0 + 0, y \mapsto 0\}$ and $\sigma_2 = \{x \mapsto s(0), y \mapsto s(0) * s(0)\}$ for example, we obtain a proper instance $v_{01} * v_{01}$ of t using τ_1 and τ_2 :

$$\begin{aligned} \mathcal{L}(N_0) \ni (0+0) * 0 &\xleftarrow{\sigma_1} x * y \quad \xrightarrow{\sigma_2} s(0) * (s(0) * s(0)) \in \mathcal{L}(N_1) \\ \mathcal{L}(N_0) \ni 0 * 0 &\xleftarrow{\tau_1} v_{01} * v_{01} \quad \xrightarrow{\tau_2} s(0) * s(0) \in \mathcal{L}(N_1). \end{aligned}$$

The computation of universal substitutions is very expensive because it involves computing many tree-language intersections to determine the mappings $N(\cdot)$ and $t(\cdot)$. Assume $\mathcal{N} = \mathcal{N}_c \cup \mathcal{N}_o$, where \mathcal{N}_c comprises n_c nonterminals representing congruence classes and \mathcal{N}_o comprises n_o other ones. A maximal set N may contain at most one nonterminal

from \mathcal{N}_c and an arbitrary subset of \mathcal{N}_o ; however, no maximal N may be a proper subset of another one. By some combinatorics, we get $(n_c + 1) \cdot \binom{n_o}{n_o/2}$ as an upper bound on $\#\mathcal{N}_{\max}$. Hence, the cardinality of $\text{dom } \tau_i$ is bounded by the square of that number. In our experience, n_o is usually small. In most applications, it does not exceed 1, resulting in $\#\text{dom } \tau_i \leq (n_c + 1)^2$. Computing the τ_i requires $n_o + 1$ grammar intersections in the worst case, viz. when $\mathcal{N}_o \cup \{N_c\}$ for some $N_c \in \mathcal{N}_c$ is maximal. In this case, $\text{dom } \tau_i$ is rather small. Since the time for testing emptiness is dominated by the intersection computation time, and $(n_c + 1) \cdot \binom{n_o}{n_o/2} \leq (n_c + 1) \cdot n_o^{n_o/2} \leq \#\mathcal{G}^{n_o+1}$, we get a time upper bound of $\mathcal{O}(\#\mathcal{G}^{n_o+1})$ for computing the τ_i .

If the grammar is deterministic, then each nonterminal produces a distinct congruence class [26, Section 2], and we need compute no intersection at all to obtain τ_1 and τ_2 . We get $\#\text{dom } \tau_i = \#\mathcal{N}^2$. In this case, $N(\cdot)$, $t(\cdot)$, and $v(\cdot, \cdot)$ can be computed in linear time from \mathcal{G} . However, in general a nondeterministic grammar is smaller in size than its deterministic counterpart.

Theorem 7 (Unconstrained E -generalization). *Let an equational theory E and two ground terms t_1, t_2 be given. Let $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R} \rangle$ be a tree grammar and $N_1, N_2 \in \mathcal{N}$ such that $\mathcal{L}\mathcal{G}(N_i) = [t_i]_E$ for $i = 1, 2$. Let τ_1, τ_2 be as in Lemma 5. Then, $[t_1]_E^{\tau_1} \cap [t_2]_E^{\tau_2}$ is a complete set of E -generalizations of t_1 and t_2 . A regular tree grammar for it can be computed from \mathcal{G} in time $\mathcal{O}(\#\mathcal{G}^2 + \#\mathcal{G}^{n_o+1})$.*

Proof. If $t \in [t_1]_E^{\tau_1} \cap [t_2]_E^{\tau_2}$, then $t\tau_1 =_E t_1$ and $t\tau_2 =_E t_2$, i.e., t is an E -generalization of t_1 and t_2 . To show the completeness, let t be an arbitrary E -generalization of t_1 and t_2 , i.e., $t\sigma_i =_E t_i$ for some σ_i . Obtain σ from Lemma 5 such that $t\sigma\tau_i \in [t_i]_E^{\tau_i}$. Then $[t_1]_E^{\tau_1} \cap [t_2]_E^{\tau_2}$ contains the instance $t\sigma$ of t . \square

Since the set of E -generalizations resulting from our method is given by a regular tree grammar, it is necessary to enumerate some terms of the corresponding tree language in order to actually obtain some results. Usually, there is a notion of *simplicity* (or *weight*), depending on the application E -generalization is used in, and it is desirable to enumerate the simplest terms (with least weight) first. The minimal weight of a term in the language of each nonterminal can be computed in time $\mathcal{O}(\#\mathcal{G} \cdot \log \#\mathcal{G})$ by [6]. After that, it is easy to enumerate a nonterminal's language in order of increasing weight in time linear to the output size using a simple PROLOG program.

Example 8. To give a simple example, we generalize 0 and $s(0)$ wrt the equational theory from Fig. 1. Fig. 3 shows all grammars that will appear during the computation. For now, assume that the grammar \mathcal{G} defining the congruence classes $[0]_E$ and $[s(0)]_E$ is already given by the topmost three rules in Fig. 3. In Example 10 below, we discuss in detail how it can be obtained from E . Nevertheless, the rules of \mathcal{G} are intuitively understandable even now; e.g., the rule for N_0 in the topmost line reads: *A term of value 0 can be built by the constant 0, the sum of two terms of value 0, the product of a term of value 0 and any other term, or vice versa.* Similarly, $\mathcal{L}(N_1) = [s(0)]_E$ and $\mathcal{L}(N_t) = \mathcal{T}_{\{t\}}$. In Example 6, we already computed the universal substitutions τ_1 and τ_2 from \mathcal{G} .

$N_0 ::=$	0	$N_0 + N_0$	$N_0 * N_t$	$N_t * N_0$		
$N_1 ::=$	$s(N_0)$	$N_0 + N_1$	$N_1 + N_0$	$N_1 * N_1$		
$N_t ::=$	0 $s(N_t)$	$N_t + N_t$		$N_t * N_t$		
<hr/>						
$N_{0*} ::= v_{00} v_{01}$	0	$N_{0*} + N_{0*}$	$N_{0*} * N_{t*}$	$N_{t*} * N_{0*}$		
$N_{1*} ::=$	$v_{10} v_{11}$	$s(N_{0*}) N_{0*} + N_{1*} N_{1*} + N_{0*}$		$N_{1*} * N_{1*}$		
$N_{t*} ::= v_{00} v_{01} v_{10} v_{11} 0$	$s(N_{t*})$	$N_{t*} + N_{t*}$		$N_{t*} * N_{t*}$		
<hr/>						
$N_{*0} ::= v_{00}$	v_{10}	0	$N_{*0} + N_{*0}$	$N_{*0} * N_{*t}$	$N_{*t} * N_{*0}$	
$N_{*1} ::=$	v_{01}	v_{11}	$s(N_{*0}) N_{*0} + N_{*1} N_{*1} + N_{*0}$		$N_{*1} * N_{*1}$	
$N_{*t} ::= v_{00} v_{01} v_{10} v_{11} 0$	$s(N_{*t})$	$N_{*t} + N_{*t}$		$N_{*t} * N_{*t}$		
<hr/>						
$N_{00} ::= v_{00}$	0	$N_{00} + N_{00}$	$N_{00} * N_{tt}$	$N_{0t} * N_{t0}$	$N_{t0} * N_{0t}$	$N_{tt} * N_{00}$
$N_{01} ::=$	v_{01}		$N_{00} + N_{01}$	$N_{01} + N_{00}$	$N_{01} * N_{t1}$	$N_{t1} * N_{01}$
$N_{0t} ::= v_{00} v_{01}$	0	$N_{0t} + N_{0t}$		$N_{0t} * N_{tt}$	$N_{tt} * N_{0t}$	
$N_{t0} ::= v_{00}$	v_{10}	0	$N_{t0} + N_{t0}$	$N_{t0} * N_{tt}$	$N_{tt} * N_{t0}$	
$N_{t1} ::=$	v_{01}	v_{11}	$s(N_{t0})$	$N_{t0} + N_{t1}$	$N_{t1} + N_{t0}$	$N_{t1} * N_{t1}$
$N_{tt} ::= v_{00} v_{01} v_{10} v_{11} 0$	$s(N_{tt})$	$N_{tt} + N_{tt}$		$N_{tt} * N_{tt}$		

Fig. 3. Grammars \mathcal{G} (top), \mathcal{G}^{τ_1} , \mathcal{G}^{τ_2} and \mathcal{G}_{12} (bottom) in Examples 8 and 10.

Fig. 3 shows the grammars \mathcal{G}^{τ_1} and \mathcal{G}^{τ_2} resulting from inverse substitution application, defining the nonterminals N_{0*} , N_{1*} , N_{t*} and N_{*0} , N_{*1} , N_{*t} , respectively. For example, $\mathcal{L}_{\mathcal{G}^{\tau_1}}(N_{0*}) = [0]_E^{\tau_1}$, where the rule for N_{0*} is obtained from that for N_0 by simply including all variables that are mapped to a member of $\mathcal{L}_{\mathcal{G}}(N_0)$ by τ_1 . They appear as new constants, i.e., $\Sigma_{\mathcal{G}^{\tau_1}} = \Sigma_{\mathcal{G}^{\tau_2}} = \Sigma_{\mathcal{G}} \cup \{v_{00}, \dots, v_{11}\}$. For each $t \in \mathcal{L}_{\mathcal{G}^{\tau_1}}(N_{0*})$, we have $t\tau_1 =_E 0$.

The bottommost 6 rules in Fig. 3 show the intersection grammar \mathcal{G}_{12} obtained from a kind of product-automaton construction and defining N_{00}, \dots, N_{tt} . We have $\mathcal{L}_{\mathcal{G}_{12}}(N_{ij}) = \mathcal{L}_{\mathcal{G}^{\tau_1}}(N_i) \cap \mathcal{L}_{\mathcal{G}^{\tau_2}}(N_j)$ for $i, j \in \{0, 1, t\}$. By Theorem 7, $\mathcal{L}_{\mathcal{G}_{12}}(N_{01})$ is a complete set of E -generalizations of 0 and $s(0)$ wrt E . We have, e.g., $N_{01} \rightarrow N_{01} * N_{t1} \rightarrow v_{01} * v_{01}$, showing that 0 and $s(0)$ are both quadratic numbers, and $(v_{01} * v_{01})\tau_1 = 0 * 0 =_E 0$, $(v_{01} * v_{01})\tau_2 = s(0) * s(0) =_E s(0)$. By repeated application of the rules $N_{01} ::= \dots N_{01} * N_{t1} \dots$ and $N_{t1} ::= \dots N_{t1} * N_{t1} \dots$, we can obtain any generalization of the form $v_{01} * \dots * v_{01}$, with arbitrary paranthesation. By way of a less intuitive example, we have $v_{01} * s(v_{10} + v_{10}) \in \mathcal{L}_{\mathcal{G}_{12}}(N_{01})$.

3.2. Setting up grammars for congruence classes

The question of how to obtain a grammar representation of the initial congruence classes $[t_1]_E$ and $[t_2]_E$ was deferred in Section 3.1. It is discussed below.

Procedures that help to compute a grammar representing the congruence class of a ground term are given, e.g., in [26, Section 3]. This paper also provides a criterion for an equational theory inducing regular tree languages as congruence classes:

Theorem 9 (Ground equations [26]). *An equational theory induces regular congruence classes iff it is the deductive closure of finitely many ground equations E .*

Proof. To prove the *if* direction, start with a grammar consisting of a rule $N_{f(t_1, \dots, t_n)} ::= f(N_{t_1}, \dots, N_{t_n})$ for each subterm $f(t_1, \dots, t_n)$ occurring in E . For each equation

$(t_1 =_E t_2) \in E$, fuse the nonterminals N_{t_1} and N_{t_2} everywhere in the grammar. Then successively fuse every pair of nonterminals whose rules have the same right-hand sides. The result is a deterministic grammar \mathcal{G} such that $t_1 =_E t_2$ iff $t_1, t_2 \in \mathcal{L}_{\mathcal{G}}(N)$ for some N . In addition to this nonoptimal but intuitive algorithm, McAllester gives an $\mathcal{O}(n \cdot \log n)$ algorithm based on congruence closure [15], where n is the written length of E . The proof of the *only if* direction need not be sketched here. \square

In order to compute a complete set of E -generalizations of two given ground terms t_1 and t_2 , we do not need *all* congruence classes to be regular; it is sufficient that those of t_1 and t_2 are. More precisely, it is sufficient that $(=_E)$ is a refinement of some congruence relation $(=_G)$ with finitely many classes only, such that $[t_i]_E = [t_i]_G$ for $i = 1, 2$.

Example 10. Let $\Sigma = \{0, s, (+), (*)\}$, and let E be the equational theory from Fig. 1. To illustrate the application of Theorem 9, we show how to obtain a grammar defining $[0]_E, \dots, [s^n(0)]_E$ for arbitrary $n \in \mathbb{N}$. Obviously, $(=_E)$ itself has infinitely many congruence classes. However, in order to consider the terms $0, \dots, s^n(0)$ only, the relation $(=_G)$, defined by the classes $[0]_E, \dots, [s^n(0)]_E$ and $C = \bigcup_{i>n} [s^i(0)]_E$, is sufficient. This relation is, in fact, a congruence wrt $0, s, (+)$, and $(*)$: in a term $t \in \bigcup_{i=0}^n [s^i(0)]_E$, a member t_c of C can occur only in some subterm of the form $0 * t_c$ or similar, which always equals 0 , regardless of the choice of t_c .

For $n = 1$, we may choose a representative term $0, s(0)$, and c for the classes $[0]_E, [s(0)]_E$, and C , respectively. We may instantiate each equation from Fig. 1 with all possible combinations of representative terms, resulting in $3 + 3^2 + 3 + 3^2$ ground equations. After adding the equation $c = s(s(0))$, we may apply Theorem 9 to obtain a deterministic grammar \mathcal{G}_d with $\mathcal{L}_{\mathcal{G}_d}(N_0) = [0]_E$, $\mathcal{L}_{\mathcal{G}_d}(N_{s(0)}) = [s(0)]_E$, and $\mathcal{L}_{\mathcal{G}_d}(N_c) = C$. The equations' ground instances, and thus \mathcal{G}_d , can be built automatically. The grammar \mathcal{G}_d is equivalent to \mathcal{G} from Fig. 3 (top), which we used in Example 8. The latter is nondeterministic for the sake of brevity. It describes $[0]_E$ and $[s(0)]_E$ by N_0 and N_1 , respectively, while $\mathcal{L}(N_t) = [0]_E \cup [s(0)]_E \cup C$.

In [2, Corollary 16], another sufficient criterion is given. Intuitively, it allows us to construct a grammar from each equational theory that describes only operators *building up* larger values (normal forms) from smaller ones:

Theorem 11 (Constructive operators). *Let E be given by a ground confluent and Noetherian term-rewriting system. For each term $t \in \mathcal{T}_{\Sigma}$, let $\text{nf}(t)$ denote its unique normal form; let NF be the set of all normal forms. Let $(<)$ be a well-founded partial ordering on NF with a finite branching degree, and let (\preceq) be its reflexive closure. If $t_i \preceq \text{nf}(f(t_1, \dots, t_n))$ for all $f \in \Sigma$, $t_1, \dots, t_n \in NF$ and $i = 1, \dots, n$, then for each $t \in \mathcal{T}_{\Sigma}$, the congruence class $[t]_E$ is a regular tree language.*

Proof. Define one nonterminal N_t for each normal-form term $t \in NF$. Whenever $t =_E f(t_1, \dots, t_n)$ for some $t_1, \dots, t_n \in NF$, include an alternative $f(N_{t_1}, \dots, N_{t_n})$ into the right-hand side of the rule for N_t . Since $t_i \preceq t$ for all i , there are only finitely many such alternatives. This results in a finite grammar \mathcal{G} , and we have $\mathcal{L}_{\mathcal{G}}(N_{\text{nf}(t)}) = [t]_E$ for all

terms t . Let a_i denote the number of i -ary function symbols in Σ and m denote the maximal arity in Σ . Then we need $\sum_{i=0}^m a_i \cdot \#NF^i$ normal-form computations to build \mathcal{G} , which has a total of these many alternatives and $\#NF$ rules. Its computation takes $\mathcal{O}(\#\mathcal{G})$ time. \square

By way of an example, consider the theory E consisting of only equations 1. and 2. in Fig. 1. E is known to be ground-confluent and Noetherian and to lead to $NF = \{s^n(0) \mid n \in \mathbb{N}\}$. Defining $s^i(0) < s^j(0) \Leftrightarrow i < j$, we observe that $s^i(0) \leq s^{i+j}(0) = nf(s^i(0) + s^j(0))$; similarly, $s^j(0) \leq nf(s^i(0) + s^j(0))$ and $s^i(0) \leq nf(s(s^i(0)))$. Hence, E leads to regular congruence classes by Theorem 11. For example, there are just five ways to obtain a term of value $s^3(0)$ from normal-form terms using s and $(+)$. Accordingly, $N_3 ::= s(N_2) \mid N_3 + N_0 \mid N_2 + N_1 \mid N_1 + N_2 \mid N_0 + N_3$ defines $[s^3(0)]_E$.

If their respective preconditions are met, Theorems 9 and 11 allow us to automatically compute a grammar describing the congruence classes wrt a given theory E . In practice, with Theorem 9 we have the problem that E is rarely given by ground equations only, while Theorem 11 requires properties that are sufficient but not necessary. For example, not even for the whole theory from Fig. 1 is there an ordering such that $s^i(0) \leq 0 = nf(s^i(0) * 0)$ and $0 \leq s^i(0) = nf(0 + s^i(0))$.

So far, it seems best to set up a *grammar scheme* for a given E manually. For example, for E from Fig. 1, it is easy to write a program that reads $n \in \mathbb{N}$ and computes a grammar describing the congruence classes $[0]_E, \dots, [s^n(0)]_E$: The grammar consists of the rules for N_0 and N_i from Fig. 3 (top) and one rule $N_i ::= s(N_{i-1}) \mid \bigvee_{j=0}^i N_j + N_{i-j} \mid \bigvee_{j,k=i} N_j * N_k$ for each $i = 1, \dots, n$. Similarly, grammar schemes for the theory of list operators like *append*, *reverse*, etc. and other theories can be implemented.

The lack of a single algorithm that computes a grammar for each E from a sufficiently large class of equational theories restricts the applicability of our E -generalization method to problems where E is not changed too often. Using grammar schemes, this restriction can be relaxed somewhat. The grammar-scheme approach is also applicable to theories given by conditional equations or even other formulas, as long as they lead to regular congruence classes and the schemes are computable.

A further problem is that not all equational theories lead to regular congruence classes. Consider, for example, subtraction $(-)$ on natural numbers. We have $0 =_E s^i(0) - s^i(0)$ for all $i \in \mathbb{N}$. Assume that $(=_E)$ is a refinement of some $(=_G)$ with finitely many classes; then $s^i(0) =_G s^j(0)$ for some $i \neq j$. If $(=_G)$ is a congruence relation, $0 =_G s^i(0) - s^i(0) =_G s^i(0) - s^j(0)$, although $0 \neq_E s^i(0) - s^j(0)$. Hence, $[0]_E = [0]_G$ is impossible. Thus, if an operator like $(-)$ is defined in E , we cannot E -generalize using our method.

However, we can still compute an *approximation* of the E -generalization set in such cases by artificially cutting off grammar rules after a maximum number of alternatives. This will result in a set that still contains only correct E -generalizations but that is usually incomplete. For example, starting from the grammar rules

$$\begin{aligned} N_0 &::= 0 & \mid N_0 - N_0 \mid N_1 - N_1 \mid N_2 - N_2 \\ N_1 &::= s(N_0) \mid N_1 - N_0 \mid N_2 - N_1 \\ N_2 &::= s(N_1) \mid N_2 - N_0, \end{aligned}$$

we obtain only E -generalization terms t whose evaluation never exceeds the value 2 on any subterm's instance. Depending on the choice of the *cut-off point*, the resulting E -generalization set may suffice for a given application.

To overcome the limited expressiveness of pure regular tree grammars, it would be desirable to extend the results of Section 3.1 to automata with equality tests. However, for automata with equality tests between siblings [4,10] or, equivalently, shallow systems of sort constraints [38], we have the problem that this language class is not closed under inverse substitution application [4, Section 5.2]. For reduction automata [9,10] and generalized reduction automata [8], it seems to be still unknown whether they are closed under inverse substitution application. Moreover, the approach using universal substitutions τ_1, τ_2 strongly depends on the fact that $\bigwedge_{j=1}^n t_j \in \mathcal{L}(N_{ij}) \Rightarrow f(t_1, \dots, t_n) \in \mathcal{L}(N)$, which is needed in the proof of Lemma 4(2). In other words, the rule $N ::= \dots f(N_{i1}, \dots, N_{in})$ is *not* allowed to have additional constraints. For these reasons, our approach remains limited to ordinary regular tree grammars, i.e., those without any equality constraints. The following lemma shows that we cannot find a more sophisticated grammar formalism that can handle *all* equational theories, anyway.

Lemma 12 (General uncomputability of E -generalization). *There are equational theories E such that it is undecidable whether the set of constrained E -generalizations for certain $t_1, t_2, \sigma_1, \sigma_2$ is empty. Such a set cannot be represented by any grammar formalism that is closed wrt language intersection and allows testing for emptiness.*

Proof. We encode a version of Post's *correspondence problem* into an E -generalization problem for a certain E . Let a set $\{\langle a_1, b_1 \rangle, \dots, \langle a_n, b_n \rangle\}$ of pairs of nonempty strings over a finite alphabet A be given. It is known to be undecidable whether there exists a sequence i_1, i_2, \dots, i_m such that $m \geq 1$ and $a_1 \cdot a_{i_2} \dots a_{i_m} = b_1 \cdot b_{i_2} \dots b_{i_m}$ [35, Section 2.7], where “ \cdot ” denotes string concatenation. Let $\Sigma = A \cup \{1, \dots, n\} \cup \{(\cdot), f, a, b\}$, and let E consist of the following equations:

$$\begin{aligned} (x \cdot y) \cdot z &= x \cdot (y \cdot z), \\ f(a, x \cdot i, y \cdot a_i) &= f(a, x, y) \quad \text{for } i = 1, \dots, n, \\ f(b, x \cdot i, y \cdot b_i) &= f(b, x, y) \quad \text{for } i = 1, \dots, n, \end{aligned}$$

where x, y , and z are variables. Then, the congruence class $[f(a, 1, a_1)]_E$ and $[f(b, 1, b_1)]_E$ equals the set of all admitted Post sequences of a_i and b_i , respectively. Let $\sigma_1 = \{x \mapsto a\}$ and $\sigma_2 = \{x \mapsto b\}$, then the set of constrained E -generalizations $[f(a, 1, a_1)]_E^{\sigma_1} \cap [f(b, 1, b_1)]_E^{\sigma_2}$ is nonempty iff the given correspondence problem has a solution. \square

4. Learning predicate definitions

In this section, we relate E -generalization to Inductive Logic Programming (ILP), which seems to be the closest approach to machine learning. We argue in favor of an *outsourcing* of equational tasks from Horn program induction algorithms, similar to what has long been common practice in the area of deduction. From a theoretical point of

Necessity:	$B \not\models F^+$
Sufficiency:	$B \wedge h \models F^+$
Weak Consistency:	$B \wedge h \not\models \text{false}$
Strong Consistency:	$B \wedge h \wedge F^- \not\models \text{false}$

Fig. 4. Requirements for hypothesis generation according to Muggleton [28].

view, a general-purpose theorem-proving algorithm is complete wrt equational formulas, too, if all necessary instances of the *congruence axioms* $s =_E t \Rightarrow f(s) =_E f(t)$ and $s =_E t \wedge p(s) \Rightarrow p(t)$ are supplied. However, in practice it proved to be much more efficient to handle the equality predicate ($=_E$) separately using specially tailored methods, like E -unification for fixed, or paramodulation for varying E .

Similarly, we show that integrating E -anti-unification into an ILP algorithm helps to restrict the hypotheses-language bias and thus the search space. In particular, learning of determinate clauses can be reduced to learning of atoms using generalization wrt an E defining a function for each determinate predicate.

We investigate the *learning* of a definition of a new predicate symbol p in four different settings. In all cases, we are given a conjunction $F \Leftrightarrow F^+ \wedge F^-$ of positive and negative ground facts, and a *background theory* B describing an equational theory E . In this section, we always assume that E leads to regular congruence classes. We generate a *hypothesis* h that must *explain* F using B . From Inductive Logic Programming, up to four requirements for such a hypothesis are known [28, Section 2.1]. They are listed in Fig. 4. More precisely, the *Necessity* requirement does not impose a restriction on h , but forbids any generation of a (positive) hypothesis, provided the positive facts are explainable without it. Muggleton remarks that this requirement can be checked by a conventional theorem prover before calling hypothesis generation. The *Weak* and *Strong Consistency* requirements coincide if there are no negative facts; otherwise, the former is entailed by the latter. In [16, Section 5.2.4], only *Sufficiency* (called *Completeness* there) and *Strong Consistency* are required.

We show under which circumstances E -generalization can generate a hypothesis satisfying the *Sufficiency* and both *Consistency* requirements. The latter are meaningful in an equational setting only if we require that ($=_E$) is *nontrivial*, i.e., $\exists x, y: \neg x =_E y$. Without this formula, *false* could not even be derived from an equational theory, however nonsensical.

Given E , we require our logical background theory B to entail the reflexivity, symmetry and transitivity axiom for ($=_E$), a congruence axiom for ($=_E$) wrt each function $f \in \Sigma$ and each predicate p occurring in B , the universal closure of each equation in E , and the nontriviality axiom for ($=_E$). As a more stringent alternative to the *Necessity* requirement, we assume that B is not contradictory and that the predicate symbol p for which a definition has to be learned does not occur in B , except in p 's congruence axiom.

To begin with, we investigate the learning of a definition of a unary predicate p by an atom $h \Leftrightarrow p(t)$. Let $F^+ \Leftrightarrow \bigwedge_{i=1}^n p(t_i)$ and $F^- \Leftrightarrow \bigwedge_{i=n+1}^m \neg p(t_i)$. For sets T^+, T^- of ground terms and an arbitrary term t , define

$$\begin{aligned} h^+(t, T^+) &\Leftrightarrow \forall t' \in T^+ \exists \chi: t \chi =_E t' \quad \text{and} \\ h^-(t, T^-) &\Leftrightarrow \forall t' \in T^- \forall \chi: t \chi \neq_E t'. \end{aligned}$$

We name the substitutions χ instead of σ in order to identify them as given by h^+ and h^- in the proofs below.

Lemma 13 (Requirements). *Let t_i, t'_i be ground terms for $i = 1, \dots, m$ and let t be an arbitrary term. Let $F^+ \Leftrightarrow \bigwedge_{i=1}^n p(t_i)$ and $F^- \Leftrightarrow \bigwedge_{i=n+1}^m \neg p(t_i)$. Let $T^+ = \{t_1, \dots, t_n\}$ and $T^- = \{t_{n+1}, \dots, t_m\}$. Then:*

- (1) $B \wedge p(t) \models p(t'_1) \vee \dots \vee p(t'_{n'})$ iff $t\sigma =_E t'_i$ for some i, σ .
- (2) $B \wedge p(t) \models F^+$ iff $h^+(t, T^+)$.
- (3) $B \wedge p(t) \wedge F^- \not\models \text{false}$ iff $h^-(t, T^-)$.

Proof.

- (1) The *if* direction is trivial. To prove the *only if* direction, observe that B has a Herbrand model containing no instances of p . If we add the set $\{p(t'') \mid \exists \sigma \text{ ground: } t\sigma =_E t''\}$ to that model, we get a Herbrand model of $B \wedge p(t)$. In this model, $p(t'_1) \vee \dots \vee p(t'_{n'})$ holds only if some $p(t'_i)$ holds, since they are all ground. This implies in turn that $p(t'_i)$ is among the $p(t'')$, i.e., $t\sigma =_E t'_i$ for some ground substitution σ .
- (2) Follows from (1) for $n' = 1$.
- (3) Follows from (1) for $\neg F^- \Leftrightarrow p(t'_1) \vee \dots \vee p(t'_{n'})$, paraphrasing Strong Consistency as $B \wedge p(t) \not\models \neg F^-$. \square

The following Lemma 14, and Lemma 20 below for the determinate case, are the workhorses of this section. They show how to apply E -generalization to obtain a hypotheses term set from given positive and negative example term sets. In the theorems based on these lemmas, we only need to enclose the hypotheses terms as arguments to appropriate predicates.

Lemma 14 (Hypotheses). *For each finite set $T^+ \cup T^-$ of ground terms, we can compute a regular set $H = H_{14}(T^+, T^-)$ such that for all $t \in \mathcal{T}_\gamma$:*

$$t \in H \Rightarrow h^+(t, T^+) \wedge h^-(t, T^-), \quad \text{and} \\ \exists \sigma: t\sigma \in H \Leftarrow h^+(t, T^+) \wedge h^-(t, T^-).$$

Proof. Let $T^+ = \{t_1, \dots, t_n\}$, let \mathcal{G} be a grammar defining $[t_1]_E, \dots, [t_n]_E$. Obtain the universal substitutions τ_1, \dots, τ_n for \mathcal{G} from Lemma 5. All τ_i have the same domain. Using the notations from Definition 3, let

$$S = \{\sigma \mid \text{dom } \sigma = \text{dom } \tau_1 \wedge \forall x \in \text{dom } \sigma \exists N \in \mathcal{N}_{\max}: x\sigma = t(N)\}.$$

The set S is finite, but large; it has $\#\mathcal{N}_{\max}^{\mathcal{N}_{\max}^n}$ elements. Define $H^+ = \bigcap_{i=1}^n [t_i]_E^{\tau_i}$ and $H^- = \bigcup_{t' \in T^-} \bigcup_{\sigma \in S} [t']_E^\sigma$. Define $H = H^+ \setminus H^-$; all these sets are regular tree languages and can be computed using standard grammar algorithms.

- For $t \in H$, we trivially have $h^+(t, T^+)$. Assume that $h^-(t, T^-)$ does not hold, i.e., $t\sigma =_E t'$ for some σ and $t' \in T^-$. Since $\text{var}(t) \subseteq \text{dom } \tau_1$ by construction, we may

assume wlog $\text{dom } \sigma = \text{dom } \tau_1$; hence $\bar{\sigma} \in S$. From Lemma 4(2), we get $t\sigma \in [t']_E \Rightarrow t\bar{\sigma} \in [t']_E \Rightarrow t \in [t']_E^{\bar{\sigma}}$, contradicting $t \notin H^-$.

- If $h^+(t, T^+) \wedge h^-(t, T^-)$, then $t\chi_i =_E t_i$ for some $\chi_i, i = 1, \dots, n$. Using Lemma 5, we get some σ such that $t\sigma \in [t_i]_E^{T_i}$, hence $t\sigma \in H^+$. If we had $t\sigma \in [t']_E^{\sigma'}$ for some $t' \in T^-$ and $\sigma' \in S$, then $t\sigma\sigma' =_E t'$, contradicting $h^-(t, T^-)$. \square

Theorem 15 (Atomic definitions). *Let t_1, \dots, t_m be ground terms. Let $F^+ \Leftrightarrow \bigwedge_{i=1}^n p(t_i)$ and $F^- \Leftrightarrow \bigwedge_{i=n+1}^m \neg p(t_i)$ be given. We can compute a regular set $H = H_{15}(F^+, F^-)$ such that*

- each $p(t) \in H$ is a hypothesis satisfying the Sufficiency and the Strong Consistency requirement wrt F^+, F^- ; and
- for each hypothesis satisfying these requirements and having the form $p(t)$, we have $p(t\sigma) \in H$ for some σ .

Proof. Define $T^+ = \{t_i \mid i = 1, \dots, n\}$ and $T^- = \{t_i \mid i = n+1, \dots, m\}$. By Lemma 13(2) and (3), $p(t)$ is a hypothesis satisfying the Sufficiency and the Strong Consistency requirement iff $h^+(t, T^+)$ and $h^-(t, T^-)$, respectively. By Lemma 14, we may thus choose $H = \{p(t) \mid t \in H_{14}(T^+, T^-)\}$, which is again a regular tree language. \square

The time requirement of the computation from Theorem 15 grows very quickly if negative examples are given. Even for deterministic grammars, up to $(m - n) \cdot \#\mathcal{N}^{\#\mathcal{N}^n}$ inverse substitution applications are needed, each requiring a renamed copy of the original grammar. If only positive examples are given, the time complexity is $\mathcal{O}(\#\mathcal{G}^n + \#\mathcal{G}^{n_o+1})$, which allows nontrivial practical applications.

By way of an example, consider again the equational theory E from Fig. 1. Let $F^+ \Leftrightarrow 0 \leq 0 \wedge 0 \leq s(0)$ and $F^- \Leftrightarrow \text{true}$ be given. In Example 8, we already computed a grammar \mathcal{G} describing $[0]_E = \mathcal{L}_{\mathcal{G}}(N_0)$ and $[s(0)]_E = \mathcal{L}_{\mathcal{G}}(N_1)$, see Fig. 3 (top). The congruence class of, say, $\langle 0, 0 \rangle$ could be defined by the additional grammar rule $N_{\langle 0, 0 \rangle} ::= \langle N_0, N_0 \rangle$. Instead of that rule, we add $N_{0 \leq 0} ::= (N_0 \leq N_0)$ to the grammar, anticipating that any $\langle t_1, t_2 \rangle \in H_{14}$ will be transformed to $\langle t_1 \leq t_2 \rangle \in H_{15}$ by Theorem 15, anyway. Similarly, we add the rule $N_{0 \leq 1} ::= (N_0 \leq N_1)$. The universal substitutions we obtain following the construction of Lemma 14 are simply τ_1 and τ_2 from Example 8. We do not extend them to also include variables like $v(\{N_{0 \leq 0}\}, \{N_{0 \leq 1}\}) = v_{0 \leq 0, 0 \leq 1}$ in their domain because neither $v_{0 \leq 0, 0 \leq 1} \in H_{15}$ nor $v_{\langle 0, 0 \rangle, \langle 0, 1 \rangle} \in H_{14}$ would make sense. From a formal point of view, retaining the τ_i from Example 8 restricts H_{15} and H_{14} to predicates and terms of the form $t_1 \leq t_2$ and $\langle t_1, t_2 \rangle$, respectively.

After lifting the extended \mathcal{G} wrt τ_1, τ_2 , we obtain the grammar \mathcal{G}_{12} from Fig. 3 (bottom), extended by some rules like $N_{0 \leq 0, 0 \leq 1} ::= (N_{00} \leq N_{01})$. By Theorem 15, each element of $H_{15}(F^+, F^-) = \mathcal{L}_{\mathcal{G}_{12}}(N_{0 \leq 0, 0 \leq 1})$ is a hypothesis satisfying the Sufficiency and the Strong Consistency requirement. Using the variable naming convention from Example 8, members of H_{15} are, e.g.:

- | | | |
|----------------------------------|---|----------------------------------|
| 1. $v_{00} \leq v_{01}$ | 3. $v_{00} * v_{00} \leq v_{01}$ | 5. $0 \leq v_{01}$ |
| 2. $v_{00} \leq v_{01} * v_{01}$ | 4. $v_{00} * v_{01} \leq v_{11} * v_{01}$ | 6. $v_{00} \leq v_{00} + v_{01}$ |

Hypothesis 1 intuitively means that (\leq) relates every possible pair of terms. Hypotheses 2 and 3 indicate that F^+ was chosen too specifically, viz. all examples had quadratic numbers as the right or left argument of (\leq) . Similarly, hypothesis 5 reflects the fact that all left arguments are actually zero. While $0 \leq x$ is a valid law, $x \leq y \Leftrightarrow x =_E 0$ is not a valid definition. Similarly, no variant of $x \leq x$ can be found in H_{15} because it does not cover the second example from F^+ . Hypothesis 6 is an acceptable definition of the (\leq) relation on natural numbers; it corresponds to $x \leq y \Leftrightarrow \exists z \in \mathbb{N} : y =_E x + z$. If we take F^+ as above, but $F^- \Leftrightarrow s(0) \not\leq 0$, we get S as the set of all 2^4 substitutions with domain $\{v_{00}, \dots, v_{11}\}$ and range $\{0, s(0)\}$. The resulting grammar for H_{15} is too large to be shown here. H_{15} still contains hypotheses 5 and 6 from above, but no longer 1 to 4.

We now demonstrate how E -generalization can be incorporated into an existing Inductive Logic Programming method to learn clauses. To be concrete, we chose the method of *relative least general generalization* (r -lgg), which originates from [30] and forms the basis of the GOLEM system [27]. We show how to extend it to deal with a given equational background theory E .

Theorem 16 (Clausal definitions). *Let two ground clauses C_1 and C_2 be given. We can compute a regular set $H = \text{lgg}_E(C_1, C_2)$ such that:*

- each $C \in H$ is a clause that E -subsumes C_1 and C_2 ; and
- each clause E -subsuming both C_1 and C_2 also subsumes an element of H .

Proof. Let $M = \{\langle L_1, L_2 \rangle \mid L_1 \in C_1 \wedge L_2 \in C_2 \wedge L_1 \text{ fits } L_2\}$. Assuming $M = \{\langle p_i(t_{1i}), p_i(t_{2i}) \rangle \mid i = 1, \dots, m\}$, let $T^+ = \{\langle t_{11}, \dots, t_{1m} \rangle, \langle t_{21}, \dots, t_{2m} \rangle\}$ and $T^- = \{\}$. Let $H = \{\langle p_i(t_i) \mid i = 1, \dots, m \rangle \mid \langle t_1, \dots, t_m \rangle \in H_{14}(T^+, T^-)\}$. H is again a regular tree language, because the regular $H_{14}(T^+, T^-)$ is the image of H under the tree homomorphism that maps $\{p_1(x_1), \dots, p_m(x_m)\}$ to $\langle x_1, \dots, x_m \rangle$, cf. [10, Theorem 7 in Section 1.4].

If $\{p_1(t_1), \dots, p_m(t_m)\} \in H$, i.e., $h^+(\langle t_1, \dots, t_m \rangle, T^+, T^-)$, then $\{p_1(t_1\chi_i), \dots, p_m(t_m\chi_i)\} \subseteq_E C_i$ for $i = 1, 2$, where χ_i are the substitutions from the definition of h^+ . Conversely, let some clause C E -subsume both C_1 and C_2 . We assume wlog $C = \{p_1(t_1), \dots, p_n(t_n)\}$ and $t_j\sigma_i =_E t_{ij}$ for some σ_i for $i = 1, 2$ and $j = 1, \dots, n$. By Lemma 5, some σ exists such that $t_j\sigma\tau_i =_E t_{ij}$. Choosing $t'_j = t_j\sigma$ for $j = 1, \dots, n$ and $t'_j = v(N(t_{1j}), N(t_{2j}))$ for $j = n+1, \dots, m$, we obtain $t'_j\tau_i =_E t_{ij}$ for $j = 1, \dots, m$. Hence, $h^+(\langle t'_1, \dots, t'_m \rangle, T^+, T^-)$ holds, i.e., $C\sigma \subseteq \{p_1(t'_1), \dots, p_m(t'_m)\} \in H$.

To compute H , the grammar defining all $[t_{ij}]_E$ must be extended to also define $[\langle t_{i1}, \dots, t_{im} \rangle]_E$. Since only nonterminals for congruence classes are added, no additional language intersections are necessary to compute the extended τ_i . \square

For an empty E , we have $\text{lgg}_E(C_1, C_2) = \{\text{lgg}(C_1, C_2)\}$, and Theorem 16 implies Plotkin's lgg theorem [30, Theorem 3] as a special case. In the terminology of Fig. 4, we have $F^+ \Leftrightarrow C_1 \wedge C_2$ and $F^- \Leftrightarrow \text{true}$. The Consistency requirement is satisfied if some predicate symbol p different from $(=_E)$ occurs in both C_1 and C_2 but not in B , except for p 's congruence axiom. In this case, each hypothesis h will have the form $p(\dots) \vee \dots$, hence $B \wedge h$ cannot be contradictory. The set $\text{lgg}_E(C_1, C_2)$ is a subset of, but is not equal to, the set of all hypothesis clauses satisfying Sufficiency. Usually, there are other clauses

that imply both C_1 and C_2 but do not E -subsume both. The same limitation applies to Plotkin's syntactical lgg .

Theorems 16 and 15 share a special case: $lgg_E(\{p(t_1)\}, \{p(t_2)\})$ from Theorem 16 equals $H_{15}(p(t_1) \wedge p(t_2), true)$ from Theorem 15. In this case, Theorem 15 is stronger because it ensures that the result set contains *all* sufficient hypotheses. On the other hand, Theorem 16 allows a more general form of both hypotheses and examples.

To illustrate Theorem 16, consider a well-known example about learning family relations. We use the abbreviations d —daughter, p —parent, f —female, e —eve, g —george, h —helen, m —mary, n —nancy, and t —tom. Let the background knowledge $K \Leftrightarrow p(h, m) \wedge p(h, t) \wedge p(g, m) \wedge p(t, e) \wedge p(n, e) \wedge f(h) \wedge f(m) \wedge f(n) \wedge f(e)$ and the positive examples $F_1 \Leftrightarrow d(m, h)$ and $F_2 \Leftrightarrow d(e, t)$ be given. By generalizing relative to K , i.e., by computing $lgg((F_1 \leftarrow K), (F_2 \leftarrow K))$, and by eliminating all body literals containing a variable not occurring in the head literal, the clausal definition of the *daughter* relation $d(v_{me}, v_{ht}) \leftarrow p(v_{ht}, v_{me}) \wedge f(v_{me}) \wedge K$ results.

In addition, using the abbreviation s —spouse, let the equations $E = \{s(g) = h, s(h) = g, s(n) = t, s(t) = n\}$ be given. The congruence classes of, e.g., g and h can be described by $N_g ::= g \mid s(N_h)$ and $N_h ::= h \mid s(N_g)$. We obtain by Theorem 16 all clauses of the form $d(v_{me}, t_{ht}) \leftarrow p(t'_{ht}, v_{me}) \wedge p(t_{gn}, v_{me}) \wedge f(v_{me})$ for any $t_{ht}, t'_{ht} \in [h]_E^{t_1} \cap [t]_E^{t_2}$ and $t_{gn} \in [g]_E^{t_1} \cap [n]_E^{t_2}$. In order to obtain a constrained clause as before, we first choose some t_{ht} for the head literal, and then choose t'_{ht} and t_{gn} from the filtered sets $[h]_E^{t_1} \cap [t]_E^{t_2} \cap \mathcal{T}_{\text{var}(t_{ht})}$ and $[g]_E^{t_1} \cap [n]_E^{t_2} \cap \mathcal{T}_{\text{var}(t_{ht})}$, respectively. We use the standard intersection algorithm for tree grammars mentioned in Section 3.1 for filtering, which in this case requires linear time in the grammar size for $[h]_E^{t_1} \cap [t]_E^{t_2}$ and $[g]_E^{t_1} \cap [n]_E^{t_2}$. Choosing the smallest solutions for t_{ht} , t'_{ht} , and t_{gn} , we obtain $d(v_{me}, v_{ht}) \leftarrow p(v_{ht}, v_{me}) \wedge p(s(v_{ht}), v_{me}) \wedge f(v_{me})$, which reflects the fact that our background knowledge did not describe any concubinages.

Below, we prove a learning theorem similar to Theorem 15, but that yields only those atomic hypotheses $p(s, t)$ that define a determinate predicate p . Formally, we are looking for those $p(s, t)$ that satisfy $\forall s'_1, s'_2, t'_1, t'_2: (B \wedge p(s, t) \wedge s'_1 =_E s'_2 \models p(s'_1, t'_1) \wedge p(s'_1, t'_2)) \Rightarrow (B \models t'_1 =_E t'_2)$. In such cases, we say that the hypothesis $p(s, t)$ is determinate. Determinacy of a hypothesis is essentially a *semantic* property [25, Section 5.6.1]; it is even undecidable for certain background theories. In order to compute the set of all determinate hypotheses, we have to make a little detour by defining a notion of *weak determinacy*, which is equivalent to a simple *syntactic* criterion (Lemma 18).

Since B does not imply anything about p except its congruence property, we assume $B \Leftrightarrow B'' \wedge (\forall x, y, x', y': x =_E x' \wedge y =_E y' \wedge p(x, y) \rightarrow p(x', y'))$, where p does not occur in B'' . We replace the full congruence axiom about p by a partial one: $B' \Leftrightarrow B'' \wedge (\forall x, y, y': y =_E y' \wedge p(x, y) \rightarrow p(x, y'))$. We call a hypothesis $p(s, t)$ weakly determinate if $\forall s', t'_1, t'_2: (B' \wedge p(s, t) \models p(s', t'_1) \wedge p(s', t'_2)) \Rightarrow (B' \models t'_1 =_E t'_2)$. For example, using E from Fig. 1, $p(x * y, x + y)$ is a weakly, but not ordinarily, determinate hypothesis. We define for sets T^+ , T^- of ground-term pairs and arbitrary terms s, t :

$$\begin{aligned} h^+(s, t, T^+) &\Leftrightarrow \forall \langle s', t' \rangle \in T^+ \exists \sigma: s\sigma = s' \wedge t\sigma =_E t' \quad \text{and} \\ h^-(s, t, T^-) &\Leftrightarrow \forall \langle s', t' \rangle \in T^- \forall \sigma: s\sigma \neq s' \vee t\sigma \neq_E t'. \end{aligned}$$

We have a lemma similar to Lemma 13, with a similar proof, which is omitted here.

Lemma 17 (Weak requirements). *Let s_i, t_i, s'_i, t'_i be ground terms and s, t arbitrary terms. Let $F^+ \Leftrightarrow \bigwedge_{i=1}^n p(s_i, t_i)$ and $F^- \Leftrightarrow \bigwedge_{i=n+1}^m \neg p(s_i, t_i)$. Let $T^+ = \{\langle s_1, t_1 \rangle, \dots, \langle s_n, t_n \rangle\}$ and $T^- = \{\langle s_{n+1}, t_{n+1} \rangle, \dots, \langle s_m, t_m \rangle\}$. Then:*

- (1) $B' \wedge p(s, t) \models p(s'_1, t'_1) \vee \dots \vee p(s'_n, t'_n)$ iff $s\sigma = s'_i \wedge t\sigma =_E t'_i$ for some i, σ .
- (2) $B' \wedge p(s, t) \models F^+$ iff $h^+(s, t, T^+)$.
- (3) $B' \wedge p(s, t) \wedge F^- \not\models \text{false}$ iff $h^-(s, t, T^-)$.

Lemma 18 (Syntactic criterion).

- (1) If $\text{var}(t) \subseteq \text{var}(s)$, then the hypothesis $p(s, t)$ is weakly determinate.
- (2) Each weakly determinate hypothesis $p(s, t)$ has a weakly determinate instance $p(s\sigma, t\sigma)$ with $\text{var}(t\sigma) \subseteq \text{var}(s\sigma)$ and $B' \models p(s, t) \leftrightarrow p(s\sigma, t\sigma)$.

Proof.

- (1) If $B' \wedge p(s, t) \models p(s'_1, t'_1) \wedge p(s'_2, t'_2)$, we have $s\sigma_1 = s' = s\sigma_2 \wedge t\sigma_1 =_E t'_1 \wedge t\sigma_2 =_E t'_2$ by Lemma 17(2). Hence, σ_1 and σ_2 coincide on $\text{var}(s) \supseteq \text{var}(t)$, and we get $t'_1 =_E t\sigma_1 = t\sigma_2 =_E t'_2$.
- (2) Define $\sigma = \{x \mapsto a \mid x \in \text{var}(t) \setminus \text{var}(s)\}$, where a is an arbitrary constant. Then $\text{var}(t\sigma) \subseteq \text{var}(s\sigma)$ by construction. Since $B' \models p(s, t) \rightarrow p(s\sigma, t\sigma)$, we have the situation that $p(s\sigma, t\sigma)$ is again weakly determinate.
Since $s\sigma = s$, $B' \wedge p(s, t) \models p(s, t) \wedge p(s, t\sigma)$. Since $p(s, t)$ is weakly determinate, this implies $t =_E t\sigma$. Since B' ensures that p is E -compatible in its right argument, we have $B' \wedge p(s\sigma, t\sigma) \models p(s, t)$. \square

Lemma 19 (Equivalence for constructor terms). *Let s be a constructor term.*

- (1) $p(s, t)$ is a weakly determinate hypothesis iff it is a determinate one.
- (2) $B \wedge p(s, t) \models p(s', t')$ iff $B' \wedge p(s, t) \models p(s', t')$, if s' is an instance of s .
- (3) $B \wedge p(s, t) \models p(s', t')$ iff $B' \wedge p(s, t) \models p(s', t')$, if s' is a constructor term.

Proof. All *if* directions follow from $B \models B'$.

- (1) Obtain some σ from Lemma 18(2) such that $p(s\sigma, t\sigma)$ is again weakly determinate, $\text{var}(s\sigma) \supseteq \text{var}(t\sigma)$, and $B' \models p(s, t) \leftrightarrow p(s\sigma, t\sigma)$. From the latter, we get $B \models p(s, t) \leftrightarrow p(s\sigma, t\sigma)$.
Hence, if $B \wedge p(s, t) \wedge s'_1 =_E s'_2 \models p(s'_1, t'_1) \wedge p(s'_2, t'_2)$ holds, we have $s\sigma\sigma_1 =_E s'_1 =_E s'_2 =_E s\sigma\sigma_2$ and $t\sigma\sigma_1 =_E t'_1 \wedge t\sigma\sigma_2 =_E t'_2$ for some σ_1, σ_2 by Lemma 13(2). Since s is a constructor term, we have $x\sigma\sigma_1 =_E x\sigma\sigma_2$ for each $x \in \text{var}(s\sigma) \supseteq \text{var}(t\sigma)$. Hence, $t'_1 =_E t\sigma\sigma_1 =_E t\sigma\sigma_2 =_E t'_2$.
- (2) Obtain $s\chi =_E s'$ and $t\chi =_E t'$ for some χ from Lemma 13(2) and the definition of h^+ . Since $s\sigma' = s'$ for some σ' , we have $s\sigma' =_E s\chi$. Since s is a constructor term, we have $x\sigma' =_E x\chi$ for all $x \in \text{var}(s) \supseteq \text{var}(t)$. Hence, $t\sigma' =_E t\chi =_E t'$. By Lemma 17(2), this implies $B' \wedge p(s, t) \models p(s', t')$.

- (3) Follows from (2), since $s\chi =_E s'$ implies that s' is an instance of s . \square

Lemma 19 ends our little detour. It ensures that weak and ordinary determinacy coincide if we supply only constructor terms to the *input* argument of a hypothesis p . On the one hand, this is a restriction because we cannot learn a hypothesis like $p(x * x, x)$, which defines a partial function realizing the integer square root. On the other hand, it is often desirable that a hypothesis correspond to an *explicit* definition, i.e., that it can be applied like a rewrite rule to a term s by purely syntactical pattern matching. Tuples built using the operator $\langle \dots \rangle$ are the most frequently occurring special cases of constructor terms. For example, a hypothesis $p(\langle x, y \rangle, x + 2 * y)$ may be preferred to $p(\langle 2 * x, y \rangle, 2 * (x + y))$ because the former is explicit and implies the latter wrt E from Fig. 1. Lemma 19(2) allows us to instantiate $\langle x, y \rangle$ from the former hypothesis arbitrarily, even with non-constructor terms like $\langle 2 * 1, z_1 + z_2 \rangle$.

The following lemma corresponds to Lemma 14, but leads to reduced time complexity. It does not need to compute universal substitutions because it uses constrained E -generalization from Definition 2. It still permits negative examples, handling them more efficiently than Lemma 14. They may make sense even if only determinate predicates are to be learned because they allow us to exclude certain undesirable hypotheses without committing to a fixed function behavior.

Lemma 20 (Weakly determinate hypotheses). *For each finite set of ground term pairs $T^+ \cup T^-$, we can compute a regular set $H = H_{20}(T^+, T^-)$ such that for all $s, t \in \mathcal{T}_Y$:*

$$\begin{aligned} \langle s, t \rangle \in H &\Rightarrow h^+(s, t, T^+) \wedge h^-(s, t, T^-) \wedge \text{var}(s) \supseteq \text{var}(t), \quad \text{and} \\ \exists \sigma: \langle s\sigma, t\sigma \rangle \in H &\Leftarrow h^+(s, t, T^+) \wedge h^-(s, t, T^-) \wedge \text{var}(s) \supseteq \text{var}(t). \end{aligned}$$

Proof. Assume $T^+ = \{\langle s_i, t_i \rangle \mid i = 1, \dots, n\}$ and $T^- = \{\langle s_i, t_i \rangle \mid i = n+1, \dots, m\}$. For $\{1, \dots, n\} \subseteq I \subseteq \{1, \dots, m\}$, let s_I be the most specific syntactical generalization of $\{s_i \mid i \in I\}$, with $s_I \sigma_{I,i} = s_i$ for each $i \in I$. Such an I is called *maximal* if $\forall \{1, \dots, n\} \subseteq I' \subseteq \{1, \dots, m\}: s_I = s_{I'} \Rightarrow I' \subseteq I$, where $(=)$ denotes term equality up to renaming.

For example, if $T^+ = \{\langle a+a, t_a \rangle\}$ and $T^- = \{\langle b+b, t_b \rangle, \langle b+c, t_c \rangle\}$, then $\{1, 2\}$ and $\{1, 2, 3\}$ are maximal, but $\{1, 3\}$ is not. Since $s_{\{1,2\}} = x + x$ can be instantiated to $a + a$ and $b + b$, we must merely ensure that $t_{\{1,2\}}\{x \mapsto a\} \neq_E t_a$ and $t_{\{1,2\}}\{x \mapsto b\} \neq_E t_b$ in order to obtain $h^-(s_{\{1,2\}}, t_{\{1,2\}}, T^-)$. However, for $s_{\{1,3\}} = x + y$, it is not sufficient to ensure $t_{\{1,3\}}\{x \mapsto a, y \mapsto a\} \neq_E t_a$ and $t_{\{1,3\}}\{x \mapsto b, y \mapsto c\} \neq_E t_c$. Since $s_{\{1,3\}}$ happens to be instantiable to $b+b$ as well, $h^-(s_{\{1,3\}}, t_{\{1,3\}}, T^-)$ could be violated if $t_{\{1,3\}}\{x \mapsto b, y \mapsto b\} =_E t_b$. Therefore, only generalizations s_I of maximal I should be considered.

Let \mathcal{I} be the set of all maximal I . For $I \in \mathcal{I}$, let $T_I = \bigcap_{i=1}^n [t_i]_E^{\sigma_{I,i}} \setminus \bigcup_{i \in I, i > n} [t_i]_E^{\sigma_{I,i}}$. Each such set T_I can be computed from $[t_1]_E, \dots, [t_m]_E$ by standard tree grammar algorithms. Given the grammar for each T_I , it is easy to compute a grammar for their *tagged union* $H = \{\langle s_I, t_I \rangle \mid I \in \mathcal{I} \wedge t_I \in T_I\}$. To prove the properties of H , first observe the following:

- (1) We always have $\text{var}(t_I) \subseteq \text{dom } \sigma_{I,1} \subseteq \text{var}(s_I)$. The first inclusion follows from $t_I \in T_I \subseteq [t_1]_E^{\sigma_{I,1}}$, the second from the definition of $\sigma_{I,1}$.

- (2) If I is maximal and $s_I \sigma = s_i$ for some $i \in \{1, \dots, m\}$ and σ , then $i \in I$:

Since $s_I \sigma_{I,j} = s_j$ for $j \in I$ and $s_I \sigma = s_i$, the term s_I is a common generalization of the set $\{s_j \mid j \in I\} \cup \{s_i\}$. Hence, its most special generalization, viz. $s_{I \cup \{i\}}$, is an instance of s_I . Conversely, $s_{I \cup \{i\}}$ is trivially a common generalization of $\{s_j \mid j \in I\}$; hence s_I is an instance of $s_{I \cup \{i\}}$. Therefore, $s_{I \cup \{i\}} = s_I$, which implies $i \in I$ because I is maximal.

- If $I \in \mathcal{I}$ and $t_I \in T_I$, then trivially $s_I \sigma_{I,i} = s_i$ and $t_I \sigma_{I,i} =_E t_i$ for each $i \leq n$. Assume $s_I \sigma = s_i$ and $t_I \sigma =_E t_i$ for some σ and some $i > n$. By (2), we have $i \in I$, and therefore $s_I \sigma_{I,i} = s_i$. Hence, $\sigma_{I,i}$ and σ coincide on $\text{var}(s_I) \supseteq \text{var}(t_I)$, using (1). We get $t_I \sigma_{I,i} = t_I \sigma =_E t_i$, which contradicts $t_I \notin [t_i]_E^{\sigma_{I,i}}$.
- If s, t are given such that $h^+(s, t, T^+)$ and $h^-(s, t, T^-)$ hold, let $I = \{1, \dots, n\} \cup \{i \mid n < i \leq m \wedge \exists \sigma'_i: s \sigma'_i = s_i\}$. Then, s is a common generalization of $\{s_i \mid i \in I\}$, and we have $s \sigma = s_I$ for some σ .

We show $I \in \mathcal{I}$: Let I' be such that $s_{I'} = s_I$ and let $i \in I'$, then $s \sigma \sigma_{I',i} = s_I \sigma_{I',i} = s_{I'} \sigma_{I',i} = s_i$, hence $i \in I$. Since i was arbitrary, we have $I' \subseteq I$, i.e., I is maximal.

For $i \leq n$, we have $s \sigma \sigma_{I,i} = s_I \sigma_{I,i} = s_i = s \sigma_i$. In other words, $\sigma \sigma_{I,i}$ and the σ_i obtained from $h^+(s, t, T^+)$ coincide on $\text{var}(s) \supseteq \text{var}(t)$. Hence, $t \sigma \sigma_{I,i} = t \sigma_i =_E t_i$, i.e., $t \sigma \in [t_i]_E^{\sigma_{I,i}}$. For $i > n$ and $i \in I$, we still have $s \sigma \sigma_{I,i} = s_i$, as above. Hence $t \sigma$ cannot be a member of $[t_i]_E^{\sigma_{I,i}}$. Therefore, $\langle s \sigma, t \sigma \rangle \in H$. \square

Theorem 21 (Atomic determinate definitions). *Let $F^+ \Leftrightarrow \bigwedge_{i=1}^n p(s_i, t_i)$ and $F^- \Leftrightarrow \bigwedge_{i=n+1}^m \neg p(s_i, t_i)$ be given such that each t_i is ground and each s_i is a ground constructor term. Then, we can compute a regular set $H = H_{21}(F^+, F^-)$ such that*

- each $p(s, t) \in H$ is a determinate hypothesis satisfying the Sufficiency and the Strong Consistency requirement wrt F^+, F^- ; and
- for each determinate hypothesis satisfying these requirements and having the form $p(s, t)$ with s a constructor term, we have $p(s \sigma, t \sigma) \in H$ for some σ .

Proof. Let $T^+ = \{\langle s_i, t_i \rangle \mid i = 1, \dots, n\}$ and $T^- = \{\langle s_i, t_i \rangle \mid i = n+1, \dots, m\}$. Define $H = \{p(s, t) \mid \langle s, t \rangle \in H_{20}(T^+, T^-)\}$, which is again a regular tree language.

- If $p(s, t) \in H$, then $\langle s, t \rangle \in H_{20}(T^+, T^-)$, i.e., $h^+(s, t, T^+)$, $h^-(s, t, T^-)$ and $\text{var}(s) \supseteq \text{var}(t)$ hold. By Lemma 18(1), $p(s, t)$ is weakly determinate; by Lemma 17(2) and (3), it satisfies the requirements wrt B' . By construction of Lemma 20, s is a constructor term. Hence, by Lemma 19(1) and (3), $p(s, t)$ is determinate and satisfies the requirements wrt B , respectively.
- Let $p(s, t)$ be a determinate hypothesis satisfying the requirements wrt B , where s is a constructor term. By Lemma 19(1) and (3), it is also weakly determinate and satisfies the requirements wrt B' , respectively. Obtain σ from Lemma 18(2) such that $p(s \sigma, t \sigma)$ additionally satisfies $\text{var}(t \sigma) \subseteq \text{var}(s \sigma)$. By Lemma 17(2) and (3), we then have $h^+(s \sigma, t \sigma, T^+)$ and $h^-(s \sigma, t \sigma, T^-)$, respectively. By Lemma 20, we have $\langle s \sigma \sigma', t \sigma \sigma' \rangle \in H_{20}(T^+, T^-)$ for some σ' , i.e., $p(s \sigma \sigma', t \sigma \sigma') \in H$.

To compute H , the union of up to $(m - n) \cdot 2^{m-n}$, the intersection of n and the difference between two grammars are needed. No additional grammar intersection is necessary to compute any universal substitution. \square

Examples of the application of Theorem 21 are given in Section 5.

We now show that learning a semi-determinate clause by lgg can be simulated by learning an equivalent constrained clause using E -generalization. By analogy to the above, obtain the background theory B' from B by replacing the full congruence axiom for p_0 with a partial one. Lemma 22 shows how a semi-determinate clause C can be transformed into an equivalent constrained clause $dlr(C)$. Theorem 23 simulates lgg -learning of C by lgg_E^c -learning of $dlr(C)$.

Lemma 22 (Determinate literal removal). *Let a semi-determinate clause $C \Leftrightarrow (p_0(s_0, t_0) \leftarrow \bigwedge_{i=1}^n p_i(s_i, x_i) \wedge \bigwedge_{i=1}^m q_i(t_i))$ be given such that $p_i(s_i, x_i) \Leftrightarrow g_i(s_i) =_E x_i$. Let $\sigma = \{x_n \mapsto g_n(s_n)\} \dots \{x_1 \mapsto g_1(s_1)\}$. Then, $dlr(C) \Leftrightarrow (p_0(s_0\sigma, t_0\sigma) \leftarrow \bigwedge_{i=1}^m q_i(t_i\sigma))$ is a constrained clause that defines the same relation for p_0 wrt B' , and hence also wrt B .*

Proof. From the properties of semi-determinacy, we have $s_0\sigma = s_0$. Since p_0 does not occur in B' outside its partial congruence axiom, we can use the following property of SLD resolution [11]: $B' \wedge (p_0(s_0, t_0) \leftarrow C) \models p_0(s, t)$ iff $s_0\sigma' = s \wedge t_0\sigma' =_E t$ and $B' \models C\sigma'$ for some σ' . A similar property holds for $dlr(C)$.

The proofs of both directions are based on establishing $x_i\sigma\sigma' =_E x_i\sigma'$. This property follows from $p_i(s_i\sigma', x_i\sigma')$ and the definitions of g_i and σ , when $(B' \wedge C \models p_0(s, t)) \Rightarrow (B' \wedge dlr(C) \models p_0(s, t))$ is proved. When the converse direction is shown, it is established by extending σ' to $\text{var}(dlr(C)) \cup \{x_1, \dots, x_n\}$ defining $x_i\sigma' = x_i\sigma\sigma'$. \square

Theorem 23 (Clausal determinate definitions). *We use the abbreviation $D = \{\neg p_i(s, t) \mid s, t \in \mathcal{T}_\emptyset \wedge p_i \text{ determinate} \wedge B' \models p_i(s, t)\}$. Let two ground Horn clauses C_1 and C_2 be given, such that each body literal of each C_i is entailed by B' and is not an element of D . We can compute a regular tree language $H = lgg_E^c(C_1, C_2)$ such that:*

- each member $C \in H$ is a constrained clause that E -subsumes C_1 and C_2 ;
- and for each semi-determinate clause $C \subseteq lgg(C_1 \cup C'_1, C_2 \cup C'_2)$ with $C'_1, C'_2 \subseteq D$, $dlr(C)$ subsumes some member of H .

Proof. For $i = 1, 2$, let $p_0(s_{0i}, t_{0i})$ be the head literal of C_i . Let M be the set of all pairs $\langle L_1, L_2 \rangle$ of body literals L_1 from C_1 and L_2 from C_2 such that L_1 fits L_2 . Assuming $M = \{\langle q_j(t_{j1}), q_j(t_{j2}) \rangle \mid j = 1, \dots, k\}$, define $T^+ = \{\langle s_{01}, \langle t_{01}, t_{11}, \dots, t_{k1} \rangle \rangle, \langle s_{02}, \langle t_{02}, t_{12}, \dots, t_{k2} \rangle \rangle\}$ and $T^- = \{\}$. Define $H = \{p_0(s_0, t_0) \leftarrow q_1(t_1) \wedge \dots \wedge q_k(t_k) \mid \langle s_0, \langle t_0, t_1, \dots, t_k \rangle \rangle \in H_{20}(T^+, T^-)\}$.

H is again a regular tree language because $H_{20}(T^+, T^-)$ is the image of H under the tree homomorphism that maps the term $p_0(x_0, y_0) \leftarrow q_1(y_1) \wedge \dots \wedge q_k(y_k)$ to $\langle x_0, \langle y_0, y_1, \dots, y_k \rangle \rangle$. Since $T^- = \{\}$, the set $H_{20}(T^+, T^-)$ contains at least one element $\langle s_0, \langle t'_0, t'_1, \dots, t'_k \rangle \rangle$, and the left component of an element of $H_{20}(T^+, T^-)$ is always s_0 .

- For each clause $p_0(s_0, t_0) \leftarrow q_1(t_1) \wedge \dots \wedge q_k(t_k)$ in H , we have $\text{var}(s_0) \supseteq \text{var}(t_0, t_1, \dots, t_k)$ and $\text{h}^+(s_0, \langle t_0, t_1, \dots, t_k \rangle, T^+, T^-)$ by Theorem 20. Hence, $\{p_0(s_0, t_0), \neg q_1(t_1), \dots, \neg q_k(t_k)\} \chi_i \subseteq_E C_i$.
- Assume

$$\begin{aligned} & (p_0(s_0, t_0) \leftarrow p_1(s_1, x_1) \wedge \dots \wedge p_n(s_n, x_n) \wedge q_1(t_1) \wedge \dots \wedge q_m(t_m)) \\ & \subseteq \text{lgg}(C_1 \cup C'_1, C_2 \cup C'_2) \end{aligned}$$

is a semi-determinate clause. Then, $(\neg q_j(t_j \sigma_i)) \in C_i$ for some σ_i —we assume wlog $t_j \sigma_i =_E t_{ji}$. Moreover, $\neg p_j(s_j \sigma_i, x_j \sigma_i)$ is a member of $C'_i \subseteq D$, implying that $x_j \sigma_i =_E g_j(s_j \sigma_i) = x_j \sigma_i$ is entailed by B' , where σ denotes the substitution from $\text{dlr}(C)$ computation by Lemma 22. Since $\text{dom } \sigma = \{x_1, \dots, x_n\}$, we have $x \sigma \sigma_i =_E x \sigma_i$ for all variables x . Therefore, $t_j \sigma \sigma_i =_E t_j \sigma_i =_E t_{ji}$, and $s_0 \sigma \sigma_i = s_0 \sigma_i = s_{0i}$ because $\text{var}(s_0)$ is disjoint from the domain of σ . Hence, we can extend the clause $\text{dlr}(C) = \{p_0(s_0 \sigma, t_0 \sigma), \neg q_1(t_1 \sigma), \dots, \neg q_m(t_m \sigma)\}$ to some superset $\{p_0(s_0 \sigma, t_0 \sigma), \neg q_1(t_1 \sigma), \dots, \neg q_m(t_m \sigma), \neg q_{m+1}(t'_{m+1}), \dots, \neg q_k(t'_k)\}$ that satisfies h^+ and $\text{var}(s_0 \sigma) \supseteq \text{var}(t_j \sigma) \cup \text{var}(t'_j)$ and is therefore a member of H .

To compute lgg_E^c , the grammar defining $[t_{ji}]_E$ must be extended by two rules to define $[\langle s_{0i}, \langle t_{0i}, t_{1i}, \dots, t_{ki} \rangle \rangle]_E$ as well. One intersection of the two extended grammars is needed; no universal substitution needs to be computed. \square

The form of Theorem 23 differs from that of Theorem 16 because neither C nor $\text{dlr}(C)$ need E -subsume the other. To establish some similarity between the second assertion of the two theorems, note that a subsumed clause defining a predicate leads to a more specific definition than its subsuming clause. Let C' subsume C_1 and C_2 and contain a nontrivial head literal $p_0(\dots)$. Then C' also subsumes $C = \text{lgg}(C_1, C_2)$. By Theorem 23, $\text{dlr}(C)$ subsumes some member of $\text{lgg}_E^c(C_1, C_2)$. That member thus leads to a more specific definition of p_0 than C' .

In order to duplicate a most flexible lgg approach, Theorem 23 allows a literal pre- and postselection strategy, to be applied before and after lgg computation, respectively. Both may serve to eliminate undesirable body literals from the lgg result clause. Preselection can be modeled using the C_i and C'_i , while postselection is enabled by choosing $C \subsetneq \text{lgg}(C_1 \cup C'_1, C_2 \cup C'_2)$. In all cases, Theorem 23 provides a corresponding constrained clause from $\text{lgg}_E^c(C_1, C_2)$, which is equivalent to, or more specific than, C .

Similar to Theorem 16, each $C \in \text{lgg}_E^c(C_1, C_2)$ is sufficient wrt $F^+ \Leftrightarrow C_1 \wedge C_2$ and $F^- \Leftrightarrow \text{true}$. Each such C is consistent if some predicate symbol q occurs in both C_1 and C_2 , but not in B , except for its congruence axiom.

Again similar to the nondeterminate case, $\text{H}_{21}(p_0(s_{01}, t_{01}) \wedge p_0(s_{02}, t_{02}), \text{true})$ equals $\text{lgg}_E^c(\{p_0(s_{01}, t_{01})\}, \{p_0(s_{02}, t_{02})\})$ from Theorem 23. In this common special case, Theorem 21, but not Theorem 23, ensures that the result set contains *all* sufficient hypotheses.

On the other hand, Theorem 23 ensures that for each purely determinate clause, i.e., a clause without any nondeterminate q_i in its body, $\text{lgg}_E^c(C_1, C_2)$ contains a clause leading to an equivalent, or more specific, definition of p_0 . In other words, lgg -learning of purely determinate clauses can be simulated by lgg_E^c -learning of atoms.

F	$p_0(b, bbb) \wedge p_0(\varepsilon, b)$		
P	$a(\varepsilon, y, y)$ $a([v \mid x], y, [v \mid z]) \leftarrow a(x, y, z)$	$a(\varepsilon, y) = y$ $a(x, \varepsilon) = x$ $a(a(x, y), z) = a(x, a(y, z))$	E
K_q	$q(\varepsilon, d) \wedge q(b, d) \wedge q(c, e) \wedge q(bb, d) \wedge q(bc, e) \wedge q(bcb, e)$		
K_a	$a(\varepsilon, \varepsilon, \varepsilon) \wedge$ $a(\varepsilon, b, b) \wedge a(b, \varepsilon, b) \wedge$ $a(\varepsilon, bb, bb) \wedge a(b, b, bb) \wedge \dots$ $a(\varepsilon, bbb, bbb) \wedge a(b, bb, bbb) \wedge \dots$	$N_\varepsilon ::= \varepsilon \mid a(N_\varepsilon, N_\varepsilon)$ $N_b ::= b \mid a(N_\varepsilon, N_b) \mid a(N_b, N_\varepsilon)$ $N_{bb} ::= a(N_\varepsilon, N_{bb}) \mid a(N_b, N_b) \dots$ $N_{bbb} ::= a(N_\varepsilon, N_{bbb}) \mid a(N_b, N_{bb}) \dots$	\mathcal{G}
lgg	$p_0(v_{b,\varepsilon}, v_{bbb,b})$ $\leftarrow a(v_{b,\varepsilon}, v_{b,\varepsilon}, v_{bb,\varepsilon})$ $\wedge a(v_{bb,\varepsilon}, b, v_{bbb,b})$ $\wedge q(v_{bb,\varepsilon}, d)$	$p_0(v_{b,\varepsilon}, a(a(v_{b,\varepsilon}, v_{b,\varepsilon}), b))$ $\leftarrow q(a(v_{b,\varepsilon}, v_{b,\varepsilon}), d)$	lgg_E^c

Fig. 5. Comparison of ILP using lgg and lgg_E^c .

Let us compare the ILP methods using lgg and lgg_E^c in an example. Assume part of the background knowledge describes lists with an associative append operator and a neutral element ε (nil). Lines 2–3 of Fig. 5 show a Horn program P and an equational theory E , each of which formalizes that knowledge, where v, x, y, z denote variables, a denotes *append* and b, c, d, e below will denote some constants. Moreover, let a conjunction K_q of facts about a predicate q be given, as shown in the fourth line of Fig. 5. We abbreviated, e.g., $q([b, c, b], [e])$ to $q(bcb, e)$. Let $F \Leftrightarrow p_0(b, bbb) \wedge p_0(\varepsilon, b)$ be given. Let us assume for now that a preselection strategy chose $K'_q \Leftrightarrow q(\varepsilon, d) \wedge q(bb, d)$.

Neither lgg nor lgg_E^c can use the first part of background knowledge directly. Most ILP systems, including GOLEM, restrict background theories to sets of ground literals. Hence, they cannot directly use equality as background knowledge because this would require formulas like $p_0(x, y) \leftarrow eq(y, y') \wedge p_0(x, y')$ in the background theory. While Plotkin's lgg is also defined for nonground clauses, it has not been defined for clause sets like P . Moreover, since F contains only ground literals, all relevant arguments of body literals must be ground to obtain the necessary variable bindings in the generalized clause. Thus, we have to derive the conjunction K_a of all ground facts implied by P that could be relevant in any respect.

On the other hand, E has to be transformed into a grammar \mathcal{G} in order to compute lgg_E^c . We can do this by Theorem 11 with (\prec) as the lexicographic path ordering, which is commonly used to prove termination of the rewrite system associated with E [13, Section 5.3]. Alternatively, we could instantiate a predefined grammar scheme like $N_{x_1 \dots x_n} ::= \mid_{n=0} \varepsilon \mid \mid_{n=1} x_1 \mid \mid_{i=0}^n a(N_{x_1 \dots x_i}, N_{x_{i+1} \dots x_n})$. At least we do not have to rack our brains over the question of which terms might be relevant—it is sufficient to define the congruence classes of all terms occurring in F or K'_q .

Lines 5–8 of Fig. 5 show the preprocessed form K_a and \mathcal{G} of P and E , respectively. Observe that a ground literal $a(s, t, u)$ in the left column corresponds to a grammar alternative $N_u ::= \dots a(N_s, N_t)$ in the right one. It is plausible to assume that there are at least as many literals in K_a as there are alternatives in \mathcal{G} . Next, we compute

$$lgg(p_0(b, bbb) \leftarrow K_a \wedge K'_q, (p_0(\varepsilon, b) \leftarrow K_a \wedge K'_q)) \quad \text{and}$$

$$lgg_E^c((p_0(b, bbb) \leftarrow K'_q), (p_0(\varepsilon, b) \leftarrow K'_q))$$

and apply some literal postselection strategy. A sample result is shown in the bottom part of Fig. 5. More precisely, lgg_E^c results in the set of all terms $p_0(v_{b,\varepsilon}, t_{bbb,b}) \leftarrow q(t_{bb,\varepsilon}, d)$ for any $t_{bbb,b} \in [bbb]_E^{[v_{b,\varepsilon} \mapsto b]} \cap [b]_E^{[v_{b,\varepsilon} \mapsto \varepsilon]}$ and $t_{bb,\varepsilon} \in [bb]_E^{[v_{b,\varepsilon} \mapsto b]} \cap [\varepsilon]_E^{[v_{b,\varepsilon} \mapsto \varepsilon]}$. The choice of $t_{bbb,b}$ and $t_{bb,\varepsilon}$ on the right-hand side in Fig. 5 corresponds to the choice of body literals about a on the left; both sides are equivalent definitions of p_0 . If the postselection strategy chooses $a(v_{b,\varepsilon}, b, v_{bb,b}) \wedge a(v_{bb,b}, v_{b,\varepsilon}, v_{bbb,b}) \wedge a(v_{b,\varepsilon}, v_{b,\varepsilon}, v_{bb,\varepsilon})$ on the left, we need only to choose $t_{bbb,b} = a(a(v_{b,\varepsilon}, b), v_{b,\varepsilon})$ on the right to duplicate that result. However, if preselection chooses different literals about q , e.g., $K'_q \Leftrightarrow q(bc, e) \wedge q(c, e)$, we have to recompute the grammar \mathcal{G} to include definitions for $[bc]_E$ and $[c]_E$.

The lgg_E^c result clause is always a constrained one, whereas lgg yields a determinate clause. The reason for the latter is that function calls have to be simulated by predicate calls, requiring extra variables for intermediate results. The deeper a term in the lgg_E^c clause is nested, the longer the extra variable chains are in the corresponding lgg clause. If $K'_q \Leftrightarrow true$ is chosen, lgg_E^c yields an atom rather than a proper clause.

When the lgg_E^c approach is used, the hypotheses search space is split. Literal pre- and postselection strategies need to handle nondeterminate predicates only. The preselected literals, i.e., K'_q , control the size and form of the grammar \mathcal{G} . Choices of, e.g., $t_{bbb,b} \in \mathcal{L}_{\mathcal{G}}(N_{bbb,b})$ can be made independently of pre- and postselection, each choice leading to the condensed equivalent of a semi-determinate clause.

Filtering of, e.g., $\mathcal{L}_{\mathcal{G}}(N_{bbb,b})$ allows us to ensure additional properties of the result clause if they can be expressed by regular tree languages. For example, orienting each equation from E in Fig. 5 left to right generates a canonical term-rewriting system R . Since all terms in E are linear, a grammar \mathcal{G}_{NF} for the set of normal forms wrt R can be obtained automatically from E . Choosing, e.g., $t_{bbb,b} \in \mathcal{L}_{\mathcal{G}}(N_{bbb,b}) \cap \mathcal{L}_{\mathcal{G}_{NF}}(N_{NF})$ ensures that no redundant clause like $p_0(v_{b,\varepsilon}, a(v_{b,\varepsilon}, a(v_{b,\varepsilon}, b))) \leftarrow q(a(v_{b,\varepsilon}, v_{b,\varepsilon}), d)$ can result. In classical ILP, there is no corresponding filtering method of similar simplicity.

5. Applications

In this section, we apply E -generalization in three different areas. In all cases, we use the paradigm of learning a determinate atomic definition from positive examples only. As indicated in Section 4, this is the most restrictive paradigm, and it therefore allows the most efficient algorithms. We intend to demonstrate that the notion of E -generalization can help to solve even comparatively ambitious tasks in Artificial Intelligence at the first attempt. We make no claim to develop a single application to full maturity. Instead, we cover a variety of different areas in order to illustrate the flexibility of E -generalization.

5.1. Candidate lemmas in inductive proofs

Auxiliary lemmas play an important role in automated theorem proving. Even in pure first-order logic, where lemmas are not strictly necessary [18], proofs may become exponentially longer without them and are consequently harder to find. In induction proofs,

which exceed first-order logic owing to the induction axiom(s), using lemmas may be unavoidable to demonstrate a certain theorem.

By way of a simple example, consider an induction proof of the multiplicative associativity law using the equational theory from Fig. 1. In the inductive case, after applying equation 4. from Fig. 1 and the induction hypothesis, the distributivity law is needed as a lemma in order to continue the proof. While this is obvious to a mathematically experienced reader, an automated prover that does not yet know the law will get stuck at this point and require a user interaction because the actual term cannot be rewritten any further. In this simple example, where only one lemma is required, the *cross-fertilization* technique of [3] would suffice to generate it automatically. However, this technique generally fails if several lemmas are needed.

In such cases, we try to simulate mathematical *intuition* by *E*-generalization in order to find a useful lemma and allow the prover to continue; i.e., to increase its level of automation. We consider the last term t_1 obtained in the proof so far ($x * (y * z') + x * y$ in the example) and try to find a new lemma that could be applied next by the prover. We are looking for a lemma of the form $t_1 \equiv_E t_2$ for some t_2 such that $\forall \sigma$ ground: $t_1 \sigma \equiv_E t_2 \sigma$ holds. Using Theorem 21, we are able to compute the set of all terms t_2 such that $t_1 \sigma \equiv_E t_2 \sigma$ holds at least for finitely many given σ .

We therefore choose some ground substitutions $\sigma_1, \dots, \sigma_n$ with $\text{var}(t_1) = \{x, y, z'\}$ as their domain, and let $F^+ \Leftrightarrow \bigwedge_{i=1}^n p(\langle x\sigma_i, y\sigma_i, z'\sigma_i \rangle, t_1\sigma_i)$. We then apply Theorem 21 to this F^+ and $F^- \Leftrightarrow \text{true}$. (See Fig. 6, where the partial congruence property of p was used to simplify the examples in F^+ .) Using the notation from Lemma 20, we have just one I in \mathcal{I} , viz. $I = \{1, \dots, n\}$, since we do not supply negative examples. Therefore, we only have to compute $T_I = \bigcap_{i=1}^n [t_1\sigma_i]_E^{\sigma_i}$.

The most specific syntactical generalization s_I of $\{\langle x, y, z' \rangle \sigma_1, \dots, \langle x, y, z' \rangle \sigma_n\}$ need not be $\langle x, y, z' \rangle$ again. However, we always have $\langle x, y, z' \rangle \sigma' = s_I$ for some σ' . If we choose σ_i that are *sufficiently different*, we can ensure that σ' has an inverse. This is the case in Fig. 6, where $\sigma' = \{x \mapsto v_{021}, y \mapsto v_{310}, z' \mapsto v_{201}\}$. By Theorem 21, $B' \wedge p(\langle x, y, z' \rangle, t_I \sigma'^{-1})$ implies $p(\langle x, y, z' \rangle \sigma_i, t_1 \sigma_i)$ for each $t_1 \in T_I$ and $i = 1, \dots, n$. Since it trivially also implies $p(\langle x, y, z' \rangle \sigma_i, t_I \sigma'^{-1} \sigma_i)$, we obtain $t_1 \sigma_i \equiv_E t_I \sigma'^{-1} \sigma_i$ using determinacy.

Therefore, defining $t_2 = t_I \sigma'^{-1}$ ensures that t_1 and t_2 have the same value under each sample substitution σ_i . This is a necessary condition for $t_1 \equiv_E t_2$, but not sufficient. Before using a lemma suggestion $t_1 \equiv_E t_2$ to continue the original proof, it must be checked for

$$\begin{array}{lcl}
 t_1 & = & x * (y * z') + x * y \\
 & & \hline
 & & p(\langle x\sigma_i, y\sigma_i, z'\sigma_i \rangle, \langle x * (y * z') + x * y \rangle \sigma_i) \\
 F^+ & \Leftrightarrow & \begin{array}{l} p(\langle 0, s^3(0), s^2(0) \rangle, 0) \\ \wedge p(\langle s^2(0), s(0), 0 \rangle, s^2(0)) \\ \wedge p(\langle s(0), 0, s(0) \rangle, 0) \end{array} \\
 & & \hline
 H & \ni & p(\langle v_{021}, v_{310}, v_{201} \rangle, v_{021} * (v_{310} * v_{201} + v_{310})) \\
 t_2 & = & x * (y * z' + y)
 \end{array}$$

Fig. 6. Generation of lemma candidates by Theorem 21.

validity by a recursive call to the induction prover itself. Two simple restrictions can help to eliminate unsuccessful hypotheses:

- Usually, only those equations $t_1 \equiv_E t_2$ are desired that satisfy $\text{var}(t_2) \subseteq \text{var}(t_1)$. For example, it is obvious that $x * (y * z') + x * y \equiv_E x + v_{123}$ is not universally valid. This restriction of the result set is already built into Theorem 21. Any lemma contradicting this restriction will not appear in the grammar language. However, all its instances that satisfy the restriction will appear.
- Moreover, if E was given by a ground-convergent term-rewriting system R [13, Section 2.4], it makes sense to require t_2 to be in normal form wrt R . For example, $x * (y * z') + x * y \equiv_E (x + 0) * (y * z' + y * s(0))$ is a valid lemma, but redundant, compared with $x * (y * z') + x * y \equiv_E x * (y * z' + y)$. The closed representation of the set T_I as a regular tree language allows us to easily eliminate such undesired terms t_2 . For left-linear term-rewriting systems [13, Section 2.3], the set of all normal-form terms is always representable as a regular tree language; hence terms in non-normal form can be eliminated by intersection. For rewriting systems that are not left-linear, we may still filter out a subset of all non-normal-form terms. If desired by some application, T_I could also be restricted to those terms t_I that satisfy $V' \subseteq \text{var}(t_I) \subseteq V$ for arbitrarily given variable sets V', V .

The more sample instances are used, the more of the enumerated lemma candidates will be valid. However, our method does not lead to *learnability in the limit* [19] because normally any result language will still contain invalid equations—regardless of the number of sample substitutions. It does not even lead to *PAC-learnability* [39], there currently being no way to compute the number of sample substitutions depending on the required δ and ϵ accuracies.

In the associativity law example from above, we get, among other equations, the lemma suggestion $x * (y * z') + x * y \equiv_E x * (y * z' + y)$, which allows the prover to continue successfully. This suggestion appears among the first ten, if T_I is enumerated by increasing term size. Most of the earlier terms are variants wrt commutativity, like $x * (y * z') + x * y \equiv_E (y + y * z') * x$.

Fig. 7 shows some examples of lemma candidates generated by our prototypical implementation (see Section 5.4). The column *Theory* shows the equational theory used. We distinguish between the truncating integer division ($//$) and the true division ($/$). For example, $7 // 3 \equiv_E 2$, while $7/3$ is not defined. The grammar rules that realize these partial functions are something like $N_2 ::= \dots \mid N_6 // N_3 \mid N_7 // N_3 \dots \mid N_6 / N_3$. The integer remainder is denoted by $(\%)$. We embedded the two-element Boolean algebra $\{0, 1\}$ into the natural numbers, with 1 corresponding to *true*. This allows us to model relations like $(<)$ and logical connectives. The functions (\uparrow) and (\downarrow) compute the maximum and minimum of two numbers, respectively. The function dp doubles a natural number in 0-*s* representation, ap concatenates two lists in *cons-nil* representation, rv reverses a list, and ln computes its length as a natural number. The *cube* theory formalizes the six possible three-dimensional 90°-degree rotations of a cube, viz. *left*, *right*, *up*, *down*, *clockwise* and *counter-clockwise*, as shown in Fig. 8 (right).

The column *Lemma* shows the corresponding lemma, its right-hand side having been supplied, its left-hand side generated by the above method. Note, for example, the differ-

Theory	Lemma	Rhs	No	Time
$+, *$	$(x + y) + z = x + (y + z)$	1,1,3	6.	21
$+, *$	$x * (y + z) = x * y + x * z$	0,2,2	10.	17
$+, *$	$x * y = y * x$	0,0	3.	0
$+, *$	$(x * y) * z = x * (y * z)$	0,0,2	31.	7
$+, *$	$(x * y) * z = x * (y * z)$	0,0,2,4	3.	22
$+, -, *, /, //, \%$	$x/z + y/z = (x + y)/z$	5,1,3	2.	263324
$+, -, *, /, //, \%$	$((x \% z) + (y \% z)) \% z = (x + y) \% z$	0,1,1	1.	19206
$+, -, *, /, //, \%$	$(x // y) * y = x - (x \% y)$	6,0,3	1.	17304
$+, -, *, /, //, \%$	$x = (x * y) // y$	2,1,3	4.	17014
$+, *, <$	$x < y \Leftrightarrow x * z < y * z$	0,1,1	3.	174958
$+, *, <$	$x < y \Leftrightarrow x + z < y + z$	0,1,1	20.	174958
$+, *, <, \uparrow, \downarrow$	$x < y \Leftrightarrow x < x \uparrow y$	0,1,1	6.	47128
$+, *, <, \uparrow, \downarrow$	$x = x \downarrow (x \uparrow y)$	3,0,3	7.	45678
$+, *, <, \uparrow, \downarrow$	$(x \uparrow y) + (x \downarrow y) = x + y$	5,1,5	2.	42670
$+, *, dp$	$dp(x) + dp(y) = dp(x + y)$	2,4	2.	6
$+, *, dp$	$dp(x) = x + x$	0,4	4.	1
$+, *, dp$	$x * dp(y) = dp(x * y)$	0,0	13.	1
$\neg, \wedge, \vee, \rightarrow$	$\neg(x \wedge y) \Leftrightarrow y \rightarrow \neg x$	1,1,1,0	1.	308
$\neg, \wedge, \vee, \rightarrow$	$\neg(x \wedge y) \Leftrightarrow \neg x \vee \neg y$	1,1,1,0	6.	308
ap, rv	$ap(rv(x), rv(y)) = rv(ap(y, x))$	2,2,2	1.	89
ap, rv	$ap(x, ap(y, z)) = ap(ap(x, y), z)$	3,3,2	1.	296
ap, rv	$x = rv(rv(x))$	0,2	4.	1
ap, rv	$rv(rv(x)) = x$	0,2	1.	1
ap, rv, ln	$ln(ap(x, y)) = ln(x) + ln(y)$	1,2	4.	4
ap, rv, ln	$ln(cons(x, ap(y, z))) = s(ln(y) + ln(z))$	2,3	10.	21
$cube$	$lf(cc(x)) = up(lf(x))$	1,1	1.	18
$cube$	$lf(cc(x)) = cc(up(x))$	1,1	2.	18

Fig. 7. Generated lemma candidates.

ence between the lemmas $x = rv(rv(x))$ and $rv(rv(x)) = x$. The column *Rhs* indicates the size of $t_1\sigma_i$ for $i = 1, \dots, n$, which is a measure of the size of grammars to be intersected. For arithmetic and list theories, the value of each number $t_1\sigma_i$ and the length of each list $t_1\sigma_i$ is given, respectively. The column *No* shows the place in which the lemma's right-hand side appeared in the enumeration sequence, while the column *Time* shows the required runtime in milliseconds (compiled PROLOG on a 933 MHz PC). Both depend strongly on the number and size of the example ground instances. The dependence of *No* can be seen in lines 4 and 5.

The runtime also depends on the grammar connectivity. In a grammar that includes, e.g., $(-)$ or $(<)$, each nonterminal can be reached from any other, while in a grammar considering, e.g., only $(+)$ and $(*)$, only nonterminals for smaller values and N_i can be reached. If the grammar defines N_i, N_0, \dots, N_6 , computing $[0]_E^{\sigma_1} \cap [1]_E^{\sigma_2} \cap [1]_E^{\sigma_3}$ leads to $8 \cdot 8 \cdot 8$ intersection nonterminals in the former case, compared with $2 \cdot 3 \cdot 3$ in the latter. For

Given: series 0, 1, 4, 9, ..., and $k = 3$

Lgth	Suffix	Next
$p(s^3(0).s^4(0).s(0).0.nil)$,	$s^9(0)$
$p(s^2(0).s(0).0.nil)$,	$s^4(0)$
$p(s(0).0.nil)$,	$s(0)$
$p(s(v_p).v_1.v_2)$,	$s(v_p)*s(v_p)$

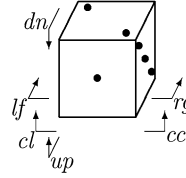


Fig. 8. Law computation by Theorem 21 (left). Cube rotations (right).

this reason, runtimes are essentially independent of the *Rhs* sizes for the 2nd to 4th theory. The exception in line 6 is due to a larger input grammar.

5.2. Construction laws of series

A second application of *E*-generalization consists in the computation of construction laws for term series, as in ordinary intelligence tests. The method is also based on Theorem 21 and is explained below.

For technical reasons, we write a series in reverse order as a *cons-nil* list, using an infix “.” for the reversed *cons* to enhance readability. We consider suffixes of this list and append a number denoting its length to each of them. We use a binary predicate $p(l.s, n)$ to denote that the suffix s of length l leads to n as the next series element. We apply Theorem 21 to the k last *leads to* relations obtained from the given series, see Fig. 8 (left), where k must be given by the user. Each result has the form $p(l.s, n)$ and corresponds to a rewrite rule $l.s \rightsquigarrow n$ that computes the next term from a series suffix and its length. By construction, the rewrite rule is guaranteed to compute at least the input terms correctly. A notion of correctness is not formally defined for later terms, anyway.

Fig. 9 shows some computed construction laws. Its first line exactly corresponds to the example in Fig. 8 (left). The column *Theory* indicates the equational theory used. The ternary function *if* realizes *if · then · else*, with the defining equations $if(s(x), y, z) = y$ and $if(0, y, z) = z$, and the unary function *ev* returns $s(0)$ for even and 0 for odd natural numbers. Using *if* and *ev*, two series can be interleaved (cf. line 5).

The column *Series* shows the given term series, $s^n(0)$ being abbreviated to n . The number k of suffixes supplied to our procedure corresponds to the number of series terms to the right of the semi-colon. Any computed hypothesis must explain all these series terms, but none of the earlier ones. The column *Law* shows the computed hypothesis. The place within the series is denoted by v_p , the first term having place 0, the second place 1, and so on. The previous series term and the last but one are denoted by v_1 and v_2 , respectively. The column *No* shows the place in which the law appeared in the enumeration sequence. In line 5, some formally smaller (wrt height) terms are enumerated before the term shown in Fig. 9, but are nevertheless equivalent to it. The column *Time* shows the required runtime in milliseconds on a 933 MHz machine, again strongly depending on k and the size of series terms.

The strength of our approach does not lie in its finding a plausible continuation of a given series, but rather in building, from a precisely limited set of operators, a nonrecursive











Theory	Series	Law	No	Time
$+, *$	0; 1, 4, 9	$v_p * v_p$	1.	2797
$+, *$	0; 2, 4, 6	$s(s(v_1))$	1.	3429
$+, *$	0; 2, 4, 6	$v_p + v_p$	3.	3429
$+, *$	1, 1; 2, 3, 5	$v_1 + v_2$	1.	857
$+, *, if, ev$	0, 1; 2, 1, 4, 1	$if(ev(v_p), v_p, 1)$	13.	13913
$+, *, if, ev$	0, 0, 1; 1, 0, 0, 1, 1	$ev(v_2)$	1.	61571
$+, *, if, ev$	0, 0; 1, 0, 0, 1	$ev(v_1 + v_2)$	1.	8573
$+, *, if, ev$	0; 1, 3, 7	$s(v_1 + v_1)$	1.	3714
$+, *, if, ev$	1, 2, 2, 3, 3, 3, 4; 4, 4, 4	—		8143
$cube, if, ev$	    	$rg(if(ev(v_p), v_1, \langle \begin{smallmatrix} \cdot \\ \cdot \end{smallmatrix} \rangle))$	1.	14713
$cube, if, ev$	    	$cl(if(ev(v_p), up(v_1), dn(v_1)))$	1.	604234

Fig. 9. Computed construction laws.

algorithm for computing the next series terms. Human superiority in the former area is demonstrated in line 9, where no construction law was found. The strength of the approach in the latter area became clear by the series 0, 0; 1, 0, 0, 1, shown in line 7. We had not expected any construction law to exist at all, because the series has a period relative prime to 2 and the trivial solution v_3 had been eliminated by the choice of k (a construction law must compute the first 1 from the preceding 0s).

It is decidable whether the result language H_{21} is finite; in such cases, we can make precise propositions about all construction laws that can be expressed using the given signature and equational theory. For example, from line 9 we can conclude that no construction law can be built from the given operators.

5.3. Generalizing screen editor commands

By way of another application, we employed E -generalization for learning complex cursor-movement commands of a screen-oriented editor like UNIX `vi`. For each $i, j \in \mathbb{N}$, let $p_{i,j}$ be a distinct constant denoting the position of a given file at column i and line j ; let $P = \{p_{i,j} \mid i, j \in \mathbb{N}\}$. For the sake of simplicity, we assume that the screen is large enough to display the entire contents of the file, so, we do not deal with scrolling commands for the present.

Assuming the file contents to be given, cursor-movement commands can be modeled as partial functions from P to itself. For example, $d(p_{i,j}) = p_{i,j+1}$ if $j + 1 \leq li$, undefined otherwise, models the *down* command, where li denotes the number of lines in the file. The constant $H = p_{1,1}$ models the *home* command. Commands may depend on the file contents. For example,

$$W(p_{i,j}) = \min\{i' \mid i < i' \leq co(j) \wedge ch(p_{i'-1,j}) \in SP \wedge ch(p_{i',j}) \notin SP\},$$

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
1	C	U	R	S	O	R																	
2	l																						
3	r																						
4	u																						
5	d																						

Fig. 10. Example file contents (left). Corresponding grammar excerpt (right).

if the minimum is defined, models the *next word* command, where $co(j)$, $ch(p)$, and SP denote the number of columns of line j , the character at position p , and the set of space characters, respectively. From a given file contents, it is easy to compute a regular tree grammar \mathcal{G} that describes the congruence classes of all its positions in time linear to the file size and the number of movement commands. Fig. 10 gives an example. For the sake of brevity, columns are “numbered” by lower-case letters, and, e.g., $\mathcal{L}(b2) = [p_{b,2}]_E$. Note that the file contents happen to explain some movement commands.

Using E -generalization, two or more cursor movements can easily be generalized to obtain a common scheme. Given the start and end positions, s_1, \dots, s_n and e_1, \dots, e_n , we apply Theorem 21 to $F^+ \Leftrightarrow p(s_1, e_1) \wedge \dots \wedge p(s_n, e_n)$ and get a rule of the form $p(x, t)$, where $t \in \mathcal{T}_{\{x\}}$ is a term describing a command sequence that achieves each of these movements.

For $n = 1$, we can compute the *simplest* term that transforms a given starting position into a given end position. This is useful to advise a novice user about advanced cursor-movement commands. Imagine, for example, that a user had typed the commands l, l, l, l, l, l, l, l to get from position $p_{k,2}$ to $p_{b,2}$. The term of least height obtained from Theorem 21, viz. $p(x, l(B(x)))$, indicates that the same movement could have been achieved by simply typing the commands B, l . Each command could also be assigned its own degree of simplicity, reflecting, for example, the number of modifier keys (like *shift*) involved, or distinguishing between simple and advanced commands. In the former case, the simplest term minimized the overall numbers of keys to be pressed.

No grammar intersection is needed if $n = 1$. Moreover, the lifting of \mathcal{G} can be done in constant time in this case. In the example, it is sufficient to include an alternative $\dots x \dots$ into the right-hand side of the rule for $k2$. Therefore, a simplest term can be computed in an overall time of $\mathcal{O}(\#\mathcal{G} \cdot \log \#\mathcal{G})$. Changes in the file content require recomputation of the grammar and the minimum term sizes. In many cases, but not if, for example, a parenthesis is changed, local content changes require local grammar changes only. It thus seems worthwhile to investigate an incremental approach, which should also cover weight recomputation.

For $n > 1$, the smallest term(s) in the result language may be used to implement an intelligent approach to repeat the last n movement command sequences. For example, the simplest scheme common to the movements $p(p_{m,2}, p_{o,2})$ and $p(p_{n,4}, p_{v,4})$ wrt the file content of Fig. 10 is computed as $p(x, d(W(u(x))))$. Since the computation time grows exponentially with n , it should be small.

In our prototypical implementation, we considered in all the `vi` commands `h`, `j`, `k`, `l`, `H`, `M`, `L`, `+`, `-`, `w`, `b`, `e`, `W`, `B`, `E`, `0`, `$`, `f`, `F`, `%`, `{`, and `}` and renamed some of them to give them more suggestive identifiers. We allow search for single characters only. In order to consider nontrivial string search commands as well, the above approach should be combined with (string) grammar inference [23,34] to learn regular search expressions. Moreover, commands that change the file content should be included in the learning mechanism. And last but not least, a satisfactory user interface for these learning features is desirable, e.g., allowing us to define command macros from examples.

5.4. Prototypical implementation

We built a prototypical implementation realizing the E -generalization method from Section 3.1 and the applications from Section 5. It comprises about 4,000 lines of PROLOG code. Fig. 11 shows its architecture, an arrow meaning that its source function uses its destination function. The application module allows us to learn series laws, candidate lemmas, and editor cursor commands (`edt cmds`). The anti-unification module contains algorithms for syntactic (`synt au`), constrained (`cs e-au`) and unconstrained (`uc e-au`) E -anti-unification. The grammar-generation module can compute grammars for a given file content (`edt grm`), for any set $\{t \in \mathcal{T} \mid V \subseteq \text{var}(t) \subseteq W\}$ (`var grm`), and for the set of normal forms wrt E (`nf grm`). The grammar algorithms module allows us to test an $\mathcal{L}(N)$ for finiteness, emptiness, and a given member t , to compute intersection and complement of two languages, to simplify a grammar, and to generate \mathcal{N}_{\max} from Lemma 5 (`max nt s`) and a grammar for $\mathcal{T}_{\{\}}^{\{\}}$ (`top nt`). For the sake of clarity, we omitted the dashed lines around the pre- and postprocessing module. The former merely contains code to choose `exm` instances for lemma generation. The latter does term evaluation to normal form, and enumeration and minimal weight computation for $\mathcal{L}(N)$. The prototype still uses monolithic, specially tailored algorithms for E -anti-unification, as originally given in [22], rather than the combination of standard grammar algorithms described in Section 3.1. For this reason, function `intersect` uses `cs e-au` as a special case, viz. $\sigma_i = \{\}$, rather than vice versa. However, all

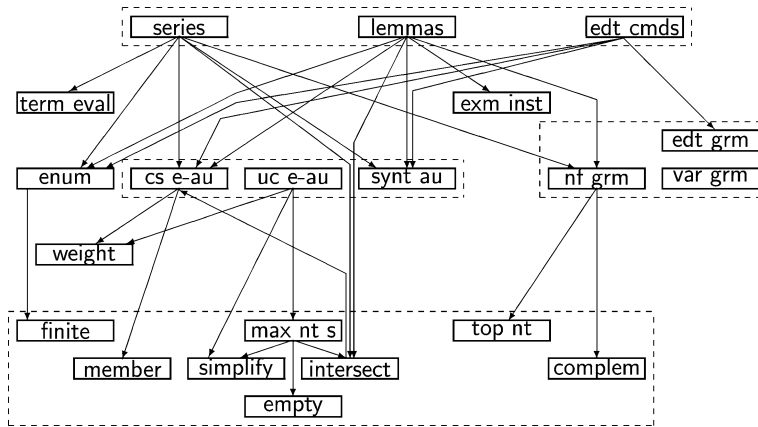


Fig. 11. Prototype architecture.

other *uses* relations would remain unchanged in an implementation strictly based on this paper.

All runtime figures given in this paper are taken from the prototype. Currently, an efficiency-oriented re-implementation in C is planned. Moreover, it will use the available memory more efficiently, thus allowing us to run larger examples than when using PROLOG. The PROLOG prototypical implementation and, in future, the C implementation can be downloaded from the web page <http://swt.cs.tu-berlin.de/~jochen/e-au>.

6. Conclusions and future work

We presented a method for computing a finite representation of the set of *E*-generalizations of given terms and showed some applications. *E*-generalization is able to cope with representation change in abstraction, making it a promising approach to an old but not yet satisfactorily solved problem of Artificial Intelligence. Our approach is based on standard algorithms for regular tree grammars. It thus allows us to add filtering components in a modular fashion, as needed by the surrounding application software. The closed form of an *E*-generalization set as a grammar and its simple mathematical characterization make it easy to prove formal quality properties if needed for an application. Using a standard grammar language enumeration algorithm, the closed form can be converted to a succession form.

Our method cannot handle every equational theory *E*. To use the analogy with *E*-deduction, our method corresponds to something between *E*-unification (concerned with a particular *E* in each case) and paramodulation (concerned with the class of canonical *E*). On the other hand, neither partial functions nor conditional equations basically prevent our method from being applicable.

In order to demonstrate that *E*-generalization can be integrated into ILP learning methods, we proved several ways of combining *lgg*-based ILP and *E*-generalization. Predicate definitions by atoms or clauses can be learned. If desired, the hypotheses space can be restricted to determinate hypotheses, resulting in faster algorithms. Learning of purely determinate clauses can be reduced to learning of atoms by *E*-generalization. An *lgg*-learner for constrained clauses with built-in *E*-generalization can learn a proper superclass, called semi-determinate predicates here. We provide completeness properties for all our hypotheses sets.

Using *E*-generalization, the search space is split into two parts, one concerned with selection of nondeterminate literals, the other with selection of their argument terms. While the first part is best handled by an elaborate strategy from *classical* ILP, the second can be left to a grammar language enumeration strategy. For example, the $\mathcal{O}(\#G \cdot \log \#G)$ algorithm to find a term of minimal complexity within a tree language apparently has no corresponding selection algorithm for determinate literals in classical ILP. Separating both search space parts allows us to modularize the strategy algorithms and to use for each part one that best fits the needs of the surrounding application.

Experiments with our prototypical implementation showed that comparatively ambitious AI tasks are solvable at the first attempt using *E*-generalization. We focus on sketching applications in a number of different areas rather than on perfectly elaborating a single

application. By doing so, we seek to demonstrate the flexibility of E -generalization, which is a necessary feature for any approach to be related to intelligence. In [2, Section 8], further applications were sketched, including divergence handling in Knuth–Bendix completion, guessing of Hoare invariants, reengineering of functional programs, and strengthening of induction hypotheses. The method given in [5] to compute a finite representation of the complete equational theory describing a given set of finite algebras is essentially based on E -generalization, too. It is shown there that the complete theory can be used to implement fast special-purpose theorem provers for particular theories.

Based on this experience, we venture to suggest that E -generalization is able to simulate an important aspect of human intelligence, and that it is worth investigating further. In particular, the restrictions regular tree grammars impose on the background equational theory E should be relaxed. In this paper, we briefly looked at some well-known representation formalisms that are more expressive than regular tree grammars but with negative results. It remains to be seen whether there are other more expressive formalisms that can be used for E -generalization. The attempt should also be made to combine it with higher-order anti-unification [21,40]. Such a combination is expected to allow recursive functions to be learned from examples.

As indicated above, the applications of E -generalization could certainly be improved. Lemma generation should be integrated into a real induction prover, in particular to test its behavior in combination with the *rippling* method [7]. While rippling suggests checking homomorphic laws like $f(g(t_1), \dots, g(t_n)) =_E g'(f(t_1), \dots, t_n))$ for validity, E -generalization is able to suggest lemmas of arbitrary forms. Empirical studies on series-based intelligence tests, e.g., using geometrical theories about *mirror*, *shift*, *rotate*, etc., should look for a saturation effect: Is there one single reasonable equational background theory that can solve a sufficiently large number of common tests? And can a reasonable *intelligence quotient* be achieved by that theory? Currently, we are investigating the use of E -generalization in analogical reasoning [12], a new application that does not fit into the schemas described in Section 4. The aim is to allow problems in intelligence tests to be stated in other ways than mere linear series, e.g., to solve $(A : B) = (C : X)$, where A, B, C are given terms and X is a term which should result from applying a rule to C that at the same time transforms A into B .

Acknowledgements

Ute Schmid, Holger Schlingloff and Ulrich Geske provided valuable advice on presentation.

References

- [1] F. Baader, Unification, weak unification, upper bound, lower bound, and generalization problems, in: Proc. 4th Conf. on Rewriting Techniques and Applications, in: Lecture Notes in Comput. Sci., vol. 488, Springer, Berlin, 1991, pp. 86–91.
- [2] J. Burghardt, B. Heinz, Implementing anti-unification modulo equational theory, Arbeitspapier 1006, GMD, June 1996.

- [3] R.S. Boyer, J.S. Moore, *A Computational Logic*, Academic Press, New York, 1979.
- [4] B. Bogaert, S. Tison, Equality and disequality constraints on direct subterms in tree automata, in: Proc. STACS 9, in: *Lecture Notes in Comput. Sci.*, vol. 577, Springer, Berlin, 1992, pp. 161–172.
- [5] J. Burghardt, Axiomatization of finite algebras, in: Proc. KI-2002, in: *Lecture Notes in Artificial Intelligence*, vol. 2479, Springer, Berlin, 2002, pp. 222–234.
- [6] J. Burghardt, Weight computation of regular tree languages, Technical Report 001, FIRST, December 2003.
- [7] A. Bundy, F. van Harmelen, A. Smaill, A. Ireland, Extensions to the rippling-out tactic for guiding inductive proofs, in: Proc. 10th CADE, in: *Lecture Notes in Artificial Intelligence*, vol. 449, Springer, Berlin, 1990, pp. 132–146.
- [8] A.-C. Caron, H. Comon, J.-L. Coquidé, M. Dauchet, F. Jacquemard, Pumping, cleaning and symbolic constraints solving, in: Proc. ICALP, in: *Lecture Notes in Comput. Sci.*, vol. 820, 1994, pp. 436–449.
- [9] A.-C. Caron, J.-L. Coquidé, M. Dauchet, Automata for reduction properties solving, *J. Symbolic Comput.* 20 (2) (1995) 215–233.
- [10] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications. Available from <http://www.grappa.univ-lille3.fr/tata>, October 2001.
- [11] K.L. Clark, Predicate logic as a computational formalism, Research Report, Imperial College, 1979.
- [12] M. Dastani, B. Indurkha, R. Scha, An Algebraic Method for Solving Proportional Analogy Problems, Dublin City University, Dublin, 1997.
- [13] N. Dershowitz, J.-P. Jouannaud, Rewrite systems, in: *Handbook of Theoretical Computer Science*, vol. B, Elsevier, Amsterdam, 1990, pp. 243–320.
- [14] T.G. Dietterich, R.S. Michalski, A Comparative Review of Selected Methods for Learning from Examples, Springer, Berlin, 1984, pp. 41–82.
- [15] P.J. Downey, R. Sethi, R.E. Tarjan, Variations on the common subexpression problem, *J. ACM* 27 (4) (1980) 758–771.
- [16] S. Džeroski, *Inductive Logic Programming and Knowledge Discovery in Databases*, MIT Press, Cambridge, MA, 1996, pp. 117–152.
- [17] M. Fay, First-order unification in an equational theory, in: Proc. 4th Workshop on Automated Deduction, 1979.
- [18] G. Gentzen, Untersuchungen über das logische Schließen, 1932.
- [19] E.M. Gold, Language identification in the limit, *Inform. and Control* 10 (1967) 447–474.
- [20] J.H. Gallier, W. Snyder, Complete sets of transformations for general *E*-unification, *Theoret. Comput. Sci.* 67 (1989) 203–260.
- [21] R.W. Hasker, The replay of program derivations, PhD thesis, Univ. of Illinois at Urbana-Champaign, 1995.
- [22] B. Heinz, Anti-Unifikation modulo Gleichungstheorie und deren Anwendung zur Lemmagenerierung, Doctoral dissertation, TU Berlin, December 1995.
- [23] V. Honavar, R. Parekh, Grammar Inference, Automata Induction, and Language Acquisition, Marcel Dekker, New York, 1999.
- [24] R. Kowalski, Predicate logic as programming language, Memo 70, Dept. of Comp. Logic, School of Artif. Intell., Univ. Edinburgh, 1973.
- [25] N. Lavrac, S. Džeroski, *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, New York, 1994.
- [26] D. McAllester, Grammar rewriting, in: Proc. CADE-11, in: *Lecture Notes in Artificial Intelligence*, vol. 607, Springer, Berlin, 1992.
- [27] S. Muggleton, C. Feng, Efficient induction of logic programs, in: Proc. 1st Conf. on Algorithmic Learning Theory, Tokyo, Omsha, 1990, pp. 368–381.
- [28] S. Muggleton, Inductive logic programming: Issues, results and the challenge of learning language in logic, *Artificial Intelligence* 114 (1999) 283–296.
- [29] S. O'Hara, A model of the redescription process in the context of geometric proportional analogy problems, in: Proc. AII '92, Dagstuhl, Germany, in: *Lecture Notes in Artificial Intelligence*, vol. 642, Springer, Berlin, 1992, pp. 268–293.
- [30] G.D. Plotkin, A note on inductive generalization, in: B. Meltzer, D. Michie (Eds.), *Machine Intelligence*, vol. 5, Edinburgh Univ. Press, 1970, pp. 153–163.
- [31] G.D. Plotkin, A further note on inductive generalization, in: B. Meltzer, D. Michie (Eds.), *Machine Intelligence*, vol. 6, Edinburgh Univ. Press, 1971, pp. 101–124.

- [32] L. Pottier, Generalisation de termes en theorie equationelle, Cas associatif-commutatif, Report 1056, INRIA, 1989.
- [33] J.C. Reynolds, Transformational systems and the algebraic structure of atomic formulas, in: B. Meltzer, D. Michie (Eds.), *Machine Intelligence*, vol. 5, Edinburgh Univ. Press, 1970, pp. 135–151.
- [34] Y. Sakakibara, Recent advances of grammatical inference, *Theoret. Comput. Sci.* 185 (1997) 15–45.
- [35] U. Schöning, *Theoretische Informatik—kurzgefaßt*, Spektrum-Hochschultaschenbuch, Heidelberg, 1997.
- [36] J.H. Siekmann, *Universal Unification*, Univ. Kaiserslautern, 1985.
- [37] J.W. Thatcher, J.B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, *Math. Syst. Theory* 2 (1) (1968).
- [38] T.E. Uribe, Sorted unification using set constraints, in: *Proc. CADE-11*, in: *Lecture Notes in Comput. Sci.*, vol. 607, 1992, pp. 163–177.
- [39] L.G. Valiant, A theory of the learnable, *Comm. ACM* 27 (1984) 1134–1142.
- [40] U. Wagner, Combinatorically restricted higher order anti-unification. Master's thesis, TU Berlin, April 2002.