Anti-Pattern Matching Modulo

Claude Kirchner, Radu Kopetz, Pierre-Etienne Moreau

▶ **To cite this version:**

Claude Kirchner, Radu Kopetz, Pierre-Etienne Moreau. Anti-Pattern Matching Modulo. The Fifth ASIAN Symposium on Programming Languages and Systems - APLAS 2007, Nov 2007, Singapore. inria-00129421v1

HAL Id: inria-00129421

https://inria.hal.science/inria-00129421v1

Submitted on 7 Feb 2007 (v1), last revised 30 Oct 2007 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Anti-Pattern Matching Modulo

Claude Kirchner, Radu Kopetz, Pierre-Etienne Moreau

INRIA & LORIA*, Nancy, France
{Claude.Kirchner, Radu.Kopetz, Pierre-Etienne.Moreau}@loria.fr

Abstract. Negation is intrinsic to human thinking and most of the time when searching for something, we base our patterns on both positive and negative conditions. In a previous work, we have extended the notion of term to the one of anti-term that may contain complement symbols. Matching such anti-terms against terms has the nice property of being unitary.
Here we generalize the syntactic anti-pattern matching to anti-pattern matching modulo an arbitrary equational theory $\mathcal{E}$, and we study the specific and practically very useful case of associativity, possibly with a unity. To this end, based on the syntacticness of associativity, we present a rule-based associative matching algorithm, and we extend it to $\mathcal{AU}$. This algorithm is then used to solve $\mathcal{AU}$ anti-pattern matching problems. This allows us to be generic enough so that for instance, the *AllDiff* standard predicate of constraint programming becomes simply expressible in this framework. $\mathcal{AU}$ anti-patterns are implemented in the Tom language and we show some examples of their usage.

1 Introduction

When searching for something, we usually base our searches on both positive and negative conditions. Indeed, when stating "except a red car", it means that whatever the other characteristics are, a red car will not be accepted. This is a very common way of thinking, more natural than a series of disjunctions like "a white car or a blue one or a black one or ...". But if this is so natural, why are complements in their full generality not supported by pattern-matching search engines?

In [16] we introduced the notion of *anti-patterns* consisting in terms that may contain complement symbols, with no restriction on the complement nesting or on linearity. Typically, imagine that we use a route-planner: expressing that we search an itinerary that do not pass through *Paris* corresponds naturally to the anti-pattern: $\neg itinerary(Paris, \_)$. Nesting complements eases expression of needs: $\neg itinerary(Paris, \neg fastest)$ expresses that we want an itinerary that does not pass through *Paris*, except if it is the fastest one. Using disjunctions, this anti-pattern corresponds to: $\neg itinerary(Paris, \_) \vee itinerary(\_, fastest)$. Non-linearity can also be very practical for searching objects that do not have

* UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP

similar sub-characteristics. For example, we may ask for an itinerary from *Nancy* to *Paris* that doesn't contain two rest-places handled by the same food sign.

Although the anti-patterns provide a compact and expressive representation for sets of objects in the empty theory, when associating them with other theories they are even more flexible and powerful. For instance, consider the list matching as provided by the Tom language — a programming language that extends C and Java with algebraic data-types, pattern matching and strategic rewriting facilities [20,17]. The pattern $(*, \neg a, *)$ denotes a list which contains at least one element different from the constant a, whereas $\neg(*, a, *)$ denotes a list which does not contain any a. By using non-linearity we can express, in a single pattern, known list constraints as *AllDiff* or *AllEqual*. Take for instance the pattern $(*, x, *, x, *)$ that denotes a list with at least two equal elements. The complement of this, $\neg(*, x, *, x, *)$ matches lists that have only distinct elements, *i.e. AllDiff*. In a similar way, as $(*, x, *, \neg x, *)$ matches the lists that have at least two distinct elements, its complement $\neg(*, x, *, \neg x, *)$ denotes any list whose elements are all equal.

This is more generally useful for arbitrary equational theories — like associativity, associativity with neutral elements or commutativity for example. Therefore, our first contribution, in Section 3, is to solve associative matching problems using a rule-based algorithm directly induced from the syntacticness property of associativity. We prove its correctness and completeness and show how this algorithm can be adapted to also support neutral elements.

Our second main contribution is to generalize, in Section 4, the notion of anti-patterns to an arbitrary equational theory. We show how an equational anti-pattern matching problem can be transformed into a finite subset of equivalent equational problems. Further on, we focus on the associative anti-patterns with neutral elements and we present an algorithm for solving such problems, along with its correctness proof. The anti-patterns provide a compact and expressive formalism for pattern matching languages and we show in the Section 5 how they are integrated and used in the Tom language.

Although we will make precise our main notations, we assume that the reader is familiar with the standard notions of algebraic rewrite systems, for example presented in [1] and rule-based unification algorithms, see *e.g.* [12].

2 Algebraic terms and anti-patterns

Terms and equality. A signature $\mathcal{F}$ is a set of function symbols, each one having a fixed arity associated. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the set of *terms* built from a given finite set $\mathcal{F}$ of function symbols where constants are denoted $a, b, c, \ldots$, and a denumerable set $\mathcal{X}$ of variables denoted $x, y, z, \ldots$ A term t is said to be *linear* if no variable occurs more than once in t. The set of variables occurring in a term t is denoted by $\mathcal{V}ar(t)$. If $\mathcal{V}ar(t)$ is empty, t is called a *ground term* and $\mathcal{T}(\mathcal{F})$ is the set of ground terms.

A *substitution* σ is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$, denoted $\sigma = \{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$ when its domain $\mathsf{Dom}(\sigma)$ is finite. Its application, written $\sigma(t)$,

is defined by $\sigma(x_i) = t_i$, $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ for $f \in \mathcal{F}$, and $\sigma(y) = y$ if $y \notin \mathsf{Dom}(\sigma)$. Given a term t, σ is called a *grounding*[1] *substitution* when $\sigma(t) \in \mathcal{T}(\mathcal{F})$. The set of substitutions is denoted Σ. The set of grounding substitutions for a term t is denoted $\mathcal{GS}(t)$. Usually σ, ρ, θ denote substitutions.

The ground semantics of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is the set of all its ground instances: $[\![t]\!]_g = \{\sigma(t) \mid \sigma \in \mathcal{GS}(t)\}$. In particular, $[\![x]\!]_g = \mathcal{T}(\mathcal{F})$.

A *position* in a term is a finite sequence of natural numbers. The subterm u of a term t at position ω is denoted $t_{|\omega}$, where ω describes the path from the root of t to the root of u. $t(\omega)$ denotes the root symbol of $t_{|\omega}$. By $t[u]_\omega$ we express that the term t contains u as subterm at position ω. Positions are ordered in the classical way: $\omega_1 < \omega_2$ if ω_1 is the prefix of ω_2.

For an equational theory $\mathcal{E}$, an *$\mathcal{E}$-matching equation* (*matching equation* for short) is of the form $p \ll^?_{\mathcal{E}} t$ where p is a term classically called a pattern and t is a term, generally considered as ground. The substitution σ is a *$\mathcal{E}$-solution of the $\mathcal{E}$-matching equation* $p \ll^?_{\mathcal{E}} t$ if $\sigma(p) =_{\mathcal{E}} t$, and it is called an *$\mathcal{E}$-match* from p to t.

An *$\mathcal{E}$-matching system* S is a possibly existentially quantified conjunction of matching equations: $\exists \bar{x}(\wedge_i p_i \ll^?_{\mathcal{E}} t_i)$. A substitution σ is an *$\mathcal{E}$-solution* of such a matching system if there exists a substitution ρ, with domain $\bar{x}$, such that σ is solution of all the matching equations $\rho(p_i) \ll^?_{\mathcal{E}} \rho(t_i)$. The set of solutions of S is denoted by $\mathcal{S}ol_{\mathcal{E}}(\mathsf{S})$.

An *$\mathcal{E}$-matching disjunction* D is a disjunction of $\mathcal{E}$-matching systems. Its solutions are the substitutions solution of at least one of its system constituents. Its free variables $\mathcal{FV}ar(\mathsf{D})$ are defined as usual in predicate logic. We use the notation $\mathsf{D}[\mathsf{S}]$ to denote that the system S occurs in the *context* D.

A binary operator f is called *associative* if it satisfies the equational axiom $\forall x, y, z \in \mathcal{T}(\mathcal{F}, \mathcal{X}) : f(f(x, y), z) = f(x, f(y, z))$ and *commutative* if $\forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) : f(x, y) = f(y, x)$. A binary operator can have neutral elements — symbols of arity zero:

- e_f is a *left neutral* operator for f if $\forall x \in \mathcal{T}(\mathcal{F}, \mathcal{X}),\ f(e_f, x) = x$.
- e_f is a *right neutral* operator for f if $\forall x \in \mathcal{T}(\mathcal{F}, \mathcal{X}),\ f(x, e_f) = x$.
- e_f is a *neutral* or *unit* operator for f if it is a left and right neutral operator for f.

When f is associative or associative with a unit, this is denoted $\mathcal{A}$ or $\mathcal{AU}$.

Anti-terms. An anti-term [16] is a term that may contain complement symbols, denoted by $\neg$. The BNF of anti-terms is:

$\mathcal{AT} ::= \mathcal{X} \mid f(\mathcal{AT}, \ldots, \mathcal{AT}) \mid \neg\mathcal{AT}$, where f respects its arity.

The set of anti-terms (resp. ground anti-terms) is denoted $\mathcal{AT}(\mathcal{F}, \mathcal{X})$ (resp. $\mathcal{AT}(\mathcal{F})$). Any term is an anti-term, *i.e.* $\mathcal{T}(\mathcal{F}, \mathcal{X}) \subset \mathcal{AT}(\mathcal{F}, \mathcal{X})$.

The $\neg$ operator behaves as a binder for all the variables that occur beneath it. We denote the free variables of an anti-term $\mathcal{FV}ar(t)$. Typically we have, for all t, $\mathcal{FV}ar(\neg t) = \emptyset$ and $\mathcal{FV}ar(f(x, \neg x)) = \{x\}$.

[1] in general different from a *ground* substitution.

The substitutions are only active on free variables. For anti-terms, a grounding substitution is a substitution that instantiates all the free variables by ground terms.

As detailed in [16], the *ground semantics* of any anti-term $q \in \mathcal{AT}(\mathcal{F}, \mathcal{X})$ is defined recursively in the following way: $[\![q[\neg q']_\omega]\!]_g = [\![q[z]_\omega]\!]_g \backslash [\![q[q']_\omega]\!]_g$, where z is a fresh variable and for all $\omega' < \omega$, $q(\omega') \neq \neg$.

Example 2.1.

1. $[\![f(a, \neg b)]\!]_g = [\![f(a,z)]\!]_g \backslash [\![f(a,b)]\!]_g = \{f(a, \sigma(z)) \mid \sigma \in \mathcal{GS}(f(a,z))\} \backslash \{f(a,b)\}$,
2. We can express that we are looking for something that is either not rooted by g, or it is $g(a)$:
$$\begin{aligned}[\![\neg g(\neg a)]\!]_g &= [\![z]\!]_g \backslash [\![g(\neg a)]\!]_g = [\![z]\!]_g \backslash ([\![g(z')]\!]_g \backslash [\![g(a)]\!]_g) \\ &= \mathcal{T}(\mathcal{F}) \backslash ([\![g(z')]\!]_g \backslash \{g(a)\}) \\ &= \mathcal{T}(\mathcal{F}) \backslash (\{g(\sigma(z')) \mid \sigma \in \mathcal{GS}(g(z'))\} \backslash \{g(a)\}) \\ &= \mathcal{T}(\mathcal{F}) \backslash \{g(z) \mid z \in \mathcal{T}(\mathcal{F}, \mathcal{X})\} \ \cup \ \{g(a)\},\end{aligned}$$
3. Non-linearity is crucial to denote any term except those rooted by f with identical subterms:
$[\![\neg f(x,x)]\!]_g = [\![z]\!]_g \backslash [\![f(x,x)]\!]_g = \mathcal{T}(\mathcal{F}) \backslash \{f(\sigma(x), \sigma(x)) \mid \sigma \in \mathcal{GS}(f(x,x))\}$.

The anti-terms are also called anti-patterns, in particular when they appear in the left hand side of a match equation. The notions of matching equations, systems and disjunctions are extended to anti-patterns by allowing the *left-hand sides* of match equations to be anti-patterns. When a match equation contains anti-patterns, we often refer to it as an *anti-pattern matching equation*. Solutions of such problems are defined later.

3 Associative matching

To provide an equational anti-matching algorithm in the next section, we first need to make precise the matching algorithm that serves us as a starting point.

As opposed to syntactic matching, matching modulo an equational theory is undecidable as well as not unitary in general [3]. When decidable, matching problems can be quite expensive either to decide matchability or to enumerate complete sets of matchers. For instance, matchability is NP-complete for associativity $\mathcal{A}$ with idempotency $\mathcal{I}$ or neutral elements $\mathcal{U}$ [2,13]. Also, counting the number of minimal complete set of matches modulo $\mathcal{A}$ or $\mathcal{AU}$ is #P-complete [10].

In this section we focus on the particular useful case of matching modulo associativity possibly with a unit. This is sometimes called list matching, as a restricted equivalence between the two concepts can be shown [24]. The reason why we chose to detail this specific theory is its tremendous usefulness in rule-based programming, where lists, and consequently list-matching, are omnipresent. Since associativity and neutral element are regular axioms (*i.e.* equivalent terms have the same set of variables), we can apply the combination results for matching modulo the union of disjoint regular equational theories [22,25] to

get a matching algorithm modulo the theory combination of an arbitrary number of associative, associative and unit as well as free symbols. Therefore we study in this section matching modulo the associativity or associativity with unit of a single binary symbol f, whose unit is denoted e_f. The only other symbols under consideration are free constants. For syntactic matching, a simple rule-based matching algorithm can be found in [6,16].

3.1 Matching associative patterns

By making precise this algorithm, our purpose is to provide a simple and intuitive one that can be easily proved to be correct and complete and that we will later adapt to anti-pattern matching. If the reader is looking into writing an efficient associative matcher, he can always refer to more appropriate approaches like [8,9].

Unification modulo associativity has been extensively studied [23,18]. Our matching algorithm $\mathcal{A}$-Matching is described in Figure 1 and is quite reminiscent from [21] although not based on a Prolog resolution strategy. It strongly relies on the syntacticness of the associative theory [14,15]. Other former works, in particular related to the ASF and to MAUDE environments are described in [8,9].

We assume the symbols $\wedge, \vee$ to be associative, commutative and idempotent, S is any conjunction of matching equations, p_i are patterns, and t_i are ground terms. The most interesting rule is Mutate, which is a direct consequence of the fact that associativity is a *syntactic theory*.

Mutate	$f(p_1,p_2) \ll_{\mathcal{A}} f(t_1,t_2)$	$\mapsto (p_1 \ll_{\mathcal{A}} t_1 \wedge p_2 \ll_{\mathcal{A}} t_2) \vee$ $\exists x(p_2 \ll_{\mathcal{A}} f(x,t_2) \wedge f(p_1,x) \ll_{\mathcal{A}} t_1) \vee$ $\exists x(p_1 \ll_{\mathcal{A}} f(t_1,x) \wedge f(x,p_2) \ll_{\mathcal{A}} t_2)$
$\mathsf{SymbolClash}_1$	$f(p_1,p_2) \ll_{\mathcal{A}} a$	$\mapsto \bot$
$\mathsf{SymbolClash}_2$	$a \ll_{\mathcal{A}} f(p_1,p_2)$	$\mapsto \bot$
ConstantClash	$a \ll_{\mathcal{A}} b$	$\mapsto \bot$ if $a \neq b$
Replacement	$z \ll_{\mathcal{A}} t \wedge \mathsf{S}$	$\mapsto z \ll_{\mathcal{A}} t \wedge \{z \to t\}\mathsf{S}$ if $z \in \mathcal{FV}ar(\mathsf{S})$
Delete	$p \ll_{\mathcal{A}} p$	$\mapsto \top$
Utility Rules:		
Exists_1	$\exists z(\mathsf{D}[z \ll_{\mathcal{A}} t])$	$\mapsto \mathsf{D}[\top]$ if $z \notin \mathcal{FV}ar(\mathsf{D}[\top])$
Exists_2	$\exists z(\mathsf{S}_1 \vee \mathsf{S}_2)$	$\mapsto \exists z(\mathsf{S}_1) \vee \exists z(\mathsf{S}_2)$
DistributeAnd	$\mathsf{S}_1 \wedge (\mathsf{S}_2 \vee \mathsf{S}_3)$	$\mapsto (\mathsf{S}_1 \wedge \mathsf{S}_2) \vee (\mathsf{S}_1 \wedge \mathsf{S}_3)$
$\mathsf{PropagClash}_1$	$\mathsf{S} \wedge \bot$	$\mapsto \bot$
$\mathsf{PropagClash}_2$	$\mathsf{S} \vee \bot$	$\mapsto \mathsf{S}$
$\mathsf{PropagSuccess}_1$	$\mathsf{S} \wedge \top$	$\mapsto \mathsf{S}$
$\mathsf{PropagSuccess}_2$	$\mathsf{S} \vee \top$	$\mapsto \top$

Fig. 1. $\mathcal{A}$-Matching

Proposition 3.1. *Given a matching equation $p \ll_{\mathcal{A}} t$ with $p \in \mathcal{T}(\mathcal{F}, \mathcal{X}), t \in \mathcal{T}(\mathcal{F})$, the application of $\mathcal{A}$-*Matching *always terminates.*

Proof. The proof is in the Appendix A. □

If no solution is lost in the application of a transformation rule, the rule is called *preserving*. It is a *sound* rule if it does not introduce unexpected solutions. A rewrite system is preserving (or sound) if all the rules it contains have this property.

Proposition 3.2. *The rules in $\mathcal{A}$-*Matching *are sound and preserving modulo $\mathcal{A}$.*

Proof. The rule Mutate is a direct consequence of the decomposition rules for syntactic theories presented in [15]. The rest of the rules are usual ones for which these results have been obtained for examples in [6]. □

Theorem 3.1. *Let us consider a normal form by the rules in $\mathcal{A}$-*Matching *of a matching equation $p \ll_{\mathcal{A}} t$ with $p \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $t \in \mathcal{T}(\mathcal{F})$.*

1. *If it is a disjunction of conjunctions like $\bigwedge_{i \in I} x_i \ll_{\mathcal{A}} t_i$ with $I \neq \emptyset$, then the substitutions $\sigma = \{x_i \mapsto t_i\}_{i \in I}$ are all the matches from p to t;*
2. *If it is $\top$ then p and t are identical modulo $\mathcal{A}$, i.e. $p =_{\mathcal{A}} t$;*
3. *If it is $\bot$ then there is no match from p to t;*

Proof. Thanks to Proposition 3.1 and Proposition 3.2, a normal form for a matching equation $p \ll_{\mathcal{A}} t$ *w.r.t.* $\mathcal{A}$-Matching always exists. Therefore, we have to prove that (i) all the quantifiers are eliminated and (ii) all match-equation's left-hand sides are variables of the initial equation. We only have existential quantifiers, introduced by Mutate, which are distributed to each conjunction by Exists_2 and later eliminated by the rule Exists_1. The validity of the condition of this latter rule is ensured by the rule Replacement, which leaves only one occurrence of each variable in a conjunction. On the other hand, we never eliminate free variables in a conjunction (only some duplicates), which justifies (ii). Finally, all normal forms are necessarily of the the form (1), (2) or (3), otherwise a rule could be further applied. □

Example 3.1. Applying $\mathcal{A}$-Matching for $f \in \mathcal{F}_{\mathcal{A}}$, $x, y \in \mathcal{X}$, $a, b, c, d \in \mathcal{T}(\mathcal{F})$:
$f(x, f(a, y)) \ll_{\mathcal{A}} f(f(b, f(a, c)), d)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{Mutate}} (x \ll_{\mathcal{A}} f(b, f(a, c)) \wedge f(a, y) \ll_{\mathcal{A}} d) \vee$
$\exists z (f(a, y) \ll_{\mathcal{A}} f(z, d) \wedge f(x, z) \ll_{\mathcal{A}} f(b, f(a, c))) \vee$
$\exists z (x = f(f(b, f(a, c)), z) \wedge f(z, f(a, y)) = d)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{SymbolClash}_1, \mathsf{PropagClash}_2} \exists z (f(a, y) \ll_{\mathcal{A}} f(z, d) \wedge f(x, z) \ll_{\mathcal{A}} f(b, f(a, c)))$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{Mutate}, \mathsf{SymbolClash}} \exists z (f(a, y) \ll_{\mathcal{A}} f(z, d) \wedge$
$((x \ll_{\mathcal{A}} b \wedge z \ll_{\mathcal{A}} f(a, c)) \vee (x \ll_{\mathcal{A}} f(b, a) \wedge z \ll_{\mathcal{A}} c)))$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{DistributeAnd}, \mathsf{Replacement}, \mathsf{Mutate}, \mathsf{SymbolClash}, \mathsf{Propag}}$
$\exists z (f(a, y) \ll_{\mathcal{A}} f(z, d) \wedge x \ll_{\mathcal{A}} b \wedge z \ll_{\mathcal{A}} f(a, c))$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{Replacement}, \mathsf{Exists}, \mathsf{Mutate}, \mathsf{SymbolClash}, \mathsf{Propag}} x \ll_{\mathcal{A}} b \wedge y \ll_{\mathcal{A}} f(c, d).$

3.2 Matching associative patterns with unit elements

It is often the case when associative operators have a unit and we know since the early works on *i.e.* OBJ, that this is quite useful from a rule programming point of view. For example, to state *a list* L *that contains the objects* a *and* b. This can be expressed by the pattern $f(x, f(a, f(y, f(b, z))))$, where $x, y, z \in \mathcal{X}$, which will match $f(c, f(a, f(d, f(b, e))))$ but not $f(a, b)$ or $f(c, f(a, b))$. But if f has for unit e_f, the previous pattern will match modulo $\mathcal{AU}$, producing the substitution $\{x \mapsto e_f, y \mapsto e_f, z \mapsto e_f\}$ for $f(a, b)$, and $\{x \mapsto c, y \mapsto e_f, z \mapsto e_f\}$ for $f(c, f(a, b))$. By allowing unit elements, we increase the agility of the associative patterns.

An immediate consequence of allowing unit elements is that the set of matches becomes trivially infinite. For instance, $\mathcal{S}ol(x \prec_{\mathcal{AU}} a) = \{\{x \mapsto a\}, \{x \mapsto f(e_f, a)\}, \{x \mapsto f(a, e_f)\}\ etc\}$, and therefore we have to compute matching substitutions normalized with the set of unit rules $\mathsf{U} = \{f(e_f, x) \to x, f(x, e_f) \to x\}$. We replace $\mathsf{SymbolClash}$ rules in $\mathcal{A}$-$\mathsf{Matching}$ to appropriately handle unit elements (remember that we assume, because of modularity, that we only have in $\mathcal{F}$ a single binary $\mathcal{AU}$ symbol f, and constants, including e_f):

$$\begin{aligned}
\mathsf{SymbolClash}_1^+ \quad & f(p_1, p_2) \prec_{\mathcal{AU}} a \mapsto (p_1 \prec_{\mathcal{AU}} e_f \wedge p_2 \prec_{\mathcal{AU}} a) \vee \\
& \qquad\qquad (p_1 \prec_{\mathcal{AU}} a \wedge p_2 \prec_{\mathcal{AU}} e_f) \\
\mathsf{SymbolClash}_2^+ \quad & a \prec_{\mathcal{AU}} f(p_1, p_2) \mapsto (e_f \prec_{\mathcal{AU}} p_1 \wedge a \prec_{\mathcal{AU}} p_2) \vee \\
& \qquad\qquad (a \prec_{\mathcal{AU}} p_1 \wedge e_f \prec_{\mathcal{AU}} p_2)
\end{aligned}$$

In addition, we keep all other transforation rules, only changing all match symbols from $\mathcal{A}$ to $\mathcal{AU}$. The new system, named $\mathcal{AU}$-$\mathsf{Matching}$, is clearly terminating without producing in general a minimal set of solutions. After proving its correctness, we will see what can be done in order to minimize the set of solutions. The proof of correctness uses the following lemma:

Lemma 3.1. *Let t_1 and t_2 be two terms, matching them modulo $\mathcal{AU}$ is equivalent to match their U-normal forms modulo $\mathcal{A}$:*

$$t_1 \prec_{\mathcal{AU}} t_2 \Leftrightarrow t_{1\downarrow_U} \prec_{\mathcal{A}} t_{2\downarrow_U}$$

Proof. Direct application of [11][Theorem 3.3], since the unit rules are linear and terminating modulo $\mathcal{A}$, and associativity is regular. □

Proposition 3.3. *The rules of $\mathcal{AU}$-$\mathsf{Matching}$ are sound and preserving modulo $\mathcal{AU}$.*

Proof. The proof is in the Appendix B. □

In order to avoid redundant solutions we further consider that all the terms are in normal form *w.r.t.* the rewrite system U. Therefore, we perform a normalized rewriting [19] modulo U. This technique ensures us that before applying any of the rules in Figure 1, the terms are in normal forms *w.r.t.* U. Another approach would be to normalize with U the solutions obtained by $\mathcal{AU}$-$\mathsf{Matching}$, and to eliminate duplicates.

Example 3.2.
We can express that we want an f that contains an a in the following way:
$f(x, f(a, y)) \prec\!\!\!\prec_{\mathcal{AU}} f(a, b)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{Mutate}}(x \prec\!\!\!\prec_{\mathcal{AU}} a \wedge f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} b) \vee$
$\exists z(f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} f(z, b) \wedge f(x, z) \prec\!\!\!\prec_{\mathcal{AU}} a) \vee \exists z(x \prec\!\!\!\prec_{\mathcal{AU}} f(a, z) \wedge f(z, f(a, y)) \prec\!\!\!\prec_{\mathcal{AU}} b)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{SymbolClash}_1^+,\mathsf{ConstantClash},\mathsf{Propag}} \exists z(f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} f(z, b) \wedge f(x, z) \prec\!\!\!\prec_{\mathcal{AU}} a) \vee$
$\exists z(x \prec\!\!\!\prec_{\mathcal{AU}} f(a, z) \wedge f(z, f(a, y)) \prec\!\!\!\prec_{\mathcal{AU}} b)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{SymbolClash}_1^+,\mathsf{ConstantClash},\mathsf{Propag}} \exists z(f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} f(z, b) \wedge f(x, z) \prec\!\!\!\prec_{\mathcal{AU}} a)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{SymbolClash}_1^+} \exists z(f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} f(z, b) \wedge$
$((x \prec\!\!\!\prec_{\mathcal{AU}} e_f \wedge z \prec\!\!\!\prec_{\mathcal{AU}} a) \vee (x \prec\!\!\!\prec_{\mathcal{AU}} a \wedge z \prec\!\!\!\prec_{\mathcal{AU}} e_f)))$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{DistributeAnd},\mathsf{Replacement},\mathsf{Exists}}(x \prec\!\!\!\prec_{\mathcal{AU}} e_f \wedge f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} f(a, b))$
$\vee (x \prec\!\!\!\prec_{\mathcal{AU}} a \wedge f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} b)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{SymbolClash}_1^+,\mathsf{ConstantClash},\mathsf{Propag}} x \prec\!\!\!\prec_{\mathcal{AU}} e_f \wedge f(a, y) \prec\!\!\!\prec_{\mathcal{AU}} f(a, b)$
$\overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathsf{Mutate},\mathsf{SymbolClash}_1^+,\mathsf{Replacement},\mathsf{ConstantClash},\mathsf{Propag},\mathsf{Delete}} x \prec\!\!\!\prec_{\mathcal{AU}} e_f \wedge y \prec\!\!\!\prec_{\mathcal{AU}} b.$

4 Equational anti-pattern matching

In [16], we studied the anti-patterns in the case of the empty theory. In this section we will generalize for an arbitrary *regular* equational theory $\mathcal{E}$, that doesn't contain the symbol $\neg$.

Definition 4.1 (Equational membership and set equality). *Given an equational theory $\mathcal{E}$ and the term sets A and B, we define:*

1. $t \in_{\mathcal{E}} A \overset{!}{\Leftrightarrow} \exists t' \in A$ *such that* $t =_{\mathcal{E}} t'$;
2. $A \subseteq_{\mathcal{E}} B \overset{!}{\Leftrightarrow} \forall t_1 \in A,\ t_1 \in_{\mathcal{E}} B$;
3. $A =_{\mathcal{E}} B \overset{!}{\Leftrightarrow} A \subseteq_{\mathcal{E}} B$ *and* $B \subseteq_{\mathcal{E}} A$.

In the empty theory, given $q \in \mathcal{AT}(\mathcal{F}, \mathcal{X})$ and $t \in \mathcal{T}(\mathcal{F})$, the matching equation $q \prec\!\!\!\prec t$ has a solution when there exists a substitution σ such that $t \in [\![\sigma(q)]\!]_g$ [16]. This is extended to matching modulo $\mathcal{E}$ as follows:

Definition 4.2 (Solutions of anti-pattern matching equations). *For all $q \in \mathcal{AT}(\mathcal{F}, \mathcal{X})$ and $t \in \mathcal{T}(\mathcal{F})$, the solutions of the anti-pattern matching equation $q \prec\!\!\!\prec_{\mathcal{E}} t$ are:*

$$Sol(q \prec\!\!\!\prec_{\mathcal{E}} t) = \{\sigma \mid t \in_{\mathcal{E}} [\![\sigma(q)]\!]_g,\ \textit{with } \sigma \in \mathcal{GS}(q)\}.$$

A general anti-pattern matching problem P is any first-order expression whose atomic formulae are anti-pattern matching equations. To define their solutions, we rely on the usual definition of validity in predicate logic:

Definition 4.3 (Solutions of anti-pattern matching problems).

1. $\models q \prec\!\!\!\prec_{\mathcal{E}} t \overset{!}{\Leftrightarrow} \models t \in_{\mathcal{E}} [\![q]\!]_g$;
2. $\mathcal{S}ol_{\mathcal{E}}(P) \triangleq \{\sigma \mid \models \sigma(P)\}$.

Let us look at several examples of anti-pattern matching modulo some usual equational theories:

Example 4.1. In the syntactic case:

- $Sol(f(\neg a, x) \prec\!\!\!\prec f(b, c)) = \{x \mapsto c\}$,
- $Sol(f(x, \neg g(x)) \prec\!\!\!\prec f(a, g(b))) = \{x \mapsto a\}$,
- $Sol(f(x, \neg g(x)) \prec\!\!\!\prec f(a, g(a))) = \emptyset$.

Assuming that f is associative provides a useful expressivity:

- $Sol(f(x, f(\neg a, y)) \prec\!\!\!\prec_{\mathcal{A}} f(b, f(a, f(c, d))) = \{x \mapsto f(b, a), y \mapsto d\}$, while
- $Sol(f(x, f(\neg a, y)) \prec\!\!\!\prec_{\mathcal{A}} f(a, f(a, f(a, a))) = \emptyset$.

We can also express that we do not want an a below f:

- $Sol(\neg f(x, f(a, y)) \prec\!\!\!\prec_{\mathcal{A}} f(b, f(a, f(c, d))) = \emptyset$,
- $Sol(\neg f(x, f(a, y)) \prec\!\!\!\prec_{\mathcal{A}} f(b, f(b, f(c, d))) = \Sigma$.

A combination of the last two associative patterns, $\neg f(x, f(\neg a, y))$ would naturally correspond to *an f with only a inside*: $Sol(\neg f(x, f(\neg a, y)) \prec\!\!\!\prec_{\mathcal{A}} f(a, f(a, f(b, a))) = \emptyset$, while $Sol(\neg f(x, f(\neg a, y)) \prec\!\!\!\prec_{\mathcal{A}} f(a, f(a, f(a, a))) = \Sigma$.

Non-linearity can be also useful $Sol(\neg f(x, x) \prec\!\!\!\prec_{\mathcal{A}} f(a, f(b, f(a, b))) = \emptyset$, but $Sol(\neg f(x, x) \prec\!\!\!\prec_{\mathcal{A}} f(a, f(b, f(a, c))) = \Sigma$.

If besides associative, we consider that f is also commutative, we have the following results for matching modulo $\mathcal{AC}$: $Sol(f(x, f(\neg a, y)) \prec\!\!\!\prec_{\mathcal{AC}} f(a, f(b, c))) = \{x \mapsto a, y \mapsto c\}, \{x \mapsto a, y \mapsto b\}, \{x \mapsto b, y \mapsto a\}, \{x \mapsto c, y \mapsto a\}$.

4.1 From anti-pattern matching to equational problems

As we did for the empty theory [16], to solve anti-pattern matching modulo, we first transform the initial matching problem into an equational one. This is performed using the following transformation rule:

$$\mathsf{ElimAnti}\ q[\neg q']_\omega \prec\!\!\!\prec_{\mathcal{E}} t \mapsto\!\!\!\!\mapsto \exists z\ q[z]_\omega \prec\!\!\!\prec_{\mathcal{E}} t\ \wedge\ \forall x \in \mathcal{FV}ar(q')\ not(q[q']_\omega \prec\!\!\!\prec_{\mathcal{E}} t)$$
$$\text{if } \forall\ \omega' < \omega,\ q(\omega') \neq \neg \text{ and } z \text{ a fresh variable}$$

An anti-pattern matching problem P not containing any $\neg$ symbol, is a first-order formula where the symbol *not* is the usual negation of predicate logic and the symbol $\prec\!\!\!\prec_{\mathcal{E}}$ is interpreted as $=_{\mathcal{E}}$. Therefore they are exactly $\mathcal{E}$-disunification problems.

Proposition 4.1. *The rule* ElimAnti *is sound and preserving modulo $\mathcal{E}$.*

Proof. We consider a position ω such that $q[\neg q']_\omega$ and $\forall\ \omega' < \omega,\ q(\omega') \neq \neg$. Considering as usual that $Sol(A \wedge B) = Sol(A) \cap Sol(B)$ we have the following result for the right hand side of the rule:

$$Sol(\exists z\ q[z]_\omega \prec\!\!\!\prec_{\mathcal{E}} t\ \wedge\ \forall x \in \mathcal{FV}ar(q')\ not(q[q']_\omega \prec\!\!\!\prec_{\mathcal{E}} t))$$
$$= Sol(\exists z\ q[z]_\omega \prec\!\!\!\prec_{\mathcal{E}} t\ ) \cap\ Sol(\forall x \in \mathcal{FV}ar(q')\ not(q[q']_\omega \prec\!\!\!\prec_{\mathcal{E}} t))$$

From Definition 4.3, $Sol(\exists z\ q[z]_\omega \prec\!\!\!\prec_{\mathcal{E}} t)$ is equal to:

$$\{\sigma \mid \mathsf{Dom}(\sigma) = \mathcal{FV}ar(q[z])\backslash\{z\} \text{ and } \exists\rho \text{ with } \mathsf{Dom}(\rho) = \{z\}, t \in_{\mathcal{E}} [\![\rho\sigma(q[z]_\omega)]\!]_g\} \tag{1}$$

Also from Definition 4.3, $Sol(\forall x \in \mathcal{FV}ar(q')\ not(q[q']_\omega \prec\!\!\!\prec_{\mathcal{E}} t))$ is equal to:

$$\{\sigma \mid t \notin_{\mathcal{E}} [\![\sigma(q[q']_\omega)]\!]_g \ with\ \mathsf{Dom}(\sigma) = \mathcal{FV}ar(q[q']) \setminus \mathcal{FV}ar(q')\} \tag{2}$$

For the left part of the rule ElimAnti, by Definition 4.2 we have:

$$\begin{aligned} Sol\ (q[\neg q']_\omega \prec\!\!\!\prec_{\mathcal{E}} t) &= \{\sigma \mid t \in_{\mathcal{E}} [\![\sigma(q[\neg q']_\omega)]\!]_g,\ with\ \mathsf{Dom}(\sigma) = \mathcal{FV}ar(q[\neg q'])\} \\ &= \{\sigma \mid t \in_{\mathcal{E}} ([\![\sigma(q[z]_\omega)]\!]_g \backslash [\![\sigma(q[q']_\omega)]\!]_g),\ with \ldots\},\ \text{since } \forall\omega' < \omega, q(\omega') \neq \neg \\ &= \{\sigma \mid t \in_{\mathcal{E}} [\![\sigma(q[z]_\omega)]\!]_g \text{ and } t \notin_{\mathcal{E}} [\![\sigma(q[q']_\omega)]\!]_g,\ with\ \mathsf{Dom}(\sigma) = \mathcal{FV}ar(q[\neg q'])\} \\ &= \{\sigma \mid t \in_{\mathcal{E}} [\![\sigma(q[z]_\omega)]\!]_g,\ with \ldots\} \cap \{\sigma \mid t \notin_{\mathcal{E}} [\![\sigma(q[q']_\omega)]\!]_g\ with \ldots\} \end{aligned} \tag{3}$$

Now it remains to check the equivalence of (3) with the intersection of (1) and (2). First of all, $\mathcal{FV}ar(q[z])\backslash\{z\} = \mathcal{FV}ar(q[q']) \setminus \mathcal{FV}ar(q') = \mathcal{FV}ar(q[\neg q'])$ which means that we have the same domain for σ in (3), (1), and (2). Therefore, we have to prove:

$$\{\sigma \mid \exists\rho \text{ with } \mathsf{Dom}(\rho) = \{z\} \text{ and } t \in_{\mathcal{E}} [\![\rho\sigma(q[z]_\omega)]\!]_g\} = \{\sigma \mid t \in_{\mathcal{E}} [\![\sigma(q[z]_\omega)]\!]_g\} \tag{4}$$

But σ does not instantiate z, and this means that the ground semantics will give to z all the possible values for the right part of (4). In the same time, having ρ existentially quantified allows z to be instantiated with any value such that $t \in_{\mathcal{E}} [\![\rho\sigma(q[z]_\omega)]\!]_g$ is valid, and therefore (4) is true. As we considered an arbitrary $\neg$, we can conclude that the rule is sound and preserving, wherever it is applied on a term. □

The normal forms *w.r.t.* ElimAnti of anti-pattern matching problems are equational problems that are undecidable in general [26] even in case of $\mathcal{A}$ or $\mathcal{AC}$ theories, but that can be solved with adapted techniques in specific cases, like for the empty theory [7] or for the existential fragment of the $\mathcal{AC}$-theory [5].

Summarizing, if we know how to solve equational problems modulo $\mathcal{E}$, then any anti-pattern matching problem modulo $\mathcal{E}$ can be translated into equivalent equational problems using ElimAnti and further solved. These affirmations are formalized by the following Proposition:

Proposition 4.2. *An anti-pattern matching problem can always be translated into an equivalent equational problem in a finite number of steps.*

Proof. We showed in the proof of Proposition 4.1 that ElimAnti preserves the solutions if applied on a matching problem. Each of its applications transforms one equation in two equivalent equations (equivalent in the means of solutions) that contain at least one $\neg$ in minus. Therefore, for a finite number n of $\neg$ symbols, ElimAnti terminates and it is easy to show that the normal forms contain at most 2^n equations and disequations. □

4.2 Matching associative anti-patterns with neutral elements

Combining equational patterns with complement symbols provides us with a very expressive concept to use in rule-based languages. From simpler searches like *the lists that do not contain* a or *the lists that contain at least one element different from* a to more complex ones like *the lists with all elements equal to* a are all made possible with the use of associative anti-patterns. For example, the patterns that correspond to previous searches would be respectively $\neg f(x, f(a, y))$, $f(x, f(\neg a, y))$ and $\neg f(x, f(\neg a, y))$.

To compute the set of solutions for an $\mathcal{AU}$ anti-pattern matching equation we develop now a specific approach that can also be adapted the the empty theory.

Definition 4.4 (Algorithm $\mathcal{AU}$-AntiMatching). *Given an $\mathcal{AU}$ anti-pattern matching problem:*

1. *Normalize with* ElimAnti *for eliminating all $\neg$ symbols,*
2. *Normalize with $\mathcal{AU}$-*Matching *each resulted equality separately,*
3. *Eliminate all variables that are quantified (with $\exists$ or $\forall$),*
4. *Normalize all the resulted expression with the rule* DistributeAnd*:* $\mathsf{S}_1 \wedge (\mathsf{S}_2 \vee \mathsf{S}_3) \mapsto\!\!\!\!\!\rightarrow (\mathsf{S}_1 \wedge \mathsf{S}_2) \vee (\mathsf{S}_1 \wedge \mathsf{S}_3)$ *followed by* Replacement *on each conjunction in a top-down manner:* $z = t \wedge \mathsf{S} \mapsto\!\!\!\!\!\rightarrow z = t \wedge \{z \to t\}\mathsf{S}$
5. *Clean the result with the rules:* $\mathsf{SymbolClash}_1$, $\mathsf{SymbolClash}_2$, Delete, $\mathsf{PropagClash}_1$, $\mathsf{PropagClash}_2$, $\mathsf{PropagSuccess}_1$, $\mathsf{PropagSuccess}_2$, $\mathsf{NotTrue}(not(\top) \mapsto\!\!\!\!\!\rightarrow \bot)$, $\mathsf{NotFalse}(not(\bot) \mapsto\!\!\!\!\!\rightarrow \top)$ *and* GroundClash $(t_1 \not\prec_{\mathcal{AU}} t_2 \mapsto\!\!\!\!\!\rightarrow \bot$ if $t_1, t_2 \in \mathcal{T}(\mathcal{F})$ *and* $t_1 \neq_{\mathcal{AU}} t_2)$.

The main idea of this algorithm is that after we normalize with ElimAnti, everything that is under a *not* symbol can be reduced to either $\top$ or $\bot$. In other words, no free variable can appear in the equalities under a *not* symbol, because they are either quantified or replaced with ground values from the equalities of the context by the rule Replacement.

To explain the 3^{rd} step, let us take a general problem on which this step is applied:

$$\ldots \exists z(\bigvee\ x_1 \not\prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge z \not\prec_{\mathcal{AU}} t_i \wedge \ldots \wedge x_n \not\prec_{\mathcal{AU}} t_n)$$
$$\wedge\ \forall y_1, \ldots, y_m\ not(\bigvee\ x_1 \not\prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge y_i \not\prec_{\mathcal{AU}} t_i \wedge \ldots \wedge x_p \not\prec_{\mathcal{AU}} t_p) \ldots$$

This is as general as possible, no matter the anti-pattern matching problem we began with, because we are working with normal forms of pattern matching as stated in Theorem 3.1. Moreover, because the algorithm $\mathcal{AU}$-Matching contains the rule Replacement, in any conjunction there is at most one occurrence of each variable .

Therefore, we have two types of variables which we intend to eliminate:

- z: this is a fresh variable, as we are insured by the definition. This means it only appears once in each conjunction, and therefore the quantification of all the expression can be changed in : $\bigvee\ x_1 \not\prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge \exists z(z \not\prec_{\mathcal{AU}} t_i) \wedge \ldots \wedge x_n \not\prec_{\mathcal{AU}} t_n$ which is equivalent to: $\bigvee\ x_1 \not\prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge x_n \not\prec_{\mathcal{AU}} t_n$.

– y_i: we perform the following sequence of transformations:

$$\begin{aligned}
&\forall y_1, \ldots, y_m\ not(\bigvee\ x_1 \prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge y_i \prec_{\mathcal{AU}} t_i \wedge \ldots \wedge x_p \prec_{\mathcal{AU}} t_p)\\
&\quad \mapsto\!\!\!\!\rightarrow not(\exists y_1, \ldots, y_m\ (\bigvee\ x_1 \prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge y_i \prec_{\mathcal{AU}} t_i \wedge \ldots \wedge x_p \prec_{\mathcal{AU}} t_p))\\
&\quad \mapsto\!\!\!\!\rightarrow not(\bigvee\ x_1 \prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge \exists y_i(y_i \prec_{\mathcal{AU}} t_i) \wedge \ldots \wedge x_p \prec_{\mathcal{AU}} t_p)\\
&\quad \mapsto\!\!\!\!\rightarrow not(\bigvee\ x_1 \prec_{\mathcal{AU}} t_1 \wedge \ldots \wedge x_p \prec_{\mathcal{AU}} t_p)
\end{aligned}$$

This step can be easily modeled by the rewrite system: ForAllTransform $(\forall y_1, \ldots, y_m\ not(S) \mapsto\!\!\!\!\rightarrow not(\exists y_1, \ldots, y_m\ S))$, Exists_1, Exists_2, $\mathsf{PropagSuccess}_1$.

The algorithm has several interesting properties:

Proposition 4.3. *The application of $\mathcal{AU}$-*AntiMatching *is sound and preserving.*

Proof. We prove these properties for each step:

– Step 1: showed in the proof of Proposition 4.1,
– Step 2: showed in the proof of Theorem 3.1,
– Step 3: results immediately from the explanations given above,
– Step 4: the rule DistributeAnd is trivial and Replacement was shown to be sound and preserving in [7,4],
– Step 5: the NotTrue,NotFalse and GroundClash are trivial, and the rest are the same as in $\mathcal{AU}$-Matching.

□

Proposition 4.4. *The normal forms of $\mathcal{AU}$-*AntiMatching *do not contain any* not *symbols.*

Proof. The non-variable terms are clearly reduced to either $\top$ or $\bot$ and further eliminated. The variables under the *not* symbol can be of two types:

1. quantified, which will be eliminated by the 4^{th} step,
2. not quantified. In this case, it means that they were not under an $\neg$ symbol, and therefore they are free variables that we can find in the context of *not*. In other words, for any $x_i \prec_{\mathcal{AU}} t_i$ under the *not* symbol, where x_i is not universally quantified, there exists a corresponding $x_i \prec_{\mathcal{AU}} t_i$ in the context. Given that, the rule Replacement will transform the equations under the *not* in simpler equations that will be further reduced to $\top$ (for our example, $t_i \prec_{\mathcal{AU}} t_i$).

□

Thanks to all these properties, the possible normal forms of $\mathcal{AU}$-AntiMatching are the same as the ones exposed in Theorem 3.1.

Example 4.2.

$$\begin{aligned}
&f(\neg a, y) \prec_{\mathcal{AU}} b \mapsto\!\!\!\!\rightarrow_{\mathsf{ElimAnti}} \exists z\ f(z, y) \prec_{\mathcal{AU}} b \wedge not(f(a, y) \prec_{\mathcal{AU}} b)\\
&\mapsto\!\!\!\!\rightarrow_{\mathsf{SymbolClash}_1^+} \exists z\ (z \prec_{\mathcal{AU}} e_f \wedge y \prec_{\mathcal{AU}} b \vee z \prec_{\mathcal{AU}} b \wedge y \prec_{\mathcal{AU}} e_f)\\
&\wedge not((y \prec_{\mathcal{AU}} e_f \wedge a \prec_{\mathcal{AU}} b) \vee (y \prec_{\mathcal{AU}} b \wedge a \prec_{\mathcal{AU}} e_f))\\
&\mapsto\!\!\!\!\rightarrow_{\mathsf{ConstantClash,Propag}} \exists z\ (z \prec_{\mathcal{AU}} e_f \wedge y \prec_{\mathcal{AU}} b \vee z \prec_{\mathcal{AU}} b \wedge y \prec_{\mathcal{AU}} e_f) \wedge not(\bot)\\
&\mapsto\!\!\!\!\rightarrow_{\mathsf{Elim.\ quantified\ variables}} (y \prec_{\mathcal{AU}} b \vee y \prec_{\mathcal{AU}} e_f) \wedge not(\bot)\\
&\mapsto\!\!\!\!\rightarrow_{\mathsf{DistributeAnd}} (y \prec_{\mathcal{AU}} b \vee y \prec_{\mathcal{AU}} e_f) \wedge not(\bot)\\
&\mapsto\!\!\!\!\rightarrow_{\mathsf{DistributeAnd}} (y \prec_{\mathcal{AU}} b \wedge not(\bot)) \vee (y \prec_{\mathcal{AU}} e_f \wedge not(\bot))\\
&\mapsto\!\!\!\!\rightarrow_{\mathsf{NotFalse,PropagSuccess}} y \prec_{\mathcal{AU}} b \vee y \prec_{\mathcal{AU}} e_f.
\end{aligned}$$

5 Anti-matching modulo in Tom

Anti-patterns are successfully integrated in the Tom[2] language. Due to the lack of space, we won't present here how they were implemented but rather show you how they can be used in this language and the expressivity they add to its pattern matching capabilities. Tom provides syntactic matching and list-matching, which is a special case of $\mathcal{AU}$ matching.

In order to support anti-patterns, we enriched the syntax of the Tompatterns to allow the use the operator '!' (representing '¬'). For syntactic matching, here is an example of a *match* in Tom:

```
%match(s) {
  f(a(),g(b()))  -> { /* action 1: executed when f(a,g(b))<<s  */ }
  f(!a(),g(b())) -> { /* action 2: when f(x,g(b))<<s with x!=a */ }
  !f(x,!g(x))    -> { /* action 3: when not f(x,y)<<s or ...   */ }
  !f(x,g(y))     -> { /* action 4 */ }
}
```

Similarly to `switch/case`, an action part is executed when its corresponding pattern matches the subject `s`. Note that non-linear patterns are allowed. When combined with lists, the expressiveness of the anti-patterns is even more impressing:

```
%match(s) {
  conc(_*,a(),_*)      -> { /* executed when s contains a */ }
  conc(_*,!a(),_*)     -> { /* when s has one elem. diff. from a */ }
  !conc(_*,a(),_*)     -> { /* when s does not contain a */ }
  !conc(_*,!a(),_*)    -> { /* when s contains only a */ }
  conc(_*,x,_*,x,_*)   -> { /* s has at least 2 equal elements */ }
  !conc(_*,x,_*,x,_*)  -> { /* s has only distinct elements */ }
  conc(_*,x,_*,!x,_*)  -> { /* s has at least 2 different elem. */ }
  !conc(_*,x,_*,!x,_*) -> { /* when s contains only equal elem. */ }
}
```

In the above patterns, `_*` stands for any sublist, `a()` is a constant and `x` is a variable that cannot be instantiated by the empty list.

There are mainly two advantages of using anti-patterns: the first one is that without their usage, one would be forced to verify additional condition in the action part, which would make the code a lot more complicated and hard to read. The second one is efficiency, because they allow the verification of some conditions earlier. For example, imagine that we want to search for a list that contains something different from `a()`, and then a `b()`. This corresponds to the anti-pattern `conc(_*,!a(),_*,b(),_*)`. Without using anti-patterns, we would write the following code:

[2] http://tom.loria.fr

```
%match(s) {
  conc(_*,x,_*,b(),_*) -> { if (x != a()) {/* action */} }
}
```

But this is quite inefficient, as we cover the rest of the list in search of a `b()`, without knowing that `x` had had the good value, and if not, we will be forced to backtrack. In the case of using `conc(_*,!a(),_*,b(),_*)`, we only continue the search in the list after finding an element different from `a()`.

6 Conclusion

We have generalized the notion of anti-pattern matching to anti-pattern matching modulo an arbitrary regular theory $\mathcal{E}$. Because of their usefulness for rule-based programming, we chose to exemplify the usefulness of anti-patterns for the $\mathcal{A}$ and $\mathcal{AU}$ theories. To that end, we presented a rule-based algorithm for solving $\mathcal{A}$ and $\mathcal{AU}$ matching and we showed its correctness and completeness. Further on, we showed how an anti-pattern matching problem modulo can be systematically translated into an equivalent equational problem modulo. We applied this technique to translate an anti-pattern matching problem modulo $\mathcal{AU}$ into equivalent equational problems modulo $\mathcal{AU}$ and we presented an algorithm for solving these later ones. We finally sketched the integration of the anti-patterns in the Tom language for syntactic as well as for list matching.

The work in this paper opens a number of challenging questions like understanding the complexity of our rule-based algorithms and investigating anti-pattern matching modulo other equational theories, like for instance $\mathcal{AC}$ possibly with a unit.

We are also planning to provide adapted syntax and efficient tools to integrate anti-pattern matching modulo in search engines, as exemplified in the introduction.

References

1. F. Baader and T. Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998. 2
2. D. Benanav, D. Kapur, and P. Narendran. Complexity of matching problems. *J. Symb. Comput.*, 3(1-2):203–216, 1987. 4
3. H.-J. Bürckert. Matching — A special case of unification? In C. Kirchner, editor, *Unification*, pages 125–138. Academic Press inc., London, 1990. 4
4. H. Comon. *Unification et disunification. Théories et applications*. Thèse de Doctorat d'Université, Institut Polytechnique de Grenoble (France), 1988. 12
5. H. Comon. Complete axiomatizations of some quotient term algebras. *Theoretical Computer Science*, 118(2), Sept. 1993. 10
6. H. Comon and C. Kirchner. Constraint solving on terms. *Lecture Notes in Computer Science*, 2002:47–103, 2001. 5, 6
7. H. Comon and P. Lescanne. Equational problems and disunification. In C. Kirchner, editor, *Unification*, pages 297–352. Academic Press inc., London, 1990. 10, 12

8. S. Eker. Associative matching for linear terms. Report CS-R9224, CWI, 1992. ISSN 0169-118X. 5
9. S. Eker. Associative-commutative rewriting on large terms. In *RTA*, volume 2706, pages 14–29, 2003. 5
10. M. Hermann and P. G. Kolaitis. The complexity of counting problems in equational matching. *Journal of Symbolic Computation*, 20(3):343–362, sep 1995. 4
11. G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, Oct. 1980. Preliminary version in 18th Symposium on Foundations of Computer Science, IEEE, 1977. 7
12. J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991. 2
13. D. Kapur and P. Narendran. Np-completeness of the set unification and matching problems. In *Proc. of the 8th international conference on Automated deduction*, pages 489–495, New York, NY, USA, 1986. Springer-Verlag New York, Inc. 4
14. C. Kirchner. Computing unification algorithms. In *Proceedings 1st IEEE Symposium on Logic in Computer Science, Cambridge (Mass., USA)*, pages 206–216, 1986. 5
15. C. Kirchner and F. Klay. Syntactic theories and unification. In *Proceedings 5th IEEE Symposium on Logic in Computer Science, Philadelphia (Pa., USA)*, pages 270–277, June 1990. 5, 6
16. C. Kirchner, R. Kopetz, and P. Moreau. Anti-pattern matching. In *Proceedings of ESOP*, 2007. 1, 3, 4, 5, 8, 9
17. C. Kirchner, P.-E. Moreau, and A. Reilles. Formal validation of pattern matching code. In P. Barahona and A. Felty, editors, *Proceedings of the 7th ACM SIGPLAN PPDP*, pages 187–197. ACM, July 2005. 2
18. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32(2):129–198, 1977. 5
19. C. Marché. Normalized rewriting: an alternative to rewriting modulo a set of equations. *Journal of Symbolic Computation*, 21(3):253–288, 1996. 7
20. P.-E. Moreau, C. Ringeissen, and M. Vittek. A Pattern Matching Compiler for Multiple Target Languages. In G. Hedin, editor, *12th Conference on Compiler Construction, Warsaw (Poland)*, volume 2622 of *LNCS*, pages 61–76. Springer-Verlag, May 2003. 2
21. T. Nipkow. Proof transformations for equational theories. In *Proceedings 5th IEEE Symposium on Logic in Computer Science, Philadelphia (Pa., USA)*, pages 278–288, June 1990. 5
22. T. Nipkow. Combining matching algorithms: The regular case. *Journal of Symbolic Computation*, 12(6):633–653, 1991. 4
23. G. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972. 5
24. A. Reilles. *Réécriture et compilation de confiance*. Thèse de Doctorat d'Université, Institut National Polytechnique de Lorraine, France, Nov. 2006. 4
25. C. Ringeissen. Combination of matching algorithms. In P. Enjalbert, E.-W. Mayr, and K.-W. Wagner, editors, *Proceedings 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen*, volume 775 of *Lecture Notes in Computer Science*, pages 187–198. Springer Verlag, feb 1994. 4
26. R. Treinen. A new method for undecidability proofs of first order theories. *Journal of Symbolic Computation*, 14(5):437–457, Nov. 1992. 10

A Proof of Proposition 3.1

Proposition 3.1. *Given a matching equation $p \ll_{\mathcal{A}} t$ with $p \in \mathcal{T}(\mathcal{F}, \mathcal{X}), t \in \mathcal{T}(\mathcal{F})$, the application of $\mathcal{A}$-*Matching *always terminates.*

Proof. Let D_0 be the initial problem, *i.e.* $D_0 = p \ll_{\mathcal{A}} t$. Further on, $D_0 \overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathcal{A}-\mathsf{Matching}} D_1 \overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathcal{A}-\mathsf{Matching}} \cdots \overset{}{\mapsto\!\!\!\!\rightarrow}_{\mathcal{A}-\mathsf{Matching}} D_n$. For all $i \in [1 \dots n]$, the size of D_i, denoted by $\|D_i\|$, is the multiset of its components, computed in the following way:

- $\|D_j \wedge D_k\| = \|D_j \vee D_k\| = \|D_i\| \cup \|D_j\|$,
- $\|\exists z(D_j)\| = \|D_j\|$,
- $\|\bot\| = \|\top\| = \{0\}$,
- $\|p' \ll_{\mathcal{A}} t'\| = \{\|t'\|\}$,
- $\|f(p_1, p_2)\| = 1 + \|p_1\| + \|p_2\|$,
- $\|a\| = 1$, for a a constant,
- $\|x\| = \|t\|$, if $x \in \mathcal{V}ar(p)$, *i.e.* a free variable of the initial problem,
- $\|x\| = \|t_i\| - 1$, if $x \notin \mathcal{V}ar(D_i)$ and $D_{i+1} = C[\exists x(C'[p_j \ll_{\mathcal{A}} t_j])]$ with $x \in \mathcal{V}ar(p_j)$. Therefore, each time a new existential variable is introduced, its size is computed and it remains unchanged afterwards.

Please note that when a free variable x is solved by a match equation $x \ll_{\mathcal{A}} t'$, $\|t'\|$ is always smaller or equal to the right-hand side of the initial problem, as we never increase the right-hand side of an equation by any of the rules. Moreover, when a existential variable is introduced in a left-hand side of an equation, its size is fixed to the size of the right-hand side minus 1. As further applications of the algorithm never increase the right-hand side, when solved, this variable's size can't exceed its fixed size.

The number of variables' occurrences in D is the sum of the occurrences in each term, and is denoted by $\#\mathcal{V}ar(D)$, *i.e.* $\#\mathcal{V}ar(D) = \sum \#\mathcal{V}ar(t)$, for all $t \in P$. The variables' occurrences in a term are computed as $\#\mathcal{V}ar(t) = \{\#(\omega) \mid t|_\omega \in \mathcal{X}\}$.

For proving termination, we consider a lexicographical order $\phi = (\phi_1, \phi_2)$, where $\phi_1 = \|D\|$, and $\phi_2 = \#\mathcal{V}ar(D)$, which is decreasing for the application of each rule:

- Mutate: $\|f(p_1, p_2) \ll_{\mathcal{A}} f(t_1, t_2)\| = \{\|f(t_1, t_2)\|\}$. The size of each equation from the right-hand side is strictly smaller. For example, $\|p_2 \ll_{\mathcal{A}} f(x, t_2)\| = \{\|f(x, t_2)\|\} < \{\|f(t_1, t_2)\|\}$ as $\|x\| = \|t_1\| - 1$. This implies that ϕ_1 is decreasing. Although ϕ_2 increases, ϕ is lexicographically decreasing.
- Replacement: when replacing a free variable, the size remains constant, as all the variables are in the left-hand sides. When replacing a quantified variable, the size remains the same or decreasing, as the variable couldn't have been instantiated with more than its initial size. Therefore ϕ_1 is decreasing or remaining constant, and as ϕ_2 is strictly decreasing, we have that ϕ is decreasing.

For the other rules that do not strictly decrease ϕ — like Exists_2 for instance, it is easy to find other orders to add to ϕ for having a global decreasing one (again, for Exists_2 would be the number of quantifiers). □

B Proof of Proposition 3.3

Proposition 3.3. *The rules of $\mathcal{AU}$-*Matching *are sound and preserving modulo $\mathcal{AU}$.*

Proof. Thanks to Proposition 3.2, we know that the rules are sound and preserving modulo $\mathcal{A}$. In order to be also valid modulo $\mathcal{AU}$, they have to remain valid in the presence of the equations for neutral elements, as defined in Section 2. Let us first see the preserving property of the rules:

- ConstantClash, Replacement, Delete, Exists_1, Exists_2, DistributeAnd, PropagClash_1, PropagClash_2, PropagSuccess_1, PropagSuccess_2: these rules do not depend on the theory we consider.
- Mutate: we need to prove that for $\sigma \in \mathcal{S}ol(f(p_1, p_2) =_{\mathcal{AU}} f(t_1, t_2))$, $\exists \rho$ such that at least one of the following is true:
  - $\sigma\rho(p_1) =_{\mathcal{AU}} \rho(t_1) \wedge \sigma\rho(p_2) =_{\mathcal{AU}} \rho(t_2)$
  - $\sigma\rho(p_2) =_{\mathcal{AU}} \rho(f(x, t_2)) \wedge \sigma\rho(f(p_1, x)) =_{\mathcal{AU}} \rho(t_1)$
  - $\sigma\rho(p_1) =_{\mathcal{AU}} \rho(f(t_1, x)) \wedge \sigma\rho(f(x, p_2)) =_{\mathcal{AU}} \rho(t_2)$

  which are equivalent, by Lemma 3.1, to:
  1. $\sigma\rho(p_1)_{\downarrow_U} =_{\mathcal{A}} \rho(t_1)_{\downarrow_U} \wedge \sigma\rho(p_2)_{\downarrow_U} =_{\mathcal{A}} \rho(t_2)_{\downarrow_U}$
  2. $\sigma\rho(p_2)_{\downarrow_U} =_{\mathcal{A}} \rho(f(x, t_2))_{\downarrow_U} \wedge \sigma\rho(f(p_1, x))_{\downarrow_U} =_{\mathcal{A}} \rho(t_1)_{\downarrow_U}$
  3. $\sigma\rho(p_1)_{\downarrow_U} =_{\mathcal{A}} \rho(f(t_1, x))_{\downarrow_U} \wedge \sigma\rho(f(x, p_2))_{\downarrow_U} =_{\mathcal{A}} \rho(t_2)_{\downarrow_U}$

  But $\sigma \in \mathcal{S}ol(f(p_1, p_2) =_{\mathcal{AU}} f(t_1, t_2)) \Rightarrow f(\sigma\rho(p_1), \sigma\rho(p_2)) =_{\mathcal{AU}} f(\rho(t_1), \rho(t_2))$ for a chosen ρ which is equivalent to $f(\sigma\rho(p_1), \sigma\rho(p_2))_{\downarrow_U} =_{\mathcal{A}} f(\rho(t_1), \rho(t_2))_{\downarrow_U}$. We have the following possible cases:
  1. neither $f(\sigma\rho(p_1), \sigma\rho(p_2))$ nor $f(\rho(t_1), \rho(t_2))$ can be reduced by U. This means that $f(\sigma\rho(p_1), \sigma\rho(p_2)) =_{\mathcal{AU}} f(\rho(t_1), \rho(t_2)) \Leftrightarrow f(\sigma\rho(p_1), \sigma\rho(p_2)) =_{\mathcal{A}} f(\rho(t_1), \rho(t_2))$, which implies (by the rule Mutate that was proved to be $\mathcal{A}$-preserving) the disjunction of the three cases above.
  2. only $f(\sigma\rho(p_1), \sigma\rho(p_2))$ can be reduced by U:
     (a) $\sigma\rho(p_1)_{\downarrow_U} \neq e_f$, $\sigma\rho(p_2)_{\downarrow_U} \neq e_f$. This gives $f(\sigma\rho(p_1)_{\downarrow_U}, \sigma\rho(p_2)_{\downarrow_U}) =_{\mathcal{A}} f(\rho(t_1), \rho(t_2))$ which again implies the three cases above.
     (b) $\sigma\rho(p_1)_{\downarrow_U} = e_f$. This results in $\sigma\rho(p_2)_{\downarrow_U} =_{\mathcal{A}} f(\rho(t_1), \rho(t_2))$ which is equivalent with the second case for $\rho(x) = \rho(t_1)$.
     (c) $\sigma\rho(p_2)_{\downarrow_U} = e_f$. Implies the second case with $\rho(x) = \rho(t_2)$.
  3. only $f(\rho(t_1), \rho(t_2))$ can be reduced. As above , we consider all the three possible cases reasoning exactly in the same fashion.
  4. both $f(\sigma\rho(p_1), \sigma\rho(p_2))$ and $f(\rho(t_1), \rho(t_2))$ are reducible. This case is just the combination of all the possibilities we have enounced above, therefore nine cases, which are solved similarly.
- SymbolClash_1^+: $\sigma \in \mathcal{S}ol(f(p_1, p_2) =_{\mathcal{AU}} g(\overline{t})) \Rightarrow f(\sigma(p_1), \sigma(p_2))_{\downarrow_U} =_{\mathcal{A}} a$. When both $\sigma(p_1)_{\downarrow_U}$ and $\sigma(p_2)_{\downarrow_U}$ are different from e_f, the equation $f(\sigma(p_1)_{\downarrow_U}, \sigma(p_2)_{\downarrow_U}) =_{\mathcal{A}} a$ has no solution as SymbolClash can be applied. If at least one of them is equal to e_f, we have the exact correspondence with the right hand side of the rule: $\sigma(p_1)_{\downarrow_U} =_{\mathcal{A}} e_f \wedge \sigma(p_2)_{\downarrow_U} =_{\mathcal{A}} a \vee \sigma(p_1)_{\downarrow_U} =_{\mathcal{A}} a \wedge \sigma(p_2)_{\downarrow_U} =_{\mathcal{A}} e_f$.

– $\mathsf{SymbolClash}_2^+$: The same reasoning as above.

The soundness justification follows the same pattern. For example, for the rule Mutate, which is the most interesting one, we have to prove that there exists ρ, such that that given σ which validates at least one of the disjunctions, we obtain the left hand side of the rule. As above, first case is when only $\sigma\rho(p_1)$ and $\sigma\rho(p_2)$ can be reduced by U, and $\sigma\rho(p_1)_{\downarrow_U} \neq e_f$ and $\sigma\rho(p_2)_{\downarrow_U} \neq e_f$. The question if $\sigma\rho(p_1)_{\downarrow_U} =_{\mathcal{A}} \rho(t_1) \wedge \sigma\rho(p_2)_{\downarrow_U} =_{\mathcal{A}} \rho(t_2)$ implies $f(\sigma(p_1)_{\downarrow_U}, \sigma\rho(p_2)_{\downarrow_U}) =_{\mathcal{A}} f(\rho(t_1), \rho(t_2))$ is obviously true. The rest of the cases are similar. □