Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding
(work in progress)

Sophie Tourret[1], Pascal Fontaine[2,3], Daniel El Ouraoui[2], Haniel Barbosa[4] *†

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] University of Lorraine, CNRS, Inria, and LORIA, Nancy, France
[3] Université de Liège, Liège, Belgium
[4] Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Abstract

Recent work in extending SMT solvers to higher-order logic (HOL) has not explored lifting quantifier instantiation algorithms to perform higher-order unification. As a consequence, widely used instantiation techniques, such as trigger- and particularly conflict-based, can only be applied in a limited manner. Congruence closure with free variables (CCFV) is a decision procedure for the E-ground (dis-)unification problem, which is at the heart of these instantiation techniques. Here, as a first step towards fully supporting trigger- and conflict-based instantiation in HOL, we define the E-ground (dis-)unification problem in λ-free higher-order logic (λfHOL), an extension of first-order logic where function symbols may be partially applied and functional variables may occur, and extend CCFV to solve it. To improve scalability in the context of handling higher-order variables, we rely on an encoding of the CCFV search as a propositional formula. We present a solution reconstruction procedure so that models for the propositional formula lead to solutions for the E-ground (dis-)unification problem. This is instrumental to port trigger- and conflict-based instantiation to be fully applied in λfHOL.

1 Introduction

Higher-order (HO) logic is a pervasive setting for reasoning about numerous real-world applications. In particular, it is widely used in proof assistants to provide trustworthy, formal, and machine-checkable proofs of theorems. A major challenge in this setting is to automate proofs as much as possible, thereby making the proof assistants easier to use. A successful approach to this automation challenge is hammering [10]. It consists in encoding proof obligations into first-order (FO) logic and use first-order provers to discharge them. A similar layered approach is also used by HO theorem provers such as Leo-III [21] and Satallax [12], which regularly invoke FO provers to discharge intermediate goals that depend solely on FO reasoning.

Such approaches have well-known performance, soundness, or completeness issues due to the black-box integration between FO and HO reasoning [8, 16, 23]. To mitigate these problems, recent work [4, 6, 7, 9, 23] has focused on extending FO provers, based on the superposition calculus [1, 17] or on SMT solving [5], to natively support HOL, so that the integration between the highly efficient FO component and the new HO one is graceful, i.e. the prover behaves mostly as its first-order counterpart on FO problems and also handles HO problems natively. The HO extension of SMT solvers [4] has not explored lifting quantifier instantiation algorithms

*The order of authors is inverse alphabetic.

†The work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka).

to perform higher-order unification. As a consequence, widely used instantiation techniques, such as trigger-based [13, 14], conflict-based [3, 19], model-based [15, 20], and enumerative [18] instantiation, can only be applied in a limited manner. Moreover, even in a limited application, the efficient implementation of these techniques depends on specific term indices that have to be adapted to handle equalities between functions and a curried representation of terms [4, Section 3.4].

Congruence closure with free variables (CCFV) [3] is a framework for casting instantiation techniques in SMT. It is based on solving the E-ground (dis)unification problem, i.e. given a conjunctive set of ground equality literals E and a conjunctive set of (possibly non-ground) equality literals L, it finds substitutions σ such that $E \models L\sigma$ holds. These substitutions lead to instantiations following the instantiation technique being used. By extending CCFV to perform HO unification effectively we can in one fell swoop remove the limitations for HOL of all the instantiation techniques supported by the framework. This can be specially helpful for techniques that heavily depend on unification (as opposed to matching), such as conflict-based instantiation, which has been shown to be particularly effective for proof obligations originating from proof assistants [19].

Given the complexity of the task at hand, we follow previous approaches by proceeding in a stepwise manner [7, 23], first extending CCFV to λ-free HOL (λfHOL), a fragment of HOL with partial applications and functional variables. In this effort we quickly realized that extending CCFV as a tableaux-like calculus, as it was originally presented [3], posed a strong limitation on our ability to add new features to the framework. Moreover, the added complications of handling functional variables and a curried representation of terms, let alone lambda terms in the future, indicated that to have a scalable implementation we should handle the combinatorial nature of the search more efficiently than via regular backtracking. We thus present a lifting of CCFV to λfHOL via an encoding into an equisatisfiable SAT problem (Section 4). The encoding is based on fully reducing the search for substitutions to SAT based on the entailment conditions of literals in L (Section 4.1), after they have been preprocessed (Section 3), while taking into account the dependencies between variables due to cycles (Section 4.2) and congruence (Section 4.3). We present a solution reconstruction procedure (Section 5) so that satisfiable assignments lead to substitutions solving the original problem.

2 Preliminaries and problem statement

We work in λ-free higher-order logic (λfHOL) with Henkin semantics, following Bentkamp et al. [7]. We introduce here the relevant notions of this logic.

We use a monomorphic type system equipped with a set of atomic types $\mathcal{S}$ and a binary function $\rightarrow$ such that given two types τ, ν, the type $\tau \rightarrow \nu$ is the type of functions from τ to ν. The sets Σ and $\mathcal{V}$ respectively contain the function symbols (a, b, f, g...) and variables (w, x, y, z) upon which terms are built. Each symbol and variable is annotated with a type, e.g. $f : \tau \rightarrow \tau$, but the types will be omitted when irrelevant or obvious, i.e. almost all the time except in the following definition. Terms are defined as: $u = a \mid x \mid (u_1 : \tau \rightarrow \nu)\,(u_2 : \tau)$ where $a \in \Sigma$, $x \in \mathcal{V}$ and u_1, u_2 are terms. Note that this entails a curried representation of terms. A term is *ground* if it does not contain variables. We always denote terms using u plus various sub- and superscripts and ground terms using t instead. Subterms of u are u, u_1, u_2 and all subterms of u_1 and u_2 (if they occur). *Strict* subterms are all subterms excluding u itself. The notations $u[u']$ and $u[u']_s$ respectively denote that u' is a subterm or a strict subterm of u.

Literals are equalities ($u_1 \simeq u_2$) and disequalities ($u_1 \not\simeq u_2$) of terms, respectively denoted *positive* and *negative* literals. It is possible to handle predicate literals, e.g. Q and $\neg Q$, implicitly as equations and disequations, e.g. $Q \simeq \top$ and $Q \not\simeq \top$, by defining a binary type for Booleans and an interpreted symbol $\top$ of this type. This is used in practice to apply the techniques presented here to predicates but it does not impact the theory so we do not mention it further in this paper. Sets of literals are denoted by L, L', etc., and sets of ground literals by E. The set of all terms, subterms included, that occur in a set of literals L is denoted by $\mathbf{T}(L)$. The set of all *ground* terms in L is denoted by $\mathbf{T}^g(L)$.

Substitutions, denoted σ, are functions that map variables to terms such that only finitely many of them are not mapped to themselves. They are extended as usual to apply on terms and sets of terms, written in postfix notation. The domain of σ is $\mathit{dom}(\sigma) = \{x \mid x \in \mathcal{V} \text{ and } x\sigma \neq x\}$ and its range is $\mathit{ran}(\sigma) = \{x\sigma \mid x \in \mathit{dom}(\sigma)\}$. A substitution is *ground* if its range is a set of ground terms. It is *acyclic* if each variable x never occurs as a subterm of $x\sigma^n$ for any $n > 0$. The fixpoint of an acyclic substitution always exists and is denoted $\sigma^\star$.

An interpretation $\mathcal{I}$ assigns: 1. to each atomic type τ a non-empty set $\mathcal{I}(\tau)$; 2. to each type $\tau \to \nu$ a subset $\mathcal{I}(\tau \to \nu)$ of the function space from $\mathcal{I}(\tau)$ to $\mathcal{I}(\nu)$; 3. to each function or variable $u : \tau$ in $\Sigma \cup \mathcal{V}$ an element $\mathcal{I}(u)$ in $\mathcal{I}(\tau)$. $\mathcal{I}$ naturally extends to an interpretation $\mathcal{I}(u)$ for each term u. An equation $u_1 \simeq u_2$ is entailed by an interpretation $\mathcal{I}$ if and only if $\mathcal{I}(u_1) = \mathcal{I}(u_2)$, and a disequation $u_1 \not\simeq u_2$ is entailed by $\mathcal{I}$ if and only if $\mathcal{I}(u_1) \neq \mathcal{I}(u_2)$. An interpretation $\mathcal{I}$ is a model of a set of literals L if it entails all of them. This is denoted by $\mathcal{I} \models L$. By extension, we write $E \models L$ when every model of E is also a model of L.[1]

Given a set of ground literals E, the *congruence closure* of E is the partition into classes of all ground terms such that two ground terms t_1 and t_2 belong to the same class if and only if $E \models t_1 \simeq t_2$. Given additionally a set of literals L, the restriction of the congruence closure to $\mathbf{T}^g(E \cup L)$ is denoted E^{CC}. The notation $[t]$ denotes the E-congruence class in which t occurs.[2] The *representative* of $[t]$ is a chosen element in the class. The notation $[t] \in E^{\mathrm{CC}}$ denotes that $E \models t \simeq t'$ for some ground term $t' \in \mathbf{T}^g(E \cup L)$. Notice that, if $[t] \in E^{\mathrm{CC}}$, then $[t'] \in E^{\mathrm{CC}}$ for any subterm t' of t. We abuse this notation by writing $[u] \in E^{\mathrm{CC}}$ and $u \in [t]$ for non-ground terms u to indicate that we want u, or rather $u\sigma$ for some grounding substitution σ, to belong to an E-equivalence class that exists in E^{CC} or to a particular class $[t] \in E^{\mathrm{CC}}$ respectively. We denote by $[\![t]\!]$ the set of *signatures* in $[t]$, that is, all the pairs of classes $[t_1][t_2]$ such that $[t_1\, t_2] = [t]$.

Example 1. Let $E = \{a \simeq f\,a, g \simeq f, g\,b \simeq h\,c\}$ and $L = \{x \simeq y\,d\}$, then

$$E^{\mathrm{CC}} = \{\{a, f\,a\}, \{f, g\}, \{g\,b, h\,c\}, \{b\}, \{c\}, \{d\}, \{h\}\}.$$

In the full congruence closure of E, the class $[a]$ is infinite since it includes all terms of the form $f^n\,a$ and $g^n\,a$ for $n \geq 0$, among others. Entailment also goes beyond E^{CC}, e.g. $E \models f\,b \simeq h\,c$ and $E \models a \simeq f\,(g\,a)$. Moreover $[\![a]\!] = \{[f][a]\}$, $[\![g\,b]\!] = \{[f][b], [h][c]\}$ and the other signatures are empty.

Following Barbosa et al. [3], we present below the definition of the E-ground (dis)unification problem in λfHOL and the theorem characterizing its set of solutions. Although both statements coincide with their first-order logic (FOL) counterparts, in λfHOL the problem and its solutions, if they exist, may include functional variables, which are now part of the set of terms over which substitutions range. Nevertheless, the lifting to λfHOL is straightforward since it can be directly encoded into FOL (e.g. by means of an applicative encoding).

[1]This is a simplification of the actual formalism by Bentkamp et al. [7].

[2]Note that $[t]$ alone refers to the class of t, while $u[t]$ refers to a term u with a subterm t.

Definition 2 (E-ground (dis)unification)**.** Given two finite sets of equational literals E and L, where E is ground, the E-ground (dis)unification problem is that of finding substitutions σ such that $E \models L\sigma$.

Theorem 3. Given an E-ground (dis)unification problem, if a substitution σ exists such that $E \models L\sigma$, then there is an acyclic substitution σ' such that $\mathit{ran}(\sigma') \subseteq \mathbf{T}(E \cup L)$, $\sigma'^{\star}$ is ground, and $E \models L\sigma'^{\star}$.

Proof. Let *app* denote the encoding of terms from λfHOL to FOL.[3] Let us assume that there exists a σ such that $E \models L\sigma$. Then $\mathit{app}(E) \models \mathit{app}(L\sigma) = \mathit{app}(L)\mathit{app}(\sigma)$. The theorem we want to prove holds in first-order logic (see Theorem 1 in [3]) thus there exists an acyclic substitution σ' such than $\mathit{ran}(\sigma') \subseteq \mathbf{T}^{\mathit{app}}(\mathit{app}(E) \cup \mathit{app}(L))$, $\sigma'^{\star}$ is ground, and $\mathit{app}(E) \models \mathit{app}(L)\sigma'^{\star}$. Then the substitution σ'', where $\mathit{app}(\sigma'') = \sigma'$ is also acyclic and such that $\mathit{ran}(\sigma'') \subseteq \mathbf{T}(E \cup L)$ and $E \models L\sigma''^{\star}$. □

3 Preprocessing

To ease the SAT encoding (Section 4) we assume that a series of standard preprocessing techniques, shown below, have been applied to L so that it comprises only literals of the form $x \not\simeq y$, $x \not\simeq t$ or $u_0 \simeq u_1\, u_2$, where at least one of u_1 and u_2 is a variable.

Normalizing**.** A set of literals is *E-normalized*, abbreviated as *normalized* because E is always clear from the context, if every ground term it contains is the representative of its congruence class modulo E. A set of literals can be normalized by replacing all occurrences of ground terms by their representative.

Example 4. Given the problem:

$$E = \{(f\, a)\, b \simeq (f\, b)\, a,\ g\, b \simeq g\, c,\ h_1 \simeq h_2,\ g \simeq f\, a\}$$
$$L = \{h_1\, x \simeq b,\ x \simeq (f\, a)\, y,\ h_1\, ((f\, x)\, b) \simeq a,\ g\, b \simeq (f\, x)\, y,\ (f\, a)\, a \simeq g\, b\}$$

The non-singleton classes in E^{CC} are:

$$[g] = \{\boldsymbol{g},\ f\, a\}$$
$$[(f\, a)\, b] = \{g\, b,\ \boldsymbol{g\, c}, (f\, a)\, b,\ (f\, b)\, a\}$$
$$[h_1] = \{h_1,\ \boldsymbol{h_2}\}$$

where bold font $\boldsymbol{t}$ identifies the representative term in a class $[t]$. The normalized L is thus:

$$L_{\mathrm{norm}} = \{\boldsymbol{h_2}\, x \simeq b,\ x \simeq \boldsymbol{g}\, y,\ \boldsymbol{h_2}\, ((f\, x)\, b) \simeq a,\ \boldsymbol{g\, c} \simeq (f\, x)\, y,\ \boldsymbol{g}\, a \simeq \boldsymbol{g\, c}\}.$$

Removing ground literals**.** Eliminating ground literals from L amounts to replace them by $\top$ if they are entailed by E and by $\bot$ otherwise, followed by removing all occurrences of $\top$ from L. If there is any occurrence of $\bot$ then L itself becomes $\{\bot\}$ since the E-ground (dis)unification problem is then unsatisfiable.

Example 5. Consider E and L as in the previous example. Removing ground literals from L_{norm} produces $L_{\text{non-ground}} = \{\bot\}$ because its last equation is not entailed by E. If we consider instead $L' = L \backslash \{(f\, a)\, a \simeq g\, b\}$ then $L'_{\mathrm{norm}} = L_{\mathrm{norm}} \backslash \{g\, a \simeq g\, c\}$ and finally $L'_{\text{non-ground}} = L'_{\mathrm{norm}}$.

[3]WiP note: The applicative encoding allows to encode λfHOL with Henkin semantics to FOL with standard semantics. In future work, we plan to replace this proof with one that does not rely on the applicative encoding, we only give a minimal description of it in Appendix A. It is also described, e.g., by Barbosa et al. [4].

Flattening. A set of literals is *flattened* if each of its literals is of the form $x \simeq t$, $x \simeq y$, $x \not\simeq t$, $x \not\simeq y$, or $u_0 \simeq u_1 u_2$ where x and y are variables, t is a ground term, and u_0, u_1, u_2 are either variables or ground terms with at least u_1 or u_2 being a variable. Any set of literals can be flattened by introducing new variables.

Example 6. Consider L' as in the previous example. The flattened version of $L'_{\text{non-ground}}$ is:

$$L'_{\text{flat}} = \{b \simeq h_2\, x,\ x \simeq g\, y,\ a \simeq h_2\, z_1,\ z_1 \simeq z_2\, b,\ z_2 \simeq f\, x,\ g\, c \simeq z_2\, y\}.$$

Trivial assignments. A set of literals L has trivial assignments if it contains literals of the following form:

- $x \simeq t$, where x is a variable and t is ground;
- $x \simeq y$, where x, y are variables;
- $x \simeq u$, where variable x does not occur elsewhere in L including in u, and u is any term (not containing x).

As long as there is a trivial assignment $x \simeq u'$ in L it is possible to consider the equivalent problem where L is replaced by $L' = (L \setminus \{x \simeq u'\})\sigma$ instead, where $\sigma = \{x \mapsto u'\}$. In the third case, this simply amounts to removing the $x \simeq u$ equation from L.

Since L' may contain new ground literals and trivial assignments, as well as ground terms that are not in normal form, it is necessary to iterate on normalization, ground literals simplification and trivial assignments instantiation until all such literals have been removed. Eliminating trivial assignments might render a literal ground, but otherwise flattening is not impacted by this transformation. Indeed, none of the three cases of trivial assigments will ever lead to the replacement of a variable inside a literal by an applied non-ground term. This process terminates since each step eliminates one variable from L among finitely many. Sect. 5 provides a way to build a solution for L from a solution for L'.

Example 7. L'_{flat} from the previous example contains no trivial assignments.

Example 8. Let $L = \{x \simeq f\, a,\ y \simeq x\, b,\ z \simeq y\, z\}$. Assume that L is already normalized for a given E. Note that L is also flattened and without ground literals. However, it contains the trivial assignment $x \simeq f\, a$ since $f\, a$ is ground. Applying the previously described procedure yields $L' = \{y \simeq (f\, a)\, b,\ z \simeq y\, z\}$. This new set is still flattened and without ground literals but it contains a new trivial assignment, namely $y \simeq (f\, a)\, b$. Assume $(f\, a)\, b$ is normalized. After another iteration of the procedure, the remaining problem is $L'' = \{z \simeq ((f\, a)\, b)\, z\}$.

Example 9. Let $L = \{y \simeq x\, a,\ z \simeq x,\ z \simeq f\, x,\ g \simeq x\, c\}$. As in the previous example, L is flattened and without ground literals and we assume it is normalized for a given E. Both literals $y \simeq x\, a$ and $z \simeq x$ are trivial assignments and are to be eliminated. The preprocessing chain yields $L' = \{x \simeq f\, x,\ g \simeq x\, c\}$.

A flattened normalized set of literals without trivial assignments and ground literals is called a *preprocessed* set of literals.

4 Encoding CCFV as a SAT problem

Every solution of the E-ground (dis)unification problem is a substitution: it maps variables to terms. Thanks to Theorem 3 we know that if a generic solution exists, then a ground one can always be found. Thus it is enough to search for ground solutions to answer the general problem. Such a ground solution associates each variable to a ground term belonging to a particular E-congruence class, in or out of E^{CC}. By considering all classes from E^{CC} and merging all the classes outside of E^{CC} together, the association between variables and classes turns into a combinatorial problem with finitely many possibilities. Barbosa et al. [3] presented CCFV, a decision procedure for E-ground (dis)unification, as a tableaux-like procedure that decomposes L in a top-down manner so that the possibilities for mapping variables are increasingly reduced. The decompositions are based on the entailment conditions for the literals in L, according to their structure. However, considering the entailment conditions in the presence of functional variables and a curried representation of terms, particularly when we move beyond λfHOL and need to take higher-order unification into account, opens up more possibilities that cannot be handled in such a high-level procedure without a severe loss of efficiency or an extremely intricate design. As an alternative solution aiming for greater flexibility and better performance, here we tackle the combinatorial problem by, from the get-go, fully decomposing each literal in L as a propositional disjunction over all the conditions for its entailment, relying on a SAT solver to determine an assignment that respects all conditions (Section 4.1). Moreover, by reducing the problem to SAT we also need to encode properties that were handled silently at the first-order level by term indexing and by the underlying ground congruence closure component, namely cyclic dependencies between variables (Section 4.2) and variables assigned by derived congruence reasoning (Section 4.3).

4.1 Encoding's core

We assume L is preprocessed. The set L thus only contains literals of the form $x \not\simeq u$ and $u_0 \simeq u_1 u_2$ where x is a variable and u, u_0, u_1, u_2 are variables or ground terms, at least one of u_1 and u_2 being a variable. The problem is encoded into a set $\mathcal{C}$ of formulas, which is the union of all $\mathcal{C}_\ell$ for $\ell \in L$.

$$\begin{array}{rl}\mathcal{C}_{x\not\simeq u}: & \bigvee_{[t_1],[t_2]\in E^{\mathrm{cc}}, E\models t_1\not\simeq t_2}(x \in [t_1] \wedge u \in [t_2]) \\ \mathcal{C}_{u_0\simeq u_1 u_2}: & [u_0] \notin E^{\mathrm{CC}} \vee \bigvee_{[t_0]\in E^{\mathrm{cc}},\ [t_1][t_2]\in [\![t_0]\!]}(u_0 \in [t_0] \wedge u_1 \in [t_1] \wedge u_2 \in [t_2])\end{array}$$

The intuition behind this encoding is as follows. Assume there exists a solution σ to the E-ground (dis)unification problem. For negative literals, $E \models (x \not\simeq u)\sigma$ holds only if there exist t_1, t_2 such that $E \models t_1 \not\simeq t_2$ and $x\sigma \in [t_1]$, $u\sigma \in [t_2]$. Moreover, for $E \models t_1 \not\simeq t_2$ to hold it must be the case that $[t_1], [t_2] \in E^{\mathrm{CC}}$. For positive literals, $E \models (u_0 \simeq u_1\, u_2)\sigma$ holds if $u_0\sigma$ and $(u_1\, u_2)\sigma$ are in the same congruence class. If $[u_0\sigma] \in E^{\mathrm{CC}}$, we know exactly which kind of applied term can belong to this class: any term $t_1\, t_2$ such that $[t_1][t_2] \in [\![u_0\sigma]\!]$. Thus in that case it is enough to consider all possible signatures of the class that u_0 is assigned to.[4]

All membership tests where u or u_i is not a variable are simplified in the above formulas (to true or false). In the literals that remain, u and u_i are always variables and literals are of the form $x \in [t]$ or $[x] \notin E^{\mathrm{CC}}$ for some variable x and some class $[t] \in E^{\mathrm{CC}}$. A new propositional variable $P_{x,[t]}$ is introduced for each literal $x \in [t]$ occurring in any $\mathcal{C}_\ell$. Another propositional variable Q_x is introduced for each literal $x \notin E^{\mathrm{CC}}$ occurring in any $\mathcal{C}_\ell$.

Note that since a variable cannot be mapped to two distinct classes at the same time, we also encode that the $P_{x,[t]}$ that occur in the encoding are mutually exclusive between themselves and with Q_x, for each $x \in \mathcal{V}$.

[4] WiP note: These informal statements will be formalized as a lifting to λfHOL of Theorem 4.4 in [2], which captures the conditions in FOL for solving the E-ground (dis)unification problem for a given literal in L based on its structure.

As illustrated by the following example, the above encoding is not enough to represent the problem in the general case.

Example 10. Let $E = \{a \simeq f\,b, a \simeq b, f \not\simeq f', k \simeq h\,f\}$ and $L = \{x \simeq y\,x, z \simeq g\,y, z \simeq v\,f, y \not\simeq y'\}$ where x, y, y', z, and v are variables. The only non-singleton classes in E^{CC} are $[a] = \{a, b, f\,b\}$ and $[k] = \{k, h\,f\}$. The non-empty signatures are $[\![a]\!] = \{[f][a]\}$ and $[\![k]\!] = \{[h][f]\}$ The set L is already preprocessed. Then:

$$\begin{array}{rcl}\mathcal{C}_{x\simeq y\,x} & = & Q_x \vee (P_{x,[a]} \wedge P_{y,[f]}) \\ \mathcal{C}_{z\simeq g\,y} & = & Q_z \\ \mathcal{C}_{z\simeq v\,f} & = & Q_z \vee (P_{z,[k]} \wedge P_{v,[h]}) \\ \mathcal{C}_{y\not\simeq y'} & = & (P_{y,[f]} \wedge P_{y',[f']}) \vee (P_{y,[f']} \wedge P_{y',[f]})\end{array}$$

and the relevant mutual exclusion constraints are

$$\begin{array}{l}\neg P_{y,[f]} \vee \neg P_{y,[f']} \\ \neg P_{y',[f]} \vee \neg P_{y',[f']} \\ \neg Q_x \vee \neg P_{x,[a]} \\ \neg Q_z \vee \neg P_{z,[k]}\end{array}$$

Note that in $\mathcal{C}_{x\simeq y\,x}$ the disjunct $P_{x,[k]} \wedge P_{y,[h]} \wedge P_{x,[f]}$ does not occur, because it assigns x to two distinct classes. Since this trivially contradicts the mutual exclusion constraints, this disjunct can never be true.[5] Another noteworthy point is that $\mathcal{C}_{z\simeq v\,f}$ is redundant to $\mathcal{C}_{z\simeq g\,y}$. This could also be detected during the encoding and simplified. It would prevent the addition of the useless mutual exclusion constraint on z-related literals.

Note that one model of the obtained formula is $\{Q_x, Q_z, P_{y,[f]}, P_{y',[f']}\}$, which implies in particular that $[x] \notin E^{\mathrm{CC}}$. However, all the solutions to the current problem must map x to $[a] \in E^{\mathrm{CC}}$ because it is the only class that contain both a term and its image by f.

In fact, this issue happens for a whole family of variables, that we denote as cyclic variables. An extra constraint is required to handle them separately.

4.2 Cycle-based constraints

Formally, a variable is *cyclic* in a set of terms L if $L \models x \simeq u[x]$. The cycle can be directly apparent in an equation in L as in the previous example, but it can also be less obvious, as in $L = \{x \simeq y\,a, y \simeq g\,x\}$ where both x and y are cyclic. Cyclic variables in an E-ground (dis)unification problem must all be mapped to terms in E^{CC}. This is a consequence of Lemma 11. Its proof is available in Appendix B.

Lemma 11. Let E be a set of ground equational terms. Let t and $t'[t]_s$ be ground terms. If $E \models t \simeq t'[t]_s$ then $[t] \in E^{\mathrm{CC}}$.

Corollary 12. Given an E-ground (dis)unification problem, if x has a cyclic definition in L then $[x] \in E^{\mathrm{CC}}$.

Proof. If $L \models x \simeq u[x]_s$ and $E \models L\sigma$ for some ground σ then $E \models x\sigma \simeq (u[x]_s)\sigma$. By Lemma 11, $[x\sigma] \in E^{\mathrm{CC}}$. □

[5] Even if this was not the case, there would still be a typing incompatibility in allowing x to be mapped to $[a]$ and to $[f]$ in different disjuncts since a and f must have distinct types.

$$\begin{tikzcd} x \arrow[loop left] \arrow[r] & y & z \arrow[l] \arrow[r] & v & y' \end{tikzcd}$$

Figure 1: Dependency graph of L from Example 10

Example 13. In Example 10, only x is cyclic. It is thus enough to add the unit clause $\neg Q_x$ to the encoding to restrict models to those that map x to a class in E^{CC}.

The dependency between variables can be encoded as a directed graph where a vertice represents a variable and an edge from the vertex x to the vertex y indicates the presence of an equation $x \simeq u_1\, u_2$ in L where y is either of u_1 and u_2. Figure 4.2 represents this graph for the set of literals L from Example 10.

Thus it is possible to use any algorithm that enumerates the cycles in a graph to find all cyclic variables in L. For our prototype implementation, we used a naive algorithm described in Appendix C but linear algorithms are known, e.g. Tarjan's strongly connected components algorithm [22]. Once the cycles are detected, the additional constraints $\neg Q_i$ must be added to the encoding for each cyclic variable as well as for all variables reachable from cyclic variables in the dependency graph.[6] The later is required because all subterms of a term in E^{CC} must also be in E^{CC}.

The core encoding and the cycle-based constraints ensure that all the variables that must be mapped to terms in E^{CC} indeed are. However, the variables that can be mapped outside of E^{CC} may also have extra constraints that are not captured by what has been described so far.

Example 14. Consider Example 10, with the addition of $\neg Q_x$ to the encoding. A model of the resulting formula is $\{P_{x,[a]}, P_{y,[f]}, P_{y',[f']}, Q_z\}$. This does not tell us anything about v, which means that this variable could be mapped to any (type-compatible) class. However, mapping v to $[h]$ does not lead to a valid solution. This would force z to be equal both to $g\, f$ and to $h\, f$, although $[g\, f] \neq [h\, f]$, which is impossible.

Further constraints are required on the variables that can be mapped outside of E^{CC}. We denote those variables as floaters.

4.3 Floater-based constraints

For a variable x, being a *floater* means it occurs on the left-hand side of an equation in L and that $[x] \notin E^{\text{CC}}$. It follows from the latter that any equation $x \simeq u_1\, u_2 \in L$ where x is a floater must either be a tautology or hold by congruence. In both cases, it means that if another equation of the same form $x \simeq u'_1\, u'_2$ also occurs in L, necessarily $[u_1] = [u'_1]$ and $[u_2] = [u'_2]$. Of course, before running the SAT solver, we don't know exactly which variables are floaters, so we have to encode these constraints conditionally. If there are n equalities $x \simeq u_{1j}\, u_{2j}$ in L where $j \in \{1..n\}$, x is not cyclic nor occurs in a disequation, and if $n > 1$, the previously described constraints can be expressed as:

$$Q_x \Rightarrow (Q_{u_{11}} \equiv \cdots \equiv Q_{u_{1n}}) \quad \text{and} \quad Q_x \Rightarrow (Q_{u_{21}} \equiv \cdots \equiv Q_{u_{2n}});$$
$$\text{for } [t] \in E^{\text{CC}},\ Q_x \Rightarrow (P_{u_{11},[t]} \equiv \cdots \equiv P_{u_{1n},[t]}) \quad \text{and} \quad Q_x \Rightarrow (P_{u_{21},[t]} \equiv \cdots \equiv P_{u_{2n},[t]}).$$

Note that as soon as one of the u_{ij} is ground for $i \in \{1, 2\}$ and $j \in \{1..n\}$, the corresponding $Q_{u_{ij}}$ is false and the truth values of the $P_{u_{ij},[t]}$s are also known beforehand for all $[t] \in E^{\text{CC}}$, which simplifies and significantly narrows the constraints.

[6]Alternatively, all occurrences of Q_i for relevant is are set to false and the formula is simplified accordingly.

Example 15. Consider again Example 10, the only floater-based constraints to add originate from the two equations involving z: $z \simeq g\, y$ and $z \simeq v\, f$. In their simplified form, because f and g are ground, they are:

$$Q_z \Rightarrow \neg Q_v, \qquad Q_z \Rightarrow P_{v,[g]}, \qquad Q_z \Rightarrow \neg P_{v,[h]},$$

$$Q_z \Rightarrow \neg Q_y, \qquad Q_z \Rightarrow P_{y,[f]}, \qquad Q_z \Rightarrow \neg P_{y,[f']}$$

Thanks to the addition of these constraints, the only model of the encoding is now $\mathcal{M} = \{P_{x,[a]}, P_{y,[f]}, P_{y',[f']}, Q_z, P_{v,[g]}\}$. The substitution $\sigma = \{x \mapsto a, y \mapsto f, y' \mapsto f', z \mapsto g\, f, v \mapsto g\}$ is a solution of the problem. It satisfies all the constraints in $\mathcal{M}$.

We conjecture that our encoding is sound and complete. The proof of this central result is still work in progress.

Conjecture 16. Given two sets of equational literals E and L such that E is ground, the E-ground (dis)unification problem has a solution if and only if the encoding of this problem is satisfiable.

5 Reconstruction of a solution from a SAT model

Let E and L form an E-ground (dis)unification problem. Let L_{pre} be a preprocessed version of L and L_{triv} be the set of trivial assignments that were removed from L during the preprocessing. Assume that the encoding of the problem is satisfiable and let $\mathcal{M}$ be a model of this encoding. Let t^r denote the representative of $[t]$.

To build a ground substitution σ such that $E \models L\sigma$, we proceed iteratively, starting with the variables in E^{CC} and those that do not depend on any other variables, then going backward through the variable dependency graph of $L_{\text{pre}} \cup L_{\text{triv}}$, until all variables are mapped to ground terms.

In more details:

- $\sigma_0 = \{x \mapsto t^r \mid \mathcal{M} \models P_{x,[t]}\}$, where x occurs in L_{pre}.
- $\sigma_1 = \sigma_0 \circ \sigma'$ such that, for any variable x not grounded by σ_0 and that only occurs on the right-hand side of equations in $L_{\text{pre}} \cup L_{\text{triv}}$, σ' maps $x : \tau$ to ${t_\tau}^r$ where t_τ denotes a default term of the appropriate type.
- The mapping of all other variables must be built iteratively. For $i > 1$ let $\mathcal{I}_i = \{(x, t) \mid x \simeq t \in (L_{\text{pre}} \cup L_{\text{triv}})\sigma_{i-1}\}$ and let $\sigma_i = \sigma_{i-1} \circ \{x \mapsto t^r \mid (x,t) \in \mathcal{I}_i\}$. The process terminates as soon as all variables have been assigned and the final result is σ.

Example 17. For the problem in Example 10, considering the model $\mathcal{M}$ given in Example 15, $\sigma_0 = \{x \mapsto a, y \mapsto f, y' \mapsto f', v \mapsto g\}$ is determined looking at all the $P_{x,[t]}$ in $\mathcal{M}$. Then $\sigma_1 = \sigma_0$ because there are no variables matching the criterion. Finally, one iteration of the last step is enough to obtain $\sigma = \sigma_0 \circ \{z \mapsto f\, g\}$ that is a ground substitution such that $E \models L\sigma$.

Note that the solution obtained may not satisfy all the $[x] \notin E^{\text{CC}}$ constraints. They will only be satisfied if the variables mapped in the second step are assigned default values outside E^{CC}. In practice, in the context of SMT, we may want to do the exact inverse and map as many of these variables as possible inside of E^{CC}.

Another noteworthy point concerning this second step is that there must be only one unique default value for each type, or at least one default class. To illustrate this, consider a problem where $x \simeq f\, y, x \simeq f\, z \in L_{\text{pre}}$, but neither of the three variables occur anywhere else. If y and

z are mapped to two terms that do not belong to the same E-congruence class, the resulting substitution will not be a solution of the problem.

Conjecture 18. Let E and L form an E-ground (dis)unification problem that admits a solution. Then the substitution σ constructed following the reconstruction method described in this section is a ground solution of the problem.

6 Ongoing and future work

A first-order version of CCFV with support for λfHO terms encoded with the applicative encoding [4] is already implemented in the λfHOL extension of the lightweight SMT solver veriT [11]. It has a term-indexing issue because terms are indexed by the symbol at their head and the applicative symbol occurs at the head of all functional symbols. Thus, contrarily to what happens in FOL where the head symbol significantly restricts the mapping possibilities, here only types can be exploited to retrieve suitable terms, which can lead to many more terms being retrieved at once in comparison with the first-order case. The present approach does not have this problem because it does not rely on the applicative encoding at all.

A prototype is under development in veriT. However, model reconstruction is not yet operational. As soon as it is, we will be able to compare our approach with the applicative version of CCFV. Given that the later, used by trigger- and conflict-based instantiation, currently takes around 40% of the overall SMT computation time in the benchmarks used by Barbosa et al. [4], we expect that any improvement on the efficiency of CCFV will lead to an important speedup in the SMT process, but it remains to be observed in practice if this is the case for our approach.

Experiments notwithstanding, there is much that can be improved in our prototype regarding the data-structures as well as the algorithms for the preprocessing and the encoding phases. In particular, cycle detection is currently implemented with a naive algorithm in $O(n^3)$. It will eventually be replaced by Tarjan's linear algorithm. Furthermore, the encoding itself could be improved. Each constraint $\mathcal{C}_{x \not\simeq u}$ and $\mathcal{C}_{u_0 \simeq u_1 u_2}$ imposes that variables belong to some classes among all, and indirectly that they do not belong to the remaining classes. This information could be propagated to other constraints and lead to further narrowing of the set of possible classes for some variables. By starting with the most restrictive constraints, one might be able to significantly restrict the size of the constraints, the number of propositional variables, and also the number of necessary mutual exclusion constraints. An incremental encoding to SAT, as well as a heuristic to guide the SAT solver so that it starts its search on the most restricted variables are also on our list of ideas to be investigated.

7 Acknowledgments

We are grateful to Jasmin Blanchette for many discussions throughout the development of this work, for providing funding for research visits and for suggesting many improvements. Experiments were carried out using the Grid'5000 testbed (https://www.grid5000.fr/), supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities as well as other organizations.

References

[1] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[2] H. Barbosa. *New techniques for instantiation and proof production in SMT solving*. PhD thesis, Université de Lorraine, Universidade Federal do Rio Grande do Norte, 2017.

[3] H. Barbosa, P. Fontaine, and A. Reynolds. Congruence closure with free variables. In A. Legay and T. Margaria, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part II*, volume 10206 of *Lecture Notes in Computer Science*, pages 214–230, 2017.

[4] H. Barbosa, A. Reynolds, D. E. Ouraoui, C. Tinelli, and C. W. Barrett. Extending SMT solvers to higher-order logic. In P. Fontaine, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 11716 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2019.

[5] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *FAIA*, chapter 26, pages 825–885. IOS Press, 2009.

[6] A. Bentkamp, J. Blanchette, S. Tourret, P. Vukmirović, , and U. Waldmann. Superposition with lambdas. In P. Fontaine, editor, *Proc. Conference on Automated Deduction (CADE)*, Lecture Notes in Computer Science. (Accepted for publication). Springer, 2019.

[7] A. Bentkamp, J. C. Blanchette, S. Cruanes, and U. Waldmann. Superposition for lambda-free higher-order logic. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Int. Joint Conference on Automated Reasoning (IJCAR)*, volume 10900 of *Lecture Notes in Computer Science*, pages 28–46. Springer, 2018.

[8] A. Bhayat and G. Reger. Set of support for higher-order reasoning. In B. Konev, J. Urban, and P. Rümmer, editors, *Practical Aspects of Automated Reasoning (PAAR)*, volume 2162 of *CEUR Workshop Proceedings*, pages 2–16. CEUR-WS.org, 2018.

[9] A. Bhayat and G. Reger. Restricted combinatory unification. In P. Fontaine, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 11716 of *Lecture Notes in Computer Science*, pages 74–93. Springer, 2019.

[10] J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.

[11] T. Bouton, D. C. B. de Oliveira, D. Déharbe, and P. Fontaine. veriT: an open, trustable and efficient SMT-solver. In R. A. Schmidt, editor, *CADE–22*, volume 5663 of *LNCS*, pages 151–156. Springer, 2009.

[12] C. E. Brown. Satallax: an automatic higher-order prover. In B. Gramlich, D. Miller, and U. Sattler, editors, *Int. Joint Conference on Automated Reasoning (IJCAR)*, volume 7364 of *LNCS*, pages 111–117. Springer, 2012.

[13] L. de Moura and N. Bjørner. Efficient e-matching for SMT solvers. In F. Pfenning, editor, *CADE–21*, volume 4603 of *LNCS*, pages 183–198. Springer, 2007.

[14] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52:365–473, 2005.

[15] Y. Ge and L. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In A. Bouajjani and O. Maler, editors, *CAV 2009*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.

[16] J. Meng and L. C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008.

[17] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of automated reasoning*, volume 1, pages 371–443. Elsevier Science, 2001.

[18] A. Reynolds, H. Barbosa, and P. Fontaine. Revisiting enumerative instantiation. In D. Beyer and M. Huisman, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 2018.

[19] A. Reynolds, C. Tinelli, and L. de Moura. Finding conflicting instances of quantified formulas in SMT. In *FMCAD 2014*, pages 195–202. IEEE, 2014.

[20] A. Reynolds, C. Tinelli, A. Goel, S. Krstić, M. Deters, and C. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In M. P. Bonacina, editor, *CADE–24*, volume 7898 of *LNCS*, pages 377–391. Springer, 2013.

[21] A. Steen and C. Benzmüller. The higher-order prover Leo-III. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Int. Joint Conference on Automated Reasoning (IJCAR)*, volume 10900 of *LNCS*, pages 108–116. Springer, 2018.

[22] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[23] P. Vukmirovic, J. C. Blanchette, S. Cruanes, and S. Schulz. Extending a brainiac prover to lambda-free higher-order logic. In T. Vojnar and L. Zhang, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 192–210. Springer, 2019.

A Applicative encoding

The applicative encoding turns every symbol in λfHOL of a non-atomic type $\tau_1 \to \tau_2$ into a symbol of the atomic type $\tau_{\tau_1 \to \tau_2}$ in non-curried FOL. It also introduces a family of function symbols $@ : \tau_{\tau_1 \to \tau_2} \times \tau_1 \mapsto \tau_2$ such that $\mathit{app}(u_1\, u_2) = @(u_1, u_2)$. For example $\mathit{app}(h_1\,((f\,x)\,b)) = @(h_1, @(@(f, x), b))$.

B Proof of the cycle-based narrowing lemma

Lemma. Let E be a set of ground equational terms. Let t and $t'[t]_s$ be ground terms. If $E \models t \simeq t'[t]$ then $[t] \in E^{\text{CC}}$.

Proof. The *congruence graph* induced by E on a set of terms is the smallest undirected graph such that:

- for each equality $u \simeq v$ in E, there is an edge between u and v.
- if there is a path from u_1 to v_1 and from u_2 to v_2, there is an edge from u_1u_2 to v_1v_2; the edges are said to be *congruence edges*.

Consider all possible terms in a given language and their partition induced by E: two terms are in the same class if and only if there is a path in the congruence graph from one to the other. We now consider the interpretation $\mathcal{I}$ such that:

- the domain is the set of partitions;
- each constant is interpreted as the partition it belongs to;
- the application of two elements is interpreted as the partition containing the application of members in the respective partitions.

This interpretation $\mathcal{I}$ is a model of E, and assigns two terms in different classes to different elements in the domain. Hence, $E \models u \simeq v$ only if u and v belong to the same class.

Within each class $[u] = [v]$ such that $[u] \notin E^{\text{CC}}$, all edges between two distinct members u, v of the class are congruence edges (all other edges being between elements of E^{CC}), which imposes that u is of the form u_1u_2 and v is of the form v_1v_2 with u_1 in the same class as v_1 and u_2 in the same class as v_2.

Notice that, if $E \models t \simeq t'[t]_s$, then $E \models t \simeq t'[t'[t]_s]_s$. This can be used to build a term of arbitrary depth $t''[t]_s$ equal to t according to E. In particular there exists a term $t''[t]_s$ such that t occurs in $t''[t]_s$ at a depth larger than the depth of t. Since $[t] \notin E^{\mathrm{CC}}$, no subterm of $t''[t]_s$ that contains t belong to E^{CC} either.

Now let us consider for a start $u = t$ and $v = t''[t]_s$. We have already shown that $E \models u \simeq v$ and $[v] \notin E^{\mathrm{CC}}$. Then $u = u_1 u_2$, and $v = v_1 v_2$, one v_i containing t. Then $E \models u_i \simeq v_i$ and $[v_i] \notin E^{\mathrm{CC}}$ for $i \in \{1, 2\}$. Similarly, we can continue decomposing u_i, until we obtain a constant after finitely many decompositions (t being a finite term). We reach a contradiction, because this constant is equal to an application term, and both terms are equal through congruence. □

C A naive cycle-detection algorithm

Given a preprocessed set of literals L, let us denote as *defining equations* the equations of the form $x \simeq u_1 u_2$ occurring in L, where x is a variable and u_1, u_2 are any term. Variables occurring on the left-hand side of defining equation are called *defined* variables and variables occurring on the right-hand side of a defining equation are called the *used* variables.

Let $\mathcal{D}$ be the set of all variables occurring in L that are both defined and used. For example, for $L_1 = \{x \simeq y\,z,\ x \simeq f\,z,\ y \simeq g\,x\}$ where x, y, z are variables, we obtain $\mathcal{D}_1 = \{x, y\}$, and for $L_2 = \{x \simeq y\,z,\ y \simeq g\,x,\ z \simeq f\,y,\ g \simeq w\,c\}$ where x, y, z, w are variables, we obtain $\mathcal{D}_2 = \{x, y, z\}$. If $\mathcal{D} = 0$ then there are no cyclic variables in L. Otherwise, the following procedure identifies the cyclic variables in L:

- Build an $n \times n$ matrix $\mathcal{M}_0$, where $n = |\mathcal{D}|$. Each row i of this matrix and its successors represents a defined variable x_i in $\mathcal{D}$ and each column j a used variable x_j in $\mathcal{D}$. Initialize the cells of $\mathcal{M}_0$ as:

$$\mathcal{M}_0[i,j] = \begin{cases} 1 & \text{if } x_j \text{ occurs in a defining equation } x_i = u_1\,u_2, \\ & \quad \text{i.e. } x_j = u_1 \text{ or } x_j = u_2; \\ 0 & \text{otherwise.} \end{cases}$$

- Iterate the following operations until the matrix is stable, i.e. until $\mathcal{M}_k = \mathcal{M}_{k+1}$. For all lines i of $\mathcal{M}_k$, let $J_i = \{j \mid \mathcal{M}_k[i,j] = 1\}$ and have:

$$\mathcal{M}_{k+1}[i] = \mathcal{M}_k[i] \vee \bigvee_{j \in J_i} \mathcal{M}_k[j]$$

  where the disjunctions are bit-wise operators.
- Let $\mathcal{M}$ be the stable result obtained in the previous step. The used variables that belong to cycles are all the $i \in \{0, ..n-1\}$ such that $\mathcal{M}[i,i] = 1$.

Example 19. Let us consider $L_2 = \{x \simeq y\,z, y \simeq g\,x, z \simeq f\,y, g \simeq w\,c\}$ as before. Then $\mathcal{D}_2 = \{x, y, z\}$ and:

$$\mathcal{M}_0 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathcal{M}_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad \mathcal{M}_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \mathcal{M}_3 = \mathcal{M}_2,$$

thus x, y and z are cyclic but not w.

Note that if the matrix is completely filled with 1's, it is not necessary to perform an extra iteration of the procedure to conclude that it is stable.

Example 20. Let us now consider $L_3 = \{x \simeq y\,z, y \simeq g\,w, z \simeq f\,x\}$, where x, y, z, w are variables. Then $\mathcal{D} = \{x, y, z\}$ and:

$$\mathcal{M}_0 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \mathcal{M}_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \quad \mathcal{M}_2 = \mathcal{M}_1,$$

showing that here only x and y are cyclic. Note that if a line of the matrix contains only zero's, it is possible to remove it completely from the problem, so here it is possible to compute only:

$$\mathcal{M}_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \mathcal{M}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathcal{M}_2 = \mathcal{M}_1.$$

Note that it is also possible to consider all variables when building $\mathcal{M}_0$, not only those in $\mathcal{D}$. This will only create more lines filled with 0's that need to be discarded.

Moreover, Note that not all lines in the matrices end up filled with only 1's or only 0's, as illustrated in the following extension of the previous example.

Example 21. Let $L_4 = \{x \simeq y\,z, y \simeq f\,w, z \simeq f\,x, w \simeq fw\}$, where x, y, z, w are variables. Then $\mathcal{D} = \{x, y, z, w\}$ and:

$$\mathcal{M}_0 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathcal{M}_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathcal{M}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathcal{M}_3 = \mathcal{M}_2.$$

It is not possible to use this algorithm to detect cycles before the flattening because it applies only on variables and it is possible to have a cycle on an applied term before flattening. For example in the non-flattened $L = \{g\,x \simeq (k\,w)\,(f\,x),\ f\,x \simeq h\,(g\,x)\}$, both $g\,x$ and $f\,x$ are involved in a cycle but they are terms, not variables. The only variables x and w are not involved in cycles, hence our algorithm will not find any cyclic variable. After flattening, $L_{\text{flat}} = \{y_1 \simeq g\,x,\ z \simeq k\,w,\ y_2 \simeq f\,x,\ y_1 \simeq z\,y_2\ y_2 \simeq h\,y_1\}$ and the variables y_1 and y_2 represent the terms involved in cycles. This can now be detected by the algorithm presented above.