Information and Computation 238 (2014) 128–156

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Efficient general AGH-unification

Zhiqiang Liu [*,1], Christopher Lynch [*,1]

Department of Mathematics and Computer Science, Clarkson University, Potsdam, NY 13699, USA

ARTICLE INFO

Article history:
Received 9 March 2011
Available online 16 July 2014

Keywords:
Abelian group
Homomorphism
General unification

ABSTRACT

General E-unification is an important tool in cryptographic protocol analysis, where the equational theory E represents properties of the cryptographic algorithm, and uninterpreted function symbols represent other functions. The property of a homomorphism over an Abelian group is common in encryption algorithms such as RSA. The general E-unification problem in this theory is NP-complete, and existing algorithms are highly nondeterministic. We give a mostly deterministic set of inference rules for solving general E-unification modulo a homomorphism over an Abelian group, and prove that it is sound, complete and terminating. These inference rules have been implemented in Maude, and will be incorporated into the Maude-NRL Protocol Analyzer (Maude-NPA).

1. Introduction

In symbolic cryptographic protocol analysis, messages are represented as terms. Actions of principals involved in the protocol are represented with rules, indicating that if a principal receives a message with a given pattern then the principal will send out a message based on the message received. Abilities of malicious intruders are represented by rules indicating how an intruder can manipulate data, where variables in the pattern indicate that the principal will accept any message of that type. A goal state represents an attack, and an analyzer decides whether the goal state is reachable. Generally, the analysis involves working back from the goal state to initial states. If this is possible, then an attack exists. Initial methods of cryptographic protocol analysis were based on the free algebra model [1]. In this method of analysis, two messages are the same only if they are represented by the same term. In this case, during the search back from the goal, a message pattern representing a received message will be compared against a message pattern representing a sent message. Syntactic unification is used to compare them against each other and find the intersection of the patterns.

However, the free algebra model is not precise enough to model properties of cryptographic algorithms [2]. For example, cryptographic algorithms may involve an encryption algorithm which has the property of a homomorphism over an Abelian group (AGH). RSA has the property ${m_1}^e {m_2}^e = (m_1 m_2)^e$, where raising to the power of e is a homomorphism and the product of the messages forms an Abelian group. Homomorphism with Abelian Groups is also commonly used in privacy-preserving protocols, for instance the electronic voting system [3,4]. Unfortunately, the free algebra approach fails to detect attacks in protocols using cryptographic algorithms with equational properties. Therefore, to conduct a more precise analysis, unification must be performed modulo this equational theory. We consider our paper as a first step towards the goal of developing practical E-unification algorithms that can be used to analyze such protocols.

* Corresponding authors.
E-mail addresses: liuzh@clarkson.edu (Z. Liu), clynch@clarkson.edu (C. Lynch).

1 Both authors are supported by NSF Grant CNS 09-05378.

http://dx.doi.org/10.1016/j.ic.2014.07.008
0890-5401/© 2014 Elsevier Inc. All rights reserved.

Since unification algorithms for AGH are essential for cryptographic protocol analysis, it is important that AGH-unification algorithms are efficient. Efficient unification algorithms have been developed for elementary AGH-unification [5,6]. However, cryptographic protocol analysis also must deal with uninterpreted function symbols. For example, besides the encryption function, the hash function and concatenation of functions are also commonly used for building other cryptographic primitives.

When uninterpreted function symbols occur in combination with AGH, the complete set of unifiers is not always a singleton, but it is finite. It is crucial that the unification algorithm creates a complete set of unifiers that is as small as possible. If this set is too large, the search space of searching for an attack quickly blows up and cryptographic protocol analysis becomes infeasible.

The goal then is to build an AGH-unification algorithm that is both efficient and creates a small complete set of unifiers. The problem of deciding solvability of AG-unification problems is NP-complete [7]. An extra homomorphic operator will not make the problem any simpler, since homomorphism may not appear in the problem. An extra homomorphic operator will not make the problem more complicated either by using the technique in [7], i.e. the problem of deciding solvability of AGH-unification problems is also NP-complete. The problem of counting the minimal complete set of unifiers of the AG-unification problem is in #P [8].

In [9] and [10], the authors present a more general idea of solving unification problems modulo monoidal theories, which include AG and arbitrary unitary homomorphic operators. They showed that solving unification problems is equivalent to solving systems of linear equations over a semiring which can be efficiently solved by computing the Hermite normal form of the corresponding integer matrix. However, this method is only applicable to the elementary E-unification problems with constants. It cannot solve the general E-unification problem.

However in [11] and [12], the authors introduced a general algorithm to solve general unification problems. It can solve a general AGH-unification problem by decomposing the problem into an elementary AGH-unification problem and a syntactic unification problem with free function symbols and then combining the two partial results together to get the final result. This algorithm has two impractical aspects: 1. it is a highly non-deterministic procedure which will increase the complexity of the implementation; 2. it generates a large complete set of unifiers which contains many redundant unifiers and this will blow up the search space during cryptographic protocols analysis. The reason for the highly nondeterministic nature of the combination algorithm is that equivalence classes must be created to determine which variables are equal, the variables must be labeled to put them in one of two sets, and a linear ordering must be given on the variables. It is not known which of these may work, so all equivalence relations, all labellings and all linear orderings must be considered. There are an exponential number of each of these even considered separately.

Another standard technique for dealing with general AGH-unification is to create a convergent equational theory and apply Narrowing to solve the unification problem [13]. Similar to the first technique, this method is also highly nondeterministic and builds a highly redundant complete set of unifiers. The reason for this is that there may be several possible applications of Narrowing at each step, and all of them must be tried, frequently leading to an exponential blowup of the search space.

In this paper, we try to overcome these problems by devising a set of inference rules that is simple, easy to implement, very deterministic in practice, and produces a small complete set of unifiers. We have developed a sound, complete and terminating set of inference rules for AGH-unification, along with uninterpreted function symbols. We implemented our inference rules in Maude [14], and they are being incorporated into Maude-NRL protocol analyzer (Maude-NPA) [2].

Generally, our algorithm introduces the concept of unconstrained variables and borrows the technique of solving linear diophantine equations such that our algorithm is easy to read and implement. The inference system includes five necessary rules: Trivial, Variable Substitution, N-Decomposition, Factorization and Annulization. A simpler version of these inference rules (without Factorization and Annulization and with simpler conditions for other rules) also applies to Abelian groups without homomorphism. This AGH-unification algorithm is also an extension of the unification algorithm for XOR with a homomorphisms as given in [15]. It is based on the same framework, but with more sophisticated inference rules. We also believe we have a useful framework that could be extended to other theories. The key point of this inference system is that in a unification problem most applications of inference rules are don't care nondeterministic, so that we never have to try other possibilities, and these rules have priority over others.

Here we give the outline of this paper. In Section 2, we give preliminary knowledge about unification and the AGH theory. In Section 3, we present a convergent rewriting system modulo $\mathcal{AC}$ for AGH. Next, we present our inference system in two parts: 1. A preprocessing step that transforms a unification problem into a purified form. This is given in Section 4. 2. This purification step is very similar to what happens in the combination algorithm, although not quite as strict. Our inference rules keep the unification problem in purified form. The inference rules which are used to solve the unification problem are given in Sections 6 and 7, where Section 6 presents the necessary rules and Section 7 presents some auxiliary rules. A nontrivial algorithm, called UnconstrainedReduce, is used in the Variable Substitution inference rule, and is presented in Section 5. The proofs of termination, soundness and completeness of our inference system are given in Sections 8, 9 and 10 respectively. Section 11 gives some concrete auxiliary rules. Section 12 shows some of our implementation results. The conclusion is given in Section 13.

2. Preliminary

2.1. Basic notation

Here we give some basic terminology which we will use in the following sections.

A *signature*, $\mathbb{F}$, is a finite set of fixed arity function symbols, where a constant is a function with 0 arguments. Let $\mathbb{V}$ be a set of variables. We say t is a term if $t \in \mathbb{V}$, or t has the form $f(t_1, t_2, \cdots, t_n)$, where $f^n \in \mathbb{F}$ (here n is the arity of the function), and t_i is a term. If $n = 0$, we call f a constant. If t and s are two terms, then we use $t[s]$ to denote that s occurs in t as a subterm.

We use $\mathbb{T}$ to denote the set of all terms. If t is a term, we use $Sym(t)$ to denote the multi-set of symbols occurring in t. We use $Vars(t)$ to denote the set of variables occurring in t, here t can be a term, or a set of terms.

- $Top(t)$ denotes the top symbol of term t. If $t = f(t_1, t_2, \cdots, t_n)$, $Top(t) = f$. If t is a variable x, $Top(x) = x$.
- Let t be a term and S be a set of terms. $Top(t; S)$ denotes the set of all terms in S, which have top symbol $Top(t)$.

The following are standard definitions [16].

A *substitution* σ is a mapping from $\mathbb{V}$ to the set of terms, which can be represented explicitly by a set of bindings of variables, i.e. $\{x_1 \longmapsto t_1, x_2 \longmapsto t_2, \cdots, x_n \longmapsto t_n\}$ representing the substitution which maps x_i to t_i for $i = 1, 2, \cdots, n$ and which maps y to itself otherwise. We let $Dom(\sigma) = \{x \mid x \longmapsto t \in \sigma$ and $x\sigma \neq x\}$ and $Range(\sigma) = \{t \mid x \longmapsto t \in \sigma$ and $x \in Dom(\sigma)\}$. When we apply σ to a term t, we denote it as $t\sigma$. If t is a variable x_i, $x_i\sigma = t_i$; if t has the form $f(s_1, s_2, \cdots, s_n)$, then $t\sigma = f(s_1\sigma, s_2\sigma, \cdots, s_n\sigma)$. The *composition* of two substitutions σ and θ, denoted $\sigma\theta$ is defined by $t(\sigma\theta) = (t\sigma)\theta$ for each term. The identity substitution is represented by Id. If some substitution σ is a solution of set of equations Γ, we write $\sigma \vDash \Gamma$.

A substitution σ is *idempotent* if $\sigma\sigma = \sigma$. We can see $Dom(\sigma) \cap Vars(Range(\sigma)) = \emptyset$, if and only if σ is idempotent. For consistency and convenience, we assume every substitution σ is idempotent.

The restriction of a substitution σ to a set of variables $\mathbf{X}$, denoted by $\sigma|_{\mathbf{X}}$, is the substitution which is equal to the identity substitution everywhere except over $X \cap Dom(\sigma)$, where it is equal to σ.

A set of *identities* E is a subset of $\mathbb{T} \times \mathbb{T}$. We write identities in the form $s \approx t$. An *equational theory* $=_E$ is induced by a set of identities E and it is the least congruence relation (i.e. reflexive, symmetric, transitive and congruence) on $\mathbb{T}$ that is closed under substitution and contains E.

If two terms, t and s, are equal with respect to a theory $=_E$, we write it as $t =_E s$.

Definition 1 *(E-unification problem, E-unifier, E-unifiable).* For a given signature $\mathbb{F}$ and a set of identities E, an *E-unification problem over* $\mathbb{F}$ is a finite multiset of equations

$$\Gamma = \left\{s_1 =^?_E t_1, s_2 =^?_E t_2, \cdots, s_n =^?_E t_n\right\}$$

between terms. A substitution σ such that $s_i\sigma =_E t_i\sigma$, $i = 1, 2, \cdots, n$ is called an *E-unifier* or a *solution* of Γ. $U_E(\Gamma)$ is the set of all E-unifiers of Γ. A unification problem Γ is called *E-unifiable* iff $U_E(\Gamma) \neq \emptyset$. If E is an empty set, we call the E-unification problem a *syntactic unification problem*.

Let $\Gamma = \{t_1 =^?_E s_1, \cdots, t_n =^?_E s_n\}$. We use $\mathbb{V}_\Gamma$ to denote the set of variables in Γ. And we use $[\sigma]$ to denote the set $\{x_1 =^? t_1, x_2 =^? t_2, \cdots, x_n =^? t_n\}$ if $\sigma = \{x_1 \longmapsto t_1, x_2 \longmapsto t_2, \cdots, x_n \longmapsto t_n\}$. For example, if we have $\sigma = x \longmapsto f(g(a), b), y \longmapsto g(f(x, a))$, then $[\sigma] = \{x =^? f(g(a), b), y =^? g(f(x, a))\}$.

Let $\Lambda = \{x_1 =^? t_1, x_2 =^? t_2, \cdots, x_n =^? t_n\}$. If $\{x_1 \longmapsto t_1, x_2 \longmapsto t_2, \cdots, x_n \longmapsto t_n\}$ is an idempotent substitution, we cay Λ is in *solved form*.

On the other hand, if Λ is in solved form: $\{x_1 =^? t_1, x_2 =^? t_2, \cdots, x_n =^? t_n\}$, we use λ_Λ to denote the corresponding substitution, namely the set $\{x_1 \longmapsto t_1, x_2 \longmapsto t_2, \cdots, x_n \longmapsto t_n\}$.

For an E-unification problem Γ, two substitutions σ and θ are called *equal*, denoted by $\sigma =_E \theta|_\Gamma$, if $x\sigma =_E x\theta$ for every variable occurring in Γ. A substitution σ is *more general modulo E* than another substitution θ, denoted $\sigma \leq_E \theta|_{\mathbb{V}_\Gamma}$ if there exists a substitution τ, such that for every variable x in $\mathbb{V}_\Gamma$, $x\sigma\tau =_E x\theta$. Note that the relation $\leq_E$ is a quasi-ordering, i.e. reflexive and transitive.

Definition 2 *(Complete set of E-unifiers).* A *complete set of E-unifiers* of an E-unification problem Γ is a set C of idempotent E-unifiers of Γ such that for each $\theta \in U_E(\Gamma)$ there exists $\sigma \in C$ with $\sigma \leq_E \theta|_{\mathbb{V}_\Gamma}$.

We call a complete set C of E-unifiers *minimal* if two distinct elements are incomparable w.r.t. $\leq_E$, i.e. if $\sigma \leq_E \theta|_{\mathbb{V}_\Gamma}$ and $\sigma, \theta \in C$ then $\sigma = \theta$.

A minimal complete set of unifiers for a syntactic unification problem Γ has only one element if it is not empty. We use $mgu(\Gamma)$ to denote this unifier. Without loss of generality, we suppose all variables in $mgu(\Gamma)$ are in $\mathbb{V}_\Gamma$.

For a given equational theory $=_E$, we will call two E-unification problems *equivalent modulo E* if they have same set of unifiers modulo theory $=_E$.

Definition 3 *(conservative extension).* Let E be a set of identities, we say a multi-set of equations Γ' is a *conservative E-extension* of another multi-set of equations Γ, if any solution of Γ' is also a solution of Γ and any solution of Γ can be extended to a solution of Γ', which means for any solution σ of Γ, there exists θ whose domain is the variables in $Vars(\Gamma') \setminus Vars(\Gamma)$ such that $\sigma\theta$ is a solution of Γ'.

The conservative E-extension relation is a transitive relation.
If $s\theta \neq t\theta$, we say a substitution θ satisfies the disequation $s \neq^? t$.
We sometimes omit E and Γ, if they are obvious from the context.

2.2. **AGH** preliminaries

Next, we introduce the basic notation related to the theory of a homomorphism over an Abelian group. We abbreviate this theory as **AGH**. The signature $\mathbb{F}$ here contains a unary function symbol h, an Abelian Group operator symbol $+$, an inverse operator symbol $-$, a constant 0 and arbitrarily many fixed-arity uninterpreted function symbols. Here h, $+$ and $-$ satisfy the following identities:

- $x + y \approx y + x$ [Commutativity – $\mathcal{C}$ for short]
- $(x + y) + z \approx x + (y + z)$ [Associativity – $\mathcal{A}$ for short]
- $x + 0 \approx x$ [Existence of Unit – $\mathcal{U}$ for short]
- $x + (-x) \approx 0$ [Existence of Inverse – $\mathcal{I}$ for short]
- $h(x) + h(y) \approx h(x + y)$ [Homomorphism – $\mathcal{H}$ for short]

We say a term t is *pure*, if $+ \notin Sym(t)$ and h can occur only as the top symbol of t. We call a term h**-term**, if the top function symbol of this term is h.

Example 4. In the following set of terms:

$$\{a + b, h(a + y), f\big(g(a), f\big(h(b)\big)\big), -x, -h(x), g\big(f(a, b)\big), h\big(f(a, b)\big)\}$$

only $-x$, $g(f(a, b))$ and $h(f(a, b))$ are pure terms; $h(a + y)$ and $h(f(a, b))$ are h-terms but $h(a + b)$ is not a pure term; $a + b$, $-h(x)$ and $f(g(a), h(b))$ are neither pure terms nor h-terms.

In a set of equations Γ, a *constrained* variable is a variable which occurs under an uninterpreted function symbol or an h symbol. Otherwise, we call it an *unconstrained* variable.

Definition 5. A term t of the form $t_1 + t_2 + \cdots + t_n$ is a PURE SUM if

- for every i, t_i is a pure term,
- in $\{t_1, \cdots, t_n\}$, there is at most one h-term,
- $n > 0$.

We will say an equation $\mathbf{S} =^? 0$ is in PURE SUM form if $\mathbf{S}$ is a PURE SUM. And we say an equation set is in PURE SUM form, if every equation in it is in PURE SUM form and each h-term occurs only once in all the equations. We say a variable x is a *pure* variable in some equation set Γ, if some equation $Q \in \Gamma$ has the form of $x + S =^? 0$. We notice here x may occur in S. We will use $PV(\Gamma)$ to denote the set of all the pure variables in an equation set Γ.

Because of the properties of Abelian Groups, $s =^?_{\mathbf{AGH}} t \iff s + (-t) =^?_{\mathbf{AGH}} 0$, we only consider equations of the form $\mathbf{S} =^? 0$. For convenience, we write an **AGH**-unification problem as $\{s_1 =^? 0, s_2 =^? 0, \cdots, s_n =^? 0\}$, where each s_i is a term.

3. Rewriting system $\mathfrak{R}_{AGH}$

We give a convergent rewriting system $\mathfrak{R}_{AGH}$ for $\mathcal{UIH}$ modulo $\mathcal{AC}$:

- $x + 0 \rightarrow x$
- $x + (-x) \rightarrow 0$
- $-0 \rightarrow 0$
- $-(-x) \rightarrow x$
- $-(x + y) \rightarrow -x + (-y)$
- $h(x) + h(y) \rightarrow h(x + y)$
- $h(0) \rightarrow 0$
- $-h(x) \rightarrow h(-x)$

- $x + y + (-x) \to y$
- $h(x) + y + h(z) \to h(x + z) + y$.

Here modulo $\mathcal{AC}$ means we consider the following two pairs of terms as identical terms:

- $x + y$ and $y + x$,
- $(x + y) + z$ and $x + (y + z)$.

Here we are using the extended rewrite system for $\mathcal{UIH}$ modulo $\mathcal{AC}$, since the extended rewriting system is much more efficient than the class rewriting system [13]. By the polynomial ordering in which $\tau(x+y) = \tau(x)+\tau(y)+1$, $\tau(-x) = 2\tau(x)$, $\tau(0) = 1$ and $\tau(h(x)) = 2\tau(x) + 3$, we can prove this rewriting system is terminating modulo $\mathcal{AC}$. The $\mathcal{AC}$-completion procedure [13] can show our rewriting system is confluent. So in our inference system, unless stated otherwise, all terms are in reduced form by $\mathfrak{R}_{AGH}$. For a term t, we use $t\downarrow$ to denote t's reduced form. For example, if $t = x + (-x) + h(a) + h(-c) + b + 0$, then $t\downarrow = h(a + (-c)) + b$. We also call $t\downarrow$ the normal form of t. We say some term t can *cancel* s, if $t\downarrow = S + (-s\downarrow)$.

Also, we always keep our terms and equations *abbreviated* by the following rules (c, d are integers):

- $x + x + \cdots + x \to cx$ if x occurs c times in the left hand side,
- $cx + dx \to (c + d)x$,
- $c(-x) \to -cx$,

where c, d are integers and $c + d$ is the common addition between integers. Without ambiguity, we still use $+$ between integers to denote the common addition between integers. We always assume every term in our equation is abbreviated unless we state otherwise. We call c in ct the *coefficient* of t. c can be negative but not zero. If c is negative, then ct is the abbreviation of $(|c|)(-t)$. If some term's coefficient is one or negative one, we call it a *monic term*.

4. Initialization

Our inference rules will have some requirement on the form of the problem. We require all the equations in the problem in PURE SUM form. We know that a unification problem in PURE SUM form means: every equation is in PURE SUM form and every h-term occurs only once in the problem. So first, we use the following rule called **Purify** to convert one equation not in PURE SUM form to PURE SUM form.

$$\frac{\Gamma \cup \{S + t[s] =^? 0\}}{\Gamma \cup \{S + t[x] =^? 0\} \cup \{x + (-s) =^? 0\}}$$

where Γ is a set of equations, S is a term, t is a pure term, s is a proper subterm of t, with $Top(s) \in \{+, h\}$, and x is a fresh variable.

Second, the following two rules are called **Singlize**, which will delete other duplicated occurrence of an h-term or an inverse of an h-term:

$$\frac{\Gamma \cup \{S + h(t) =^? 0\} \cup \{T + h(t) =^? 0\}}{\Gamma \cup \{S + h(t) =^? 0\} \cup \{S + (-T) =^? 0\}}$$

and

$$\frac{\Gamma \cup \{S + h(t) =^? 0\} \cup \{T + h(-t) =^? 0\}}{\Gamma \cup \{S + h(t) =^? 0\} \cup \{S + T =^? 0\}}$$

where Γ is a set of equations in PURE SUM form, S and T are terms, and t is a pure term.

For example, in the unification problem $\{x + h(x) =^? 0, y + h(x) =^? 0\}$, $h(x)$ occurs twice, but we want it to only occur once. So we replace $y + h(x) =^? 0$ by $y + x =^? 0$.

The procedure of converting a unification problem to a unification problem in PURE SUM form is called *purification* and we say the problem is *purified*. When we purify the unification problem, we will apply **Purify** exhaustively then apply **Singlize**.

Since

- the numbers of $+$ and h symbols (denoted by N) which occur under f or h is finite, and
- the number of duplicated occurrences of an h-term or an inverse of an h-term (denoted by N_h) is finite, and
- **Purify** will decrease N, and
- **Singlize** will decrease N_h but not increase N since each equation has at most one h-term.

The purification procedure will finally terminate.

For purification, we also have the following lemma:

Lemma 6. *Any set of equations Γ can be purified to a set of equations Γ' in* PURE SUM *form, which is a conservative extension of Γ.*

Proof. In **Purify** and **Singlize**, some non-pure term or some h-term will be removed and none will ever be created, purification on a unification problem Γ will halt in a PURE SUM form, Γ', which is a conservative extension of Γ. □

Example 7. Purify the equation:

$$5f\big(x_1, h(x_2)\big) + h(x_2) + 2g\big(x_3 + f(x_2, x_1)\big) =^? 0$$

First we introduce a new variable v_1 to substitute for $h(x_2)$ in $f(x_1, h(x_2))$ by applying Purify and we get:

$$\big\{5f(x_1, v_1) + h(x_2) + 2g\big(x_3 + f(x_2, x_1)\big) =^?,\ 0v_1 + h(-x_2) =^? 0\big\}$$

Next we use the new variable v_2 to replace $x_3 + f(x_2, x_1)$ in $g(x_3 + f(x_2, x_1))$ by applying Purify and we get:

$$\big\{5f(x_1, v_1) + h(x_2) + 2g(v_2) =^? 0,\ v_1 + h(-x_2) =^? 0,\ v_2 + (-x_3) + \big(-f(x_2, x_1)\big) =^? 0\big\}$$

There is $h(x_2)$ and $h(-x_2)$ in these two equations, so we can use Singlize to delete one of them and get the final PURE SUM set of equations.

$$\big\{5f(x_1, v_1) + h(x_2) + 2g(v_2) =^? 0,\ v_1 + 5f(x_1, v_1) + 2g(v_2) =^? 0,\ v_2 + (-x_3) + \big(-f(x_2, x_1)\big) =^? 0\big\}$$

We will apply Purification whenever the unification problem is not in a PURE SUM form after applying each inference rule. So from now on, we suppose every equation has the form $\mathbf{S} =^? 0$, where $\mathbf{S}$ is a PURE SUM. In the following sections, we will use Γ to denote a set of equations, $\mathbf{S}$ to denote a PURE SUM, s, t, s_i, t_i to denote pure terms, x, y, v, x_i, y_i, v_i, etc. to denote variables, σ, θ to denote substitutions.

5. UnconstrainedReduce algorithm

In [5], the authors present the relation between unification modulo Abelian Groups and solving linear equation systems. And in [9] and [10], the authors show that solving unification problems modulo monoidal theories (AGH is one of these theories) is equivalent to solving systems of linear equations over a semiring which can be efficiently solved, for instance, by computing the Hermite normal form of the corresponding integer matrix. However these papers only focuses on elementary unification, i.e. no uninterpreted function symbols. Unification becomes much more complicated when uninterpreted function symbols are considered. However, we still can *borrow* some techniques from solving linear diophantine equations.

Let use have a look at some linear diophantine equations:

- $4x + 5 = 0$, no integer solutions.
- $2x + 6 = 0$, the solution is $x = -3$.
- $3x + 2y = 0$, the solution is $x = -2y'$, $y = 3y'$ for any integer y'.

In [17], Knuth first looks at the coefficients of the variables and the constant of a single equation. If the greatest common divisor of them is 1, some variable's coefficient is not 1 and the constant is not zero, then there is no solution. Otherwise a method is given to get the solution. We use the above examples to explain the method here (for the detail of this method, please refer to [17]):

Example 8.

$$2x + 6 = 0$$

First introduce a new variable x and let $x = x' - 3$, then the original equation becomes $2x' = 0$. We can get $x' = 0$. Then $x = 0 - 3 = -3$, which is a solution.

Example 9.

$$3x + 2y = 0$$

First introduce a new variable y' and let $y = y' - x$, then the original equation becomes $x + 2y' = 0$. We can get $x = -2y'$. Put this back to $y = y' - x$ and get $y = 3y'$.

We find that this method just keeps introducing a new variable through replacing some preexisting variable, whose coefficient's absolute value is the smallest one among all the coefficients and the constant, and it reduces other variables' coefficients and constant until some variable's coefficient becomes 1 or -1, then solves it.

Let use have a look at what will happen if we apply this method to the equations which has no solution:

Example 10.

$4x + 5 = 0$

Introduce x' such that $x = x' - 1$. The original equation becomes $4x' + 1 = 0$. If we continue to introduce a new variable $x' = x''$, we will get a similar equation $4x'' + 1 = 0$. This will become an infinite loop.

From the Extended Euclidean Algorithm, we can prove if there is no solution, then we will create an equation meeting the following conditions:

- only one variable is left in the equation,
- the constant is not zero, and
- the absolute value of the variable's coefficient is greater that the constant's absolute value.

This property can be proven by applying the Extended Euclidean Algorithm.

We can abstract these equations by the following **AGH**-unification problems:

- $4x + 5a =^? 0$;
- $2x + 6a =^? 0$;
- $3x + 2y =^? 0$.

We can get the same result as for the linear diophantine equations. However, there is still some difference in the following aspects:

- There are arbitrarily many constants and function symbols in the **AGH**-unification problem;
- Some variables may occur under some function symbol. For example, if we have $x + f(x) =^? 0$, we cannot let $x \longmapsto x' + f(x)$, which will cause a loop.
- Homomorphism plays an important role in our unification problem. For example, for $2x + h(y) =^? 0$, $x \longmapsto h(z)$, $y \longmapsto -2z$ is a solution.

Our inference system will be based on the idea of solving linear diophantine equations while considering the above issues. Thus before we give the inference rules, we will give an algorithm, which is based on the algorithm solving linear diophantine equations from [17] we mentioned above. It is called **UnconstrainedReduce**.

This algorithm has two parts.

The first part results in an equation containing one unconstrained variable such that the absolute value of this variable's coefficient is maximum in the equation. If some unconstrained variable's coefficient is one, we will solve this variable. This part uses the method we mentioned above.

After the first part, we can get an equation with only one unconstrained variable occurring in it, for example $2x + h(z) =^? 0$. We can delete other occurrences of this unconstrained variable in other equations. For example, if we also have $3x + 4z =^? 0$, we can replace it by $8z - h(3z) =^? 0$, since $2x + h(z) =^? 0 \iff 6x + h(3z) =^? 0$ and $3x + 4z =^? 0 \iff 6x + 8z =^? 0$. This step is done in the second part of the algorithm.

After introducing the algorithm, we will give a proof that the result of the algorithm will give a conservative extension of the original unification problem.

Next,we give the technical details of **UnconstrainedReduce**.

The first part is called **Maxi**(Q, V). It takes an equation Q and a set of variables V which is a subset of $PV(Q)$ and outputs another pair $(\Omega, \tilde{\Lambda})$, where Ω is an empty set or a set of one equation $\{\tilde{Q}\}$, and $\tilde{\Lambda}$ is an equation set in solved form. $\tilde{Q} \cup \tilde{\Lambda}$ is a conservative extension of Q and $\tilde{Q}$ has only one unconstrained variable occurring in it and the absolute value of this variable is maximum in $\tilde{Q}$. If Ω is an empty set, then we say Q was *solved* during the process of **Maxi**(Q, V). $\tilde{\Lambda}$ is used to store the substitutions which are generated during **Maxi**(Q, V).

Generally, the purpose of our algorithm is to solve the variables in Q. So the set V, a subset of $PV(Q)$ is our *target set* of variables to be waiting to be solved. The change of V obeys the following rules:

- Initially, V is a subset of $PV(Q)$, we can choose the members by our needs. For example, we may choose all the unconstrained variables in a unification problem occurring in Q.
- If some variable is replaced by other terms (we say this variable is *solved*) or is canceled in Q during the process of this algorithm, it will be removed from this target set.
- Any fresh variable which is introduced in the algorithm will be put into this target set.

The result of running this algorithm will be that V is a singleton set with some condition (we will give the condition later) or Q is solved (namely, Q is replaced by a set of solved form and Ω is empty).

Theoretically, we can solve any variable by introducing a fresh variable. For example if we have an equation $4x + 3y + 5z =^? 0$, then different Vs will give different results. If $V = \{z\}$, then 5 is the biggest coefficient already. We do not need to

```
Algorithm MAXI
Input:
    • A set of variables $V = \{x_1, x_2, \cdots, x_n\}$, where each $x_i$ occurs in pure form in $Q$.
    • An equation $Q: c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m =^? 0$, where each $t_j$ is a monic term.
Output:
    • A pair $(\Omega, \tilde{\Lambda})$, where $\Omega$ is an empty set or a set of one equation $\{\tilde{Q}\}$, and $\tilde{\Lambda}$ is an equation set in solved form.
 1: Let $\Lambda' = \emptyset$, $Q' = Q$ and $V' = V$. Build
    • a multiset $C_Q$, which is $\{|c_1|, |c_2|, \cdots, |c_n|\}$, to denote the set of absolute values of coefficients of $V$ in $Q'$, and
    • a multiset $C'_Q$, which is $\{|c'_1|, |c'_2|, \cdots, |c'_n|\}$, to denote the set of absolute values of coefficients of the other terms in $Q'$.
 2: Calculate $|C_Q|$:
 3: if $|C_Q| = 1$ then
 4:     calculate $|C'_Q|$:
 5:     if $|C'_Q| = 0$ then
 6:         add $x_1 =^? 0$ into $\Lambda'$;
 7:         return $(\emptyset, \Lambda')$.
 8:     else if $|C'_Q| \neq 0$ and $|c_1|$ is bigger than all the elements in $C'_Q$ then
 9:         return $(Q', \Lambda')$.
10:     end if
11: else $\{|C_Q| \neq 1\}$
12:     let $min = j$ if $|c_j| \leq |c_i|$ for all $0 < i \leq n$; {If $j$ is not unique, choose any one of them.}
13:     if $c_{min} = 1$ then
14:         add the equation $x_{min} =^? -c_1x_1 + \cdots + (-c_{min-1}x_{min-1}) + (-c_{min+1}x_{min+1}) + \cdots + (-c_nx_n) + (-c'_1t_1) + \cdots + (-c'_mt_m)$ into $\Lambda'$;
15:         return $(\emptyset, \Lambda')$
16:     else if $c_{min} = -1$ then
17:         add the equation $x_{min} =^? c_1x_1 + \cdots + c_{min-1}x_{min-1} + c_{min+1}x_{min+1} + \cdots + c_nx_n + c'_1t_1 + \cdots c'_mt_m$ into $\Lambda'$;
18:         return $(\emptyset, \Lambda')$.
19:     else $\{|c_{min}| > 1\}$
20:         Introduce a new variable $x'$, and add the equation $P$
            $$x_{min} =^? x' + \left(-\left\lfloor \frac{c_1}{c_{min}} \right\rfloor x_1\right) + \cdots + \left(-\left\lfloor \frac{c_{min-1}}{c_{min}} \right\rfloor x_{min-1}\right) + \left(-\left\lfloor \frac{c_{min+1}}{c_{min}} \right\rfloor x_{min+1}\right) + \cdots + \left(-\left\lfloor \frac{c_n}{c_{min}} \right\rfloor x_n\right) + \left(-\left\lfloor \frac{c'_1}{c_{min}} \right\rfloor t_1\right) + \cdots + \left(-\left\lfloor \frac{c'_m}{c_{min}} \right\rfloor t_m\right)$$
            into $\Lambda'$;
21:         apply $\lambda_{\{P\}}$ to $Q'$ and $\Lambda'$;
22:         remove the variables which are not in $Q'$ from $V'$ and add $x'$ to $V'$;
23:         update $C_Q$ as the set of absolute values of coefficients of $V'$ in $Q'$ and $C'_Q$ as the set of absolute values of coefficients of all the terms in $Q'$;
24:         go to step 2.
25:     end if
26: end if
```

Fig. 1. Algorithm MAXI.

do anything. If $V = \{x\}$, then we will choose x and let $x \longmapsto x' - z$ and get $4x' + 3y + z =^? 0$. x is solved and x' is introduced, so x is removed from V and x' is added into V. Now the coefficient of x' is the largest already. The algorithm will not run on this equation. However, if $V = \{x, z\}$ and after the first step, V becomes $\{x', z\}$ and z's coefficient is the smallest among the coefficients of x' and z, then we will do a further step to solve z and get $z \longmapsto -4x' - 3y$ and z is removed from V. Q is now solved.

Hence, in order to make our algorithm more general, we use a set V to denote the set of variables we are going to solve. Of course, the variables in V have to occur pure in Q, so V is a subset of $PV(Q)$. In our inference rules, we will only try to solve the free variables occurring in Q.

So formally the purpose of this algorithm is to achieve the following results:

- $\Omega \cup \tilde{\Lambda}$ is a conservative extension of Q.
- If $\Omega = \{\tilde{Q}\}$, then $V = \{x\}$ and x occurs in $\tilde{Q}$ and x's coefficient's absolute value is the largest one among all the coefficients' absolute values in $\tilde{Q}$'s monic terms.
- If $\Omega = \emptyset$, then it means Q is solved by this algorithm.

Algorithm. We present the algorithm **Maxi** in Fig. 1.

The second part of the **UnconstrainedReduce** algorithm is **Uniq**$(\Gamma, (\Omega, \tilde{\Lambda}))$. It takes an equation set Γ and the output of **Maxi**$(\mathbf{Q}, \mathbf{V})$ as input and outputs another pair $(\tilde{\Gamma}, \tilde{\Lambda}')$. As we said above, this part removes other duplicated (or inverse) occurrences of an unconstrained variables. Since Ω is the output of **Maxi**$(\mathbf{Q}, \mathbf{V})$, we know which is the unconstrained variable if Ω is not empty. Γ is the set of equations containing the duplicated occurrences of the unconstrained variable. So $\tilde{\Gamma}$ contains the result of Γ after deleting the duplicated occurrences from Γ. $\tilde{\Lambda}$ is also used to store the substitutions which are generated during **Maxi**(Q, V).

```
Algorithm Uniq
Input:
        • An equation set $\Gamma$.
        • A set $\Omega$, which is the empty set or $\{cx + \mathbf{T} =^? 0\}$, where $|c|$ is larger than all the absolute values of the coefficients of the monic terms in $T$.
        • An equation set in solved form $\tilde{\Lambda}$.
Output:
        • A pair $(\tilde{\Gamma}, \Lambda')$, where $\tilde{\Gamma}$ is a set of equations and $\Lambda'$ is an equation set in solved form.
 1: Let $\Gamma' = \Gamma\lambda_{\tilde{\Lambda}}$ and $\Lambda' = \tilde{\Lambda}$, then look at $\Omega$:
 2: if if $\Omega$ is the empty set then
 3:    return $(\Gamma', \Lambda')$.
 4: else {$\Omega$ is not an empty set}
 5:    for each equation in $\Gamma'$, which has the form $c'x + \mathbf{S} =^? 0$ do
 6:       if S is zero then
 7:          add equation $x =^? 0$ into $\Lambda'$;
 8:          let $\tilde{\Gamma} = (\Gamma' \cup \{T =^? 0\})[x \longmapsto 0]$;
 9:          return $(\tilde{\Gamma}, \Lambda')$.
10:       else {S is not zero}
11:          remove $c'x + \mathbf{S} =^? 0$ from $\Gamma'$ and add $-c'\mathbf{T} + c\mathbf{S} =^? 0$ into $\Gamma'$.
12:       end if
13:    end for
14: end if
15: Let $\tilde{\Gamma} = \Gamma' \cup \{cx + \mathbf{T} =^? 0\}$;
16: return $(\tilde{\Gamma}, \Lambda')$
```

Fig. 2. Algorithm Uniq.

So formally, the purpose of this algorithm is to achieve the followings:

- If Ω is the empty set, then $\tilde{\Gamma} = \Gamma\lambda_{\tilde{\Lambda}}$
- If Ω is a singleton set $\{\tilde{Q}\}$, then for x, the pure variable in $\tilde{Q}$ with the largest absolute value of the coefficients among all the terms in $\tilde{Q}$, $\tilde{\Gamma}/\Omega$ does not contain x as a pure variable, which means only $\tilde{Q}$ contains x as a pure variable.
- In both of the above cases, $\tilde{\Gamma} \cup \tilde{\Lambda}'$ is a conservative extension of $\Gamma \cup \Omega \cup \tilde{\Lambda}$.

The algorithm **UnconstrainedReduce**$(\Gamma, Q, V) =$ **Uniq**$(\Gamma,$ **Maxi**$(Q, V))$. Next we present the algorithm **Uniq** in Fig. 2.

Here we give an example to explain how **Maxi** works.

Example 11. Run **Maxi**$(5x + 13y + 7z + 4f(z) =^? 0, \{x, y\})$. First x has the smallest coefficient among x and y, we set $x \longmapsto v_1 - 2y - z$ and get

$$5v_1 + 3y + 2z + 4f(z) =^? 0$$

and the new variable set $\{v_1, y\}$. y's coefficient is the smallest one now. So set $y \longmapsto v_2 + (-v_1) + (-f(z))$ and get

$$2v_1 + 3v_2 + 2z + f(z) =^? 0$$

and the new variable set $\{v_1, v_2\}$. Choose v_1 this time and we set $v_1 \longmapsto v_3 + (-v_2) + (-z)$ and get

$$2v_3 + v_2 + f(z) =^? 0$$

and the new variable set $\{v_3, v_2\}$. Here v_2's coefficient is 1, we can solve this equation by setting $v_2 \longmapsto (-2v_3) + (-f(z))$.
Finally, we get a solved equation set $\tilde{\Lambda} = \{x \longmapsto -13v_3 + -4z + 7f(z), y \longmapsto -5v_3 - 3f(z) + z\}$.
If we run this algorithm for this equation on the variable set $\{x\}$, then since 5 is less then 13, we will set $x \longmapsto v_1 - 2y - z$ and get $5v_1 + 3y + 2z + 4f(z) =^? 0$ and the new variable set is $\{v_1\}$. Now v_1's coefficient is the biggest among 5, 3, 2 and 4, so the process will stop. Now, the solved equation set Λ is $\{x =^? v_1 + (-2y) + (-z)\}$.

Here we give an example to explain how **Uniq** works.

Example 12. Let

$$\Omega = \{4v + 2z + 3y + f(y) =^? 0\},$$
$$\tilde{\Lambda} = \{x =^? v + (-2y) + (-f(y))\},$$
$$\Gamma = \{3x + 4f(z) =^? 0,\ 2x + (-3y) =^? 0\}.$$

Run **Uniq**$(\Gamma, (\Omega, \tilde{\Lambda}))$. First we apply $\lambda_{\tilde{\Lambda}}$ to Γ and get

$$\Gamma' = \{3v + (-6y) + (-3f(y)) + 4f(z) =^? 0,\ 2v + (-7y) + (-2f(y)) =^? 0\}.$$

In Ω, v has the largest coefficient and both of the equations in Γ contain v as a pure variable, so:

- remove $3v + (-6y) + (-3f(y)) + 4f(z) =^? 0$ and add $\{-6z + (-33y) + (-15f(y)) + 16f(z) =^? 0\}$ to Γ', and
- remove $2v + (-7y) + (-2f(y)) =^? 0$ and add $\{-2z + (-17y) + (-5f(y)) =^? 0\}$ into Γ'

then we get

$$\Gamma' = \{-6z + (-33y) + (-15f(y)) + 16f(z) =^? 0,\ -2z + (-17y) + (-5f(y)) =^? 0,\ 4v + 2z + 3y + f(y) =^? 0\}$$

Finally, we output $(\Gamma', \tilde{\Lambda})$.

Before we prove our algorithm can achieve the results we mentioned, we need the following fact: For free Abelian Groups, $t =^? 0 \iff ct =^? 0$, where t is a term and c is a coefficient. This follows from the properties of the theory **AGH**.

Lemma 13. *Let Q be an equation, and V be a subset of the pure variables occurring in Q. Then after finitely many steps,* **Maxi(Q, V)** *will output a pair $(\Omega, \tilde{\Lambda})$, where Ω is an empty set or a singleton of an equation $\tilde{Q}$ and $\tilde{\Lambda}$ is an equation in solved form, such that:*

1. *If $\Omega = \{\tilde{Q}\}$, then there is a variable x, such that either*
   - *$PV(\tilde{Q}) \cap V = \{x\}$, or*
   - *$PV(\tilde{Q}) \cap V = \emptyset$, $PV(\tilde{Q}) \setminus PV(Q) = \{x\}$ and x is a variable which does not occur in Q.*

   In both cases, x's coefficient's absolute value is the largest one among all the coefficients' absolute values in $\tilde{Q}$'s monic terms.
2. *$\Omega \cup \tilde{\Lambda}$ is a conservative extension of Q.*

Proof. From the procedure, we can use N, the largest absolute value among all the coefficients in Q, which with standard $\leq$ on natural number is a well founded ordering, to be the measure in this algorithm. We see every time we run **Maxi**:

- Some variable's coefficient absolute value is 1, Q is removed and the algorithm halts. $N = 0$.
- Suppose no variable's coefficient absolute value is 1 and the smallest one is n. Suppose a term, t's coefficient is N. Then after the third step, t's coefficient will be $N \bmod n$. Since $n < N$, the absolute value of the coefficient of t decreases.

So this algorithm will halt in finitely many steps.

For proving (1), first we want to prove $|PV(\tilde{Q}) \cap V| \leq 1$. V is a subset of pure variables in Q, and V' is a subset of pure variables in Q', which keeps changing in the algorithm starting from Q and ending with $\tilde{Q}$ if Ω is not empty. V' and Q' are changed only in the third part of step 3. In step 3, we only add fresh variables into V' and remove variables from V' if the variables are replaced by some fresh variables, and all the added variables to Q' also are added into V'. Thus $(PV(Q') \cap V) \subseteq V'$ and $(PV(Q') \setminus PV(Q)) \subseteq V'$ during the procedure of the algorithm. So it is enough to show finally, $|V'| \leq 1$. Assume $|V'| > 1$, then from the second part of step 2 in **Maxi**, we go to step 3. In step 3, if $|c_{min}| = 1$, the equation will be removed and Ω will be the empty set after this step. If $|c_{min}| \neq 1$, then the third part of step 3 applies, and after it, step 2 will be started again. We already proved the algorithm will halt. So finally, if $\Omega \neq \emptyset$, $|V'| \leq 1$, which means $|PV(\tilde{Q}) \cap V| \leq 1$.

If $\Omega \neq \emptyset$, then the algorithm only halts at the first part of the second step. When the algorithm stops, $|C_Q| = 1$, which means $|V'| = 1$. And the variable in V' has the largest absolute coefficient value among all the coefficients of Q'. We claim that this variable in V' satisfies the conditions for x in (1). In the third part of the third step, we only add *fresh* variables into V', so if finally, $V' \cap V = \emptyset$, which means $|PV(\tilde{Q}) \cap V| = 0$, then $PV(Q') \setminus PV(Q) = V' = \{x\}$. If $V' \cap V \neq \emptyset$ finally and $|V'| = \{x\}$, then $PV(\tilde{Q}) \cap V = \{x\}$.

For proving (2), it is enough to prove after every step, $\{Q'\} \cup \Lambda'$ is a conservative extension of the old $\{Q'\} \cup \Lambda'$ before the step was implemented. For reducing the ambiguity, we denote the old $\{Q'\} \cup \Lambda'$ before the step was implemented as $\{Q''\} \cup \Lambda''$.

In step 1. We do nothing to $\{Q''\} \cup \Lambda''$.

In step 2. If $|C_Q| = 1$ but $|C'_Q| \neq 0$, then we do nothing to $\{Q''\} \cup \Lambda''$. If $|C'_Q| = 0$, then $\{Q''\} = \{c_1x_1 =^? 0\}$, $\{Q'\} = \emptyset$ and $\Lambda' = \Lambda' \cup \{x =^? 0\}$. Hence $\{Q'\} \cup \Lambda'$ is a conservative extension of $\{Q''\} \cup \Lambda''$.

In step 3. If $c_{min} = 1$, then

$$\{Q''\} = \{c_1x_1 + \cdots + x_{min} + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m =^? 0\}$$
$$\{Q'\} = \emptyset$$
$$\Lambda' = \Lambda'' \cup \{x_{min} = -c_1x_1 + \cdots + (-c_{min-1}x_{min-1}) + (-c_{min+1}x_{min+1}) + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m =^? 0\}$$

Since $\theta \vDash t + s =^? 0 \iff \theta \vDash t =^? -s$, we know $\{Q'\} \cup \Lambda'$ is a conservative extension of $\{Q''\} \cup \Lambda''$.

If $c_{min} = -1$, then it is same as the case where $c_{min} = 1$.

If $|c_{min}| \neq 1$, then

$$\begin{aligned}\{Q''\} &= \{c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m =^? 0\}\\ \{Q'\} &= \{(c_1 \bmod c_{min})x_1 + \cdots + (c_{min-1} \bmod c_{min})x_{min-1}c_{min}x' + (c_{min+1} \bmod c_{min+1})x_{min+1} + \cdots\\ &\quad + (c_n \bmod c_{min})x_n(c'_1 \bmod c_{min})t_1 + \cdots + (c'_m \bmod c_{min})t_m =^? 0\}\end{aligned}$$

and

$$\begin{aligned}\Lambda' = \Lambda'' \cup \Bigg\{x_{min} &=^? x' + \left(-\left\lfloor\frac{c_1}{c_{min}}\right\rfloor x_1\right) + \cdots + \left(-\left\lfloor\frac{c_{min-1}}{c_{min}}\right\rfloor x_{min-1}\right) + \left(-\left\lfloor\frac{c_{min+1}}{c_{min}}\right\rfloor x_{min+1}\right) + \cdots\\ &+ \left(-\left\lfloor\frac{c_n}{c_{min}}\right\rfloor x_n\right) + \left(-\left\lfloor\frac{c'_1}{c_{min}}\right\rfloor t_1\right) + \cdots + \left(-\left\lfloor\frac{c'_m}{c_{min}}\right\rfloor t_m\right)\Bigg\}\end{aligned}$$

Because $\lfloor\frac{c_i}{c_{min}}\rfloor + (c_i \bmod c_{min}) = c_i$, if

$$(c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m)\theta = 0$$

then

$$\begin{aligned}&(c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m)\theta\\ &\quad = (c_1 \bmod c_{min})x_1\theta + \cdots + (c_n \bmod c_{min})x_n\theta + (c'_1 \bmod c_{min})t_1\theta + \cdots + (c_n \bmod c_{min})t_n\theta + \left(\left\lfloor\frac{c_1}{c_{min}}\right\rfloor\right)x_1\theta + \cdots\\ &\qquad + \left(\left\lfloor\frac{c_n}{c_{min}}\right\rfloor\right)x_n\theta + \left(\left\lfloor\frac{c'_1}{c_{min}}\right\rfloor\right)t_1\theta + \cdots + \left(\left\lfloor\frac{c'_m}{c_{min}}\right\rfloor\right)t_m\theta\end{aligned}$$

Because $\lfloor\frac{c_{min}}{c_{min}}\rfloor = 1$ and $c_{min} \bmod c_{min} = 0$, we extend θ to be

$$\begin{aligned}\theta' = \theta \cup \Bigg\{x_{min} &\longmapsto x' + \left(-\left\lfloor\frac{c_1}{c_{min}}\right\rfloor x_1\right) + \cdots + \left(-\left\lfloor\frac{c_{min-1}}{c_{min}}\right\rfloor x_{min-1}\right) + \left(-\left\lfloor\frac{c_{min+1}}{c_{min}}\right\rfloor x_{min+1}\right) + \cdots\\ &+ \left(-\left\lfloor\frac{c_n}{c_{min}}\right\rfloor x_n\right) + \left(-\left\lfloor\frac{c'_1}{c_{min}}\right\rfloor t_1\right) + \cdots + \left(-\left\lfloor\frac{c'_m}{c_{min}}\right\rfloor t_m\right)\Bigg\}\end{aligned}$$

which satisfies Λ'.

Then

$$\begin{aligned}&(c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m)\theta'\\ &\quad = (c_1 \bmod c_{min})x_1\theta' + \cdots + (c_n \bmod c_{min})x_n\theta' + (c'_1 \bmod c_{min})t_1\theta' + \cdots + (c_n \bmod c_{min})t_n\theta'\end{aligned}$$

which means $\theta' \vDash (\{Q'\} \cup \Lambda')$.

On the other hand, suppose $\theta \vDash (\{Q'\} \cup \Lambda')$, which means

$$\begin{aligned}x_{min}\theta = x'\theta &+ \left(-\left\lfloor\frac{c_1}{c_{min}}\right\rfloor x_1\theta\right) + \cdots + \left(-\left\lfloor\frac{c_{min-1}}{c_{min}}\right\rfloor x_{min-1}\right) + \left(-\left\lfloor\frac{c_{min+1}}{c_{min}}\right\rfloor x_{min+1}\theta\right) + \cdots + \left(-\left\lfloor\frac{c_n}{c_{min}}\right\rfloor x_n\theta\right)\\ &+ \left(-\left\lfloor\frac{c'_1}{c_{min}}\right\rfloor t_1\theta\right) + \cdots + \left(-\left\lfloor\frac{c'_m}{c_{min}}\right\rfloor t_m\theta\right)\end{aligned}$$

then

$$\begin{aligned}x'\theta = x_{min}\theta &+ \left(\left\lfloor\frac{c_1}{c_{min}}\right\rfloor x_1\theta\right) + \cdots + \left(\left\lfloor\frac{c_{min-1}}{c_{min}}\right\rfloor x_{min-1}\right) + \left(\left\lfloor\frac{c_{min+1}}{c_{min}}\right\rfloor x_{min+1}\theta\right) + \cdots\\ &+ \left(\left\lfloor\frac{c_n}{c_{min}}\right\rfloor x_n\theta\right) + \left(\left\lfloor\frac{c'_1}{c_{min}}\right\rfloor t_1\theta\right) + \cdots + \left(\left\lfloor\frac{c'_m}{c_{min}}\right\rfloor t_m\theta\right)\end{aligned}$$

Then replacing $x'\theta$ in $Q'\theta$ and using $c_i = \lfloor\frac{c_i}{c_{min}}\rfloor + (c_i \bmod c_{min})$, we get

$$c_1x_1\theta + \cdots + c_nx_n\theta + c'_1t_1\theta + \cdots + c'_mt_m\theta = 0$$

which means $\theta \vDash \{Q''\}$.

From the above, we know $\Omega \cup \tilde{\Lambda}$ is a conservative extension of $\{Q\}$. □

Lemma 14. *Let Γ be a unification problem, and $(\Omega, \tilde{\Lambda})$ be an output of* **Maxi**(Q, V). *In the algorithm* **Uniq**$(\Gamma, (\Omega, \tilde{\Lambda}))$, *after finitely many steps, we get a pair $(\tilde{\Gamma}, \tilde{\Lambda}')$, such that*

1. • *If Ω is the empty set, then $\tilde{\Gamma} = \Gamma\lambda_{\tilde{\Lambda}}$ and $\tilde{\Lambda}' = \tilde{\Lambda}$.*
   • *If Ω is the singleton $\{\tilde{Q}\}$, then either $\tilde{Q} \in \tilde{\Gamma}$ and $\tilde{\Gamma} \setminus \{\tilde{Q}\}$ does not contain x as a pure variable, or $\tilde{\Gamma}$ does not contain x as a pure variable, where x is the pure variable in $\tilde{Q}$ which has the largest absolute value of the coefficient.*
2. *$\tilde{\Gamma} \cup \tilde{\Lambda}'$ is a conservative extension of $\Gamma \cup \Omega \cup \tilde{\Lambda}$.*

Proof. We use N_x, which is the number of occurrences of x in Γ', to be the measure in this procedure. Here x is the pure variable with the largest absolute value of the coefficient in $\tilde{Q}$.

Let us look at the algorithm step by step. For step 1, we do nothing. In step 2, if $\Omega = \emptyset$, stop. If $\Omega \neq \emptyset$, then for every $c'x + \mathbf{S} =^? 0$ we will do several things:

- If $\mathbf{S} = 0$, then the algorithm will stop after this step.
- If $\mathbf{S} \neq 0$, $c'x + \mathbf{S} =^? 0$ will be removed from Γ' and add $c'(-T) + cS =^? 0$ which has no x in pure form in it. So N_x decreases.

In step 3, the algorithm will stop.

So this algorithm will halt in finitely many steps.

If Ω is empty, then after the first step in which $\Gamma' = \Gamma\lambda_{\tilde{\Lambda}}$, the algorithm goes into the first part of step 2. $\tilde{\Gamma} = \Gamma' = \Gamma\lambda_{\tilde{\Lambda}}$. Because we did nothing to Γ except for applying $\lambda_{\tilde{\Lambda}}$, then $\tilde{\Lambda}' = \tilde{\Lambda}$ and $\tilde{\Gamma} \cup \tilde{\Lambda}'$ is a conservative extension of $\Gamma \cup \Omega \cup \tilde{\Lambda}$.

If Ω is not empty, but there is an equation in Γ' with the form $c'x =^? 0$, then from the first part of the second part in step 2, we will solve x from Γ' and $\tilde{Q}$ and stop. So in this case, finally, there is no x occurring in $\tilde{\Gamma}$. Hence the new $\Gamma' \cup \Lambda' \cup \tilde{Q}$ is a conservative extension of $\Gamma' \cup \Lambda' \cup \{T =^? 0\}$ in this step.

If Ω is not empty and no equation in Γ' has the form $c'x =^? 0$, then if we have $c'x + \mathbf{S} =^? 0$ in it, the second part of the second part in step 2 will remove it and add a new $-c'T + cS =^? 0$. As we proved above, every time we run this step, N_x will decrease until no $c'x + \mathbf{S} =^? 0$ is left in Γ. After that, in step 3, we will add $\tilde{Q}$ into $\tilde{\Gamma}$, so in this case, finally, $\tilde{Q} \in \tilde{\Gamma}$ and except for $\tilde{Q}$, no equation contains x as a pure variable.

In this step, for proving the newer $\Gamma' \cup \Lambda' \cup \tilde{Q}$ is a conservative extension of the old $\Gamma' \cup \Lambda' \cup \tilde{Q}$, it is enough to show that there is a substitution θ, such that

$$\theta \vDash \left\{c'x + \mathbf{S} =^? 0, cx + \mathbf{T} =^? 0\right\} \iff \theta \vDash \left\{c'(-\mathbf{T}) + c\mathbf{S} =^? 0, cx + \mathbf{T} =^? 0\right\}$$

If $c'x\theta + \mathbf{S}\theta = 0$ and $cx\theta + \mathbf{T}\theta = 0$, then $cc'x\theta + c\mathbf{S}\theta = 0$ and $c'cx\theta + c'\mathbf{T}\theta = 0$. For the second equation, we get $cc'x\theta = -c'\mathbf{T}\theta$. Then $cc'x\theta + c\mathbf{S} = -c'\mathbf{T} + c\mathbf{S}\theta = 0$, which means $\theta \vDash c'(-\mathbf{T}) + c\mathbf{S} =^? 0$.

We can prove the other direction similarly.

Thus $\tilde{\Gamma} \cup \tilde{\Lambda}'$ is a conservative extension of $\Gamma \cup \Omega \cup \tilde{\Lambda}$. □

Combining the above two lemmas, we get

Theorem 15. *Let Γ be a unification problem, Q be an equation and $V = \{x \in PV(Q) : x \text{ is an unconstrained variable in } \Gamma\}$.* **UnconstrainedReduce**(Γ, Q, V), *which is* **Uniq**$(\Gamma, $ **Maxi**$(Q, V))$, *will terminate after finitely many steps, and output a pair $(\tilde{\Gamma}, \tilde{\Lambda}')$, which satisfies:*

1. *$\tilde{\Gamma} \cup \tilde{\Lambda}'$ is a conservative extension of $\Gamma \cup \{Q\}$.*
2. • *either some unconstrained variable in Q is solved, or*
   • *there exists an unconstrained variable x in Γ, which is a fresh variable or was in V. x occurs only once in $\tilde{\Gamma}$ and x has the largest absolute value of the coefficient among all the coefficients of terms in the equation in which it occurs.*

Proof. This follows from Lemma 13 and Lemma 14 by restricting V to the unconstrained variables in Γ. □

If **Uncons**(Γ) is used to denote the set: $\{x \in Vars(\Gamma) : x \text{ is an unconstrained variable in } \Gamma\}$ and some variable $x \in \Gamma$ satisfies:

- x occurs only once in Γ, and
- x is the unique unconstrained variable in the equation where x occurs in Γ, and
- the absolute value of coefficient of x is the largest in the equation where x occurs,

we call x a **reduced unconstrained variable** in Γ.

6. Inference system $\mathfrak{I}_{AGH}$

We will use $\mathfrak{I}_{AGH}$ to denote our inference system. It contains five **necessary rules** and seven **auxiliary rules**.

6.1. Problem format

For efficiency and convenience, in our inference procedure we will use a quadruple $\Gamma \| \Delta \| \Lambda \| \Psi$, where $\|$ is used to separate these four sets.

In $\Gamma \| \Delta \| \Lambda \| \Psi$, Γ is a unification problem, a set of the form $\{\mathbf{S_1} =^? 0, \mathbf{S_2} =^? 0, \cdots, \mathbf{S_n} =^? 0\}$, where each $\mathbf{S_i}$ is a PURE SUM. Δ is a set of disequations. Every disequation in Δ has the form $f(s_1, s_2, \cdots, s_n) + (-f(t_1, t_2, \cdots, t_n)) \neq^? 0$ or $0 \neq^? 0$, where f is an uninterpreted symbol from Γ and s_i and t_i are pure terms. Δ is used to track nondeterministic choices in our inference system. Every time we make a choice, we will add a disequation into Δ. Initially, this set is empty.

Λ is a set of equations in solved form. An equation $x =^? S$ will be added to Λ if we apply the substitution $x \longmapsto S$ to Γ. All the equations in Λ have the form $x =^? S$, where x is called a *solved variable* in Γ, which means that x does not occur in Γ. Also we call x is *solved* when $x =^? S$ is put into Λ. Initially, this set is empty.

Before explaining Ψ, let us have a look at an example:

$$3x + h(y) =^? 0$$

There is a solution: $[x \longmapsto h(z), y \longmapsto -3z]$. Since the unconstrained variable's (x) coefficient is greater than that of $h(y)$, we can not solve it by **UnconstrainedReduce** directly. We use the following idea to *guess* (this guess will be proven in Section 10):

- If there is a solution θ, such that $3x\theta + h(y)\theta = 0$, then the greatest common divisor of the coefficients of $y\theta$ must be a multiple of 3. For example, $y \longmapsto 6b + 5a$, no matter what x's substitution is, is not a solution. If $y \longmapsto 3a$, we can let $x \longmapsto h(-a)$ to get the solution. So we can guess the solution of this algorithm has a solution of the form $y \longmapsto 3z$.
- This is not enough, since it is possible the equation contains other terms. For example $3x + 2z + h(y) =^? 0$ (if z occurs in other equations), $y \longmapsto 3w + 2w', x \longmapsto h(-w), z \longmapsto h(-w')$ is a possible solution. So $y\theta$ may contain two parts: the greatest common divisor of the coefficients of one part is a multiple of 3, and the greatest common divisor of the coefficients of the other part is a multiple of 2. So we can guess the solution of this equation has a solution of the form $y \longmapsto 3w + w_1$, then guess $w_1 \longmapsto 2w'$.
- Generally, from the Division Algorithm, if we guess $y \longmapsto kw + w'$, then next, if we need to guess $w' \longmapsto k'w'' + t$, then k' should be less than k.

Ψ is used to remember this k, such that the next time k' is not greater than or equal to k. Technically, Ψ is a set of pairs of the form, which consist of numbers and variables:

$$\{(N_1, x_1), (N_2, x_2), \cdots, (N_n, x_n)\}$$

where x_i is a variable satisfying that $h(x_i)$ occurs in Γ. All the N_is are natural numbers or ∞. If we have a substitution θ, such that

$$(x_i\theta)\downarrow = c_1 s_1 + c_2 s_2 + \cdots + c_k s_k$$

where s_i is a monic term, then we say θ satisfies the pair (N_i, x_i) if $|c_j| < N_i$ for all $1 \leq j \leq k$. If some substitution θ satisfies each pair in Ψ, we say θ satisfies Ψ. So Ψ is used to constrain the form of the solution for the terms. From this meaning, we can know $\{(N_1, x), (N_2, x)\} = \{(N_1, x)\}$ if $N_1 < N_2$.

Initially, we set all N_i to ∞, and write initial Ψ as Ψ^{ini}_{Γ}. Because this restriction only applies to variables under an h symbol, every time some corresponding h-term is removed from Γ, we will remove the corresponding pair.

Sometimes, the term occurring under an h symbol will be changed by our rewriting system and purification procedure. For example, we have $h(x) + h(y) \to h(x + y)$ and in **Purify**, we will introduce a new variable to replace $x + y$ because it is not pure anymore. So suppose we have (N_s, s) and (N_t, t), and we have an equation $h(cs) + h(dt) + S =^? 0$ in Γ. After being rewritten and **Purify**, this equation is replaced by $h(x') + S =^? 0$ and $x' + (-cs) + (-dt) =^? 0$, in this case, we remove (N_s, s) and (N_t, t) from Ψ, and add $(|c|N_s + |d|N_t, x')$ into Ψ. Here, it is possible that s or t is not a variable. In this case, we let N_s or N_t be 1. In **Singlize**, we will remove extra identical (or the inverse of an) h-terms, but will not affect the pairs in Ψ.

For convenience, we call Γ an *equation set*, Δ a *disequation set*, Λ a *solved equation set*, Ψ a *pair set* and $\Gamma \| \Delta \| \Lambda \| \Psi$ a set quadruple. We say a substitution θ satisfies the set quadruple $\Gamma \| \Delta \| \Lambda \| \Psi$, if θ satisfies every equation in Γ and Λ, every disequation in Δ and every pair in Ψ, and write that relation as $\theta \vDash \Gamma \| \Delta \| \Lambda \| \Psi$. Similarly, we call $\Gamma \| \Lambda$ a set pair, and a substitution θ satisfies the set pair $\Gamma \| \Lambda$ if θ satisfies every equation in Γ and Λ. We use **Fail** to be a special set quadruple with no solution.

6.2. Necessary rules

$\mathfrak{I}_{AGH}$ contains five **necessary rules**: Trivial, Variable Substitution, N-Decomposition, Factorization and Annulization; and seven **auxiliary rules** used for efficiency. Trivial, Variable Substitution, Factorization, Annulization and the auxiliary rules are deterministic, which means any branch they choose will not lose any solutions, and N-Decomposition is nondeterministic which means we have to consider all the branches. In our inference procedure, there are four priorities for applying rules. The rules with the highest priority are Trivial, Variable Substitution and the auxiliary rules. They will be applied whenever they can be applied. The rule with the second highest priority is N-Decomposition. The rule with the third highest priority is Factorization. The rule with the lowest priority is Annulization. Rules can only be applied if no rules with higher priority can be applied.

Given a quadruple $\Gamma \| \Delta \| \Lambda \| \Psi$, if no rules can be applied and $\Gamma = \emptyset$, then Λ is a solution. Let $\tilde{\Psi} = \{(\infty, x_1), (\infty, x_2), \cdots, (\infty, x_n)\}$, where $h(x_i)$ occurs in Γ. Then exhaustively applying the inference rules to $\Gamma \| \emptyset \| \emptyset \| \tilde{\Psi}$, yields a set of quadruples in which every element has the form of $\emptyset \| \Delta' \| \Lambda' \| \Psi'$, such that Λ' is an **AGH**-unifier of Γ, **Fail** if Γ is not unifiable, or a stuck quadruples to which no rules are applicable and Γ is not empty. At the end, we will prove the third case is equivalent to **Fail**.

Now we present the inference rules.

6.2.1. Trivial

The first necessary rule is called Trivial, which is used to remove the trivially true equations. It has the highest priority.

Trivial

$$\frac{\Gamma \cup \{0 =^? 0\} \| \Delta \| \Lambda \| \Psi}{\Gamma \| \Delta \| \Lambda \| \Psi}$$

6.2.2. Variable substitution

The second necessary rule is used to solve variables based on the **UnconstrainedReduce** algorithm. It also has the highest priority.

The purpose of this rule is trying to solve the unconstrained variables (if the absolute value of its coefficient can be reduced to 1) or make the absolute value of the coefficient of the unique unconstrained variable in one equation be maximum. The latter case will help us to check whether this equation has no solution (like $3x + a =^? 0$) or we need to guess a solution (like $3x + h(z) =^? 0$), which needs our other inference rules.

Formally, the inference rule is as follows:

Variable Substitution

$$\frac{(\Gamma \cup \{c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m =^? 0\}) \| \Delta \| \Lambda \| \Psi}{\Gamma'\sigma \| \Delta\sigma \| \Lambda'\sigma \| \Psi'\sigma}$$

if each x_i is an unconstrained variable in $\Gamma \cup \{c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m =^? 0\}$ and each t_i is a pure term, but not an unconstrained variable, and

- every t_i is not an h-term, or
- every equation in Γ, containing an unconstrained variable of Γ, contains an h-term.

and

- $n > 1$ or
- $|c_1| \leq |c'_i|$ for some i or
- $x_1 \in \Gamma$.

Suppose the output of **UnconstrainedReduce**$(\Gamma, c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m =^? 0, \{x_1, x_2, \cdots, x_n\})$ is $(\tilde{\Gamma}, \tilde{\Lambda})$, then $\Gamma' = \tilde{\Gamma}$, $\sigma = \lambda_{\tilde{\Lambda}}$ and $\Lambda' = \Lambda \cup \tilde{\Lambda}$. In the conclusion of this rule, if the equation set $\Gamma'\sigma$ is not pure, purification will be applied to $\Gamma'\sigma$. The non-pure term must be $h(s + t)$, which is from $h(s) + h(t)$. Purification will replace $h(s + t)$ by $h(x')$ and add another equation $x' - s - t =^? 0$ into Γ'. The change from Ψ to Ψ' will be according the following rule:

$$\frac{\Gamma \cup \{S + h(s + t) =^? 0\} \| \Delta \| \Lambda \| \{(N_s, s), (N_t, t)\} \cup \Psi}{\Gamma \cup \{S + h(x) =^? 0\} \cup \{x + (-s) + (-t) =^? 0\} \| \Delta \| \Lambda \| \{(N_s + N_t, x)\} \cup \Psi}$$

Here s and t are variables and x is a fresh variable.

Here we give a simple example to explain how this rule works.

Example 16. In the equation set:

$$\left\{4x+2y+z=^? 0, 2y+5x+f(z)=^? 0\right\}$$

y and x are both unconstrained variables. Because there are no h-terms, for convenience, we will omit Δ, Λ and Ψ.

We choose the first one to apply Variable Substitution. After applying it we get the new equation set:

$$\left\{2v_1+z=^? 0, x+f(z)+(-z)=^? 0\right\}$$

where $y \longmapsto v_1+(-2x)$.

Here the first equation is unconstrained reduced already. We choose the second one to apply Variable Substitution and get

$$\left\{2v_1+z=^? 0\right\}$$

where $x \longmapsto z+(-f(z))$. Now z is unconstrained, and we get $z \longmapsto -2v_1$. So the final solution is

$$\left[x \longmapsto -2v_1+\left(-f(2v_1)\right), y \longmapsto 5v_1+2f(2v_1), z \longmapsto -2v_1\right]$$

The condition that every t_i is not an h-term or every equation containing an unconstrained variable of Γ, contains an h-term, is used to avoid looping. For example:

Example 17. For convenience, we will omit Λ, Δ and Ψ here. For the equation set:

$$\left\{x+h(y)=^? 0, x+z+h(z)=^? 0, x+y+z=^? 0\right\}$$

If we apply Variable Substitution without obeying the conditions, we will go into a loop. For example, if we choose $x+h(y)=^? 0$ first, applying Variable Substitution, we get $x \longmapsto h(-y)$ and a new equation set

$$\left\{z+h(v_1)=^? 0, v_1+y-z=^? 0, y+z+h(-y)=^? 0\right\}$$

where v_1 is generated by purification.

Because z is unconstrained now, we choose $z+h(v_1)=^? 0$ to apply Variable Substitution, we get $z \longmapsto h(-v_1)$ and a new equation set

$$\left\{v_1+y+h(v_1)=^? 0, y+h(v_2)=^? 0, v_2+v_1+y=^? 0\right\}$$

This equation set is the same as the original equation set except for different variable names. This will go into a loop.

If we obey our condition, we choose $x+y+z=^? 0$ and solve x first, we get $x \longmapsto -y+(-z)$ and a new equation set:

$$\left\{-y+(-z)+h(y)=^? 0, -y+h(z)=^? 0\right\}$$

We have to stop here because we have no rules to solve it so far. We will give the solution of this example in Example 20.

The reason we only apply our algorithm on unconstrained variables is to control looping. For example, suppose we have $\{x+f(z)=^? 0, z+x+y=^? 0\}$. If we apply our algorithm on the constrained variable z, we get $z \longmapsto -x+(-y)$, then $x+f(z)=^? 0$ becomes $x+f(-x+(-y))$. After purification, we will get $\{x+f(v_1)=^? 0, v_1+x+y=^? 0\}$, which is the same as original problem set up to variable renaming. In Section 9, we will see these conditions will not lose any solutions.

6.2.3. N-Decomposition

The next inference rule N-Decomposition is nondeterministic. This is to solve a case, like $f(x)+f(y)+f(z)=^? 0$. In this case there are several possible guesses: $\{f(x)=f(y)\neq f(z)\}$, $\{f(z)=f(y)\neq f(x)\}$, $\{f(x)=f(z)\neq f(y)\}$, $\{f(x)\neq f(x), f(x)\neq f(y), f(y)\neq f(z)\}$, and $\{f(x)=f(y)=f(z)\}$. Since our problems is in PURE SUM form, these guesses are several syntactical unification problems which we will use Decomposition method to solve, that's why this rule is called N-Decomposition.

When we apply N-Decomposition, we get two new independent problems, whose combined solutions are the solutions of the original problem. We will use $\bigvee$ between these two problems.

N-Decomposition

$$\frac{(\Gamma \cup \{\mathbf{S} + cf(s_1, \cdots, s_m) + (-df(t_1, \cdots, t_m)) =^? 0\}) \| \Delta \| \Lambda \| \Psi}{((\Gamma\sigma \cup \{Q\}\sigma) \| (\Delta\sigma) \| (\Lambda\sigma \cup [\sigma]) \| \Psi\sigma) \vee (\Gamma_1' \| \Delta_1' \| \Lambda \| \Psi)}$$

if $c > 0$, $d > 0$ and $f(s_1, s_2, \cdots, s_m) + (-f(t_1, t_2, \cdots, t_m)) \neq^? 0 \notin \Delta$, where

$$\sigma = mgu\big(s_1 =^? t_1, s_2 =^? t_2, \cdots, s_m =^? t_m\big)$$
$$\Gamma_1' = \Gamma \cup \big\{\mathbf{S} + cf(s_1, s_2, \cdots, s_m) + \big(-df(t_1, t_2, \cdots, t_m)\big) =^? 0\big\}$$
$$\Delta_1' = \Delta \cup \big\{f(s_1, s_2, \cdots, s_m) + \big(-f(t_1, t_2, \cdots, t_m)\big) \neq^? 0\big\}$$
$$Q = \mathbf{S} + (c-d)f(s_1, s_2, \cdots, s_m) =^? 0$$

f is an uninterpreted function symbol.

Note: After applying the N-Decomposition rule, we might make two h-terms identical. In this case, **Singlize** will be applied.

Because we do not know whether a possible solution will make two f-terms equal or not, we need to think of both possible cases. Here $\bigvee$ means we convert the original quadruple into two possible quadruples.

Example 18. For the equation set:

$$\big\{x + 2f(t) + \big(-2f(z)\big) =^? 0, y + 3f(x) + \big(-3f(w)\big) =^? 0\big\}$$

we can do nothing via Variable Substitution, so choose two of the pure terms in one equation to apply the N-Decomposition rule. There are no h-terms in it, so $\Psi = \emptyset$

$$\big\{x + 2f(y) + \big(-2f(z)\big) =^? 0, y + 3f(x) + \big(-3f(w)\big) =^? 0\big\} \| \emptyset \| \emptyset \| \emptyset$$
$$\Longrightarrow \text{ (N-Decomposition)}$$
$$\big\{x =^? 0, z + 3f(x) + \big(-3f(w)\big) =^? 0\big\} \| \emptyset \| \big\{y =^? z\big\} \| \emptyset \bigvee \Gamma_1 \| \Delta_1 \| \emptyset \| \emptyset$$
$$\overset{*}{\Rightarrow} \text{ (several steps by Variable Substitution)}$$
$$\emptyset \| \emptyset \| \big\{x =^? 0, y =^? 3f(w) + \big(-3f(0)\big), z =^? 3f(w) + \big(-3f(0)\big)\big\} \| \emptyset \bigvee \Gamma_1 \| \Delta_1 \| \emptyset \| \emptyset$$

where

$$\Gamma_1 = \big\{x + 2f(y) + \big(-2f(z)\big) =^? 0, y + 3f(x) + 3f(w) =^? 0\big\}$$
$$\Delta_1 = \big\{f(y) + \big(-f(z)\big) \neq^? 0\big\}$$

So from the first part, we get a solution $\sigma_1 = [x \longmapsto 0, y \longmapsto 3f(w) - 3f(0), z \longmapsto 3f(w) - 3f(0)]$.

We apply N-Decomposition to $\Gamma_1 \| \Delta_1 \| \emptyset \| \emptyset$, and use a similar procedure to get the second solution $\sigma_2 = [x \longmapsto 2f(z) - 2f(0), y \longmapsto 0, w \longmapsto 2f(z) - 2f(0)]$ (We will prove Δ can be ignored at the end.) and another branch of our procedure is

$$\Gamma_1 = \big\{x + 2f(y) + \big(-2f(z)\big) =^? 0, y + 3f(x) + 3f(w) =^? 0\big\}$$
$$\Delta_1 = \big\{f(y) + \big(-f(z)\big) \neq^? 0, f(x) + \big(-f(w)\big) =^? 0\big\}$$

So σ_1 and σ_2 are two solutions to this problem. The third branch fails, because no inference rule can be applied to it.

6.2.4. Factorization

Next we introduce one of the necessary rules with the third priority. Factorization is used to solve a case like $3x + h(z) = 0$. It cannot be solved by Variable Substitution, because 3 is the biggest absolute value among the coefficients in this equation. We guess that z has the form $z \longmapsto 3z'$, since $[x \longmapsto z', z \longmapsto 3z']$ is a solution of $3x + h(z) = 0$.

Factorization makes a more general guess as follows:

Factorization

If

- Trivial, Variable Substitution, N-Decomposition and other auxiliary rules cannot be used, and
- x does not occur under an uninterpreted function symbol, and
- $n > |c|$ and y is an unconstrained variable,

then

$$\frac{(\Gamma \cup \{cy + \mathbf{S} + dx + h(x) =^? 0\}) \| \Delta \| \Lambda \| (\Psi \cup \{(n, x)\})}{(\Gamma \cup \{cy' + \mathbf{S} + dx'' + h(x'') =^? 0\})\sigma \| \Delta\sigma \| \Lambda\sigma \cup [\sigma] \| (\Psi \cup \{(|c|, x'')\})\sigma}$$

where $\sigma = [x \longmapsto cx' + x'', y \longmapsto y' + (-dx') + h(-x')]$, d can be 0 and x', x'' and y' are fresh variables.

Example 19. For the equation set

$$\{2y + x_2 + h(x_1) =^? 0, 3z + x_1 + h(x_2) =^? 0\}$$

we can only apply Factorization since we have no other rules applicable. There are two h-terms in it, so initially, $\Psi = \{(\infty, x_1), (\infty, x_2)\}$.

$$\{2y + x_2 + h(x_1) =^? 0, 3z + x_1 + h(x_2) =^? 0\} \| \emptyset \| \emptyset \| \{(\infty, x_1), (\infty, x_2)\}$$
$$\Longrightarrow \text{(Factorization)}$$
$$\{2v_3 + x_2 + h(v_2) =^? 0, 3z + 2v_1 + v_2 + h(x_2) =^? 0\} \| \emptyset \|$$
$$\{x_1 =^? 2v_1 + v_2, y =^? v_3 + h(-v_1)\} \| \{(2, v_2), (\infty, x_2)\}$$
$$\Longrightarrow \text{(Variable Substitution on the second equation)}$$
$$\{2v_3 + x_2 + h(v_2) =^? 0\} \| \emptyset \|$$
$$\{x_1 =^? 6v_4 + 3v_2 + h(2x_2),\ y =^? v_3 + h(-3v_4 + (-v_2) + h(-x_2)), v_1 =^? 3v_4 + v_2 + h(x_2),$$
$$z =^? -2v_4 + (-v_2) + h(-x_2)\} \| \{(2, v_2)\}$$
$$\Longrightarrow \text{(Variable Substitution)}$$
$$\emptyset \| \emptyset \| \{x_1 =^? 6v_4 + 3v_2 + h(-4v_3 + h(-2v_2)), y =^? v_3 + h(-3v_4 + (-v_2) + h(2v_3 + h(v_2))),$$
$$v_1 =^? 3v_4 + v_2 + h(-2v_3 + h(-v_2)), x_2 =^? -2v_3 + h(-v_2),\ z =^? -2v_4 + (-v_2) + h(2v_3 + h(v_2))\} \| \emptyset$$

Then we get the solution set

$$\{x_1 \longmapsto 6v_4 + 3v_2 + h(-4v_3 + h(-2v_2)), y \longmapsto v_3 + h(-3v_4 + (-v_2) + h(2v_3 + h(v_2))),$$
$$x_2 \longmapsto -2v_3 + h(-v_2), z \longmapsto -2v_4 + (-v_2) + h(2v_3 + h(v_2))\}$$

The purpose of $n > |c|$ is to avoid loops, we will see this case in Example 21 which is after the next rule. In the next section, we will prove this constraint will not lose any solutions.

6.2.5. Annulization

The next rule is based on the homomorphism property $h(0) =_{AGH} 0$ and it has the lowest priority in our inference system. It solves a case like $x + h(x) = 0$, the only possible solution is making $h(x)$ be 0. Formally:

Annulization

$$\frac{(\Gamma \cup \{\mathbf{S} + h(t) =^? 0\}) \| \Delta \| \Lambda \| (\Psi \cup \{(n, t)\})}{\Gamma \cup \{\mathbf{S} =^? 0\} \cup \{t =^? 0\} \| \Delta \| \Lambda \| \Psi}$$

if there is no term with some uninterpreted function symbol on the top occurring in Γ and $\mathbf{S}$, and $\mathbf{S}$ may be empty.

First we solve the problem in Example 17.

Example 20. For convenience, we still do not include Δ, Λ and Ψ here.

For equation set

$$\{x + h(y) =^? 0, x + z + h(z) =^? 0, x + y + z =^? 0\},$$

we can apply Variable Substitution on $x + y + z =^? 0$ and get $x \longmapsto -y + (-z)$, and our equations become

$$\{-y + (-z) + h(y) =^? 0, -y + h(z) =^? 0\}.$$

Here, we have no rules to be applied except Annulization, since there are no unconstrained variables in it. So we set $y \longmapsto 0$ and get

$$\{-z =^? 0, h(z) =^? 0\}.$$

We still only can apply Annulization on it and set $z \longmapsto 0$ and get

$$\{0 =^? 0\}.$$

Using Trivial, we get the empty set and the solution

$$\{x \longmapsto 0, y \longmapsto 0, z \longmapsto 0\}$$

We use the next example to show the constraint in Factorization is necessary:

Example 21. For equation

$$3y + 2x + h(x) =^? 0$$

we can only apply Factorization because no other rules with higher priority are applicable. Since there is only one h-term, initially, $\Psi = \{(\infty, x)\}$.

$$\{3y + 2x + h(x) =^? 0\} \| \emptyset \| \emptyset \| \{(\infty, x)\}$$
$$\Longrightarrow \text{(Factorization)}$$
$$\{3v_3 + 2v_2 + h(v_2) =^? 0\} \| \emptyset \| \{x \longmapsto 3v_1 + v_2, y \longmapsto v_3 + (-2v_1) + h(-v_1)\} \| \{(3, v_2)\}$$
$$\Longrightarrow \text{(Annulization)}$$
$$\{3v_3 =^? 0\} \| \emptyset \| \{x \longmapsto 3v_1, y \longmapsto v_3 + (-2v_1) + h(-v_1), v_2 \longmapsto 0\} \| \emptyset$$
$$\Longrightarrow \text{(Annulization)}$$
$$\emptyset \| \{x \longmapsto 3v_1, y \longmapsto -2v_1 + h(-v_1), v_2 \longmapsto 0, v_3 \longmapsto 0\} \| \emptyset$$

So we get the solution $\{x \longmapsto 3v_1, y \longmapsto -2v_1 + h(-v_1)\}$.

From this example, we can see the condition $n > c$ is very important, or the procedure of Factorization will go on forever.

7. Simplifier

In the next section, we will give the proof details of the termination, soundness and completeness of our inference system. Before that, we give some notation and definitions about an abstract inference rule called *Simplifier*, which covers all of our concrete auxiliary rules, given later. Before giving the proof, we define directed conservative extension:

Definition 22 *(Directed conservative extension).* Let $\bigvee_i(\Gamma_i \| \Delta_i \| \Lambda_i \| \Psi_i)$ and $\bigvee_i(\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$ be two set quadruples. $\bigvee_i(\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$ is called a *directed conservative extension* of $\bigvee_i(\Gamma_i \| \Delta_i \| \Lambda_i \| \Psi_i)$, if for any substitution θ, such that $\theta \vDash \Gamma_i \| \Delta_i \| \Lambda_i \| \Psi_i$, then there exists j and σ, whose domain is the variables in $Vars(\Gamma_j' \cup \Lambda_j') \setminus Vars(\Gamma_i \cup \Lambda_i)$, such that $\theta\sigma \vDash \Gamma_j' \| \Delta_j' \| \Lambda_j' \| \Psi_j'$. If $\bigvee_i(\Gamma_i \| \Delta_i \| \Lambda_i \| \Psi_i)$ (resp. $\bigvee_i(\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$) only contains one quadruple $\Gamma \| \Delta \| \Lambda \| \Psi$ (resp. $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$), we say $\bigvee_i(\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$ (resp. $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$) is directed conservative extension of $\Gamma \| \Delta \| \Lambda \| \Psi$.

From the definition, we can see this is one direction of conservative extension, but with more conditions. We will use it to prove the inference rules never lose any solutions.

We give two mappings here: P and μ. We call $P : \mathcal{P}(\Gamma) \to \{True, False\}$ a *property* of Γ. and $\mu : \mathcal{P}(\Gamma \| \Delta \| \Lambda \| \Psi) \to \mathbf{N}$ a *measure* of $\Gamma \| \Delta \| \Lambda \| \Psi$, where $\mathbf{N}$ is a well-ordered set.

Definition 23 *(Preserving rules and simplifiers).* Let $=_E$ be an equational theory, where E is a set of identities. Let I be an inference rule of the following form:

$$\frac{\Gamma\|\Delta\|\Lambda\|\Psi}{\Gamma'\|\Delta'\|\Lambda'\|\Psi'}$$

If $\Gamma\|\Delta\|\Lambda\|\Psi$ has the same set of solutions as $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ and every solution of $\Gamma'\|\Lambda'$ is a solution of $\Gamma\|\Lambda$, and $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ is a directed conservative extension of $\Gamma\|\Delta\|\Lambda\|\Psi$, we say I is *E-equation preserving.* If for some property P, $P(\Gamma')$ is true whenever $P(\Gamma)$ is true, we say I is *P-preserving.* If μ is a measure such that $\mu(\Gamma\|\Delta\|\Lambda\|\Psi) > \mu(\Gamma'\|\Delta'\|\Lambda'\|\Psi')$, we say I is *μ-reducing.* If I is E-Equation Preserving, P-preserving and μ-reducing, we say I is a *(P, E, μ)-Simplifier.*

If P, E and μ are clear, we will write it as *simplifier.*

8. Termination

For two set quadruples, $\Gamma\|\Delta\|\Lambda\|\Psi$ and $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$, we will use the following notation in the following sections.

- $\Gamma\|\Delta\|\Lambda\|\Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\Psi'$, means $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ is deduced from $\Gamma\|\Delta\|\Lambda\|\Psi$ by applying a rule from $\mathfrak{I}_{AGH}$ once (we call it one *step*).
- $\Gamma\|\Delta\|\Lambda\|\Psi \overset{*}{\Rightarrow}_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\Psi'$, means $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ is deduced from $\Gamma\|\Delta\|\Lambda\|\Psi$ by zero or more steps.
- $\Gamma\|\Delta\|\Lambda\|\Psi \overset{+}{\Rightarrow}_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\Psi'$ means $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ is deduced from $\Gamma\|\Delta\|\Lambda\|\Psi$ by one or more steps.

Note that for N-Decomposition, $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ could represent either of the choices.

As we can see in the inference rules, N-Decomposition divides $\Gamma\|\Delta\|\Lambda\|\Psi$ into two parts, $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ and $\Gamma''\|\Delta''\|\Lambda''\|\Psi''$. So for a quadruple $\Gamma\|\Delta\|\Lambda\|\Psi$, after applying some inference rules, the result may be a disjunction of quadruples $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, not just one triple. So we give the following notation:

- $\Gamma\|\Delta\|\Lambda\|\Psi \Rrightarrow_{\mathfrak{I}_{AGH}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, where $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$ is a disjunction of quadruples, means after applying some inference rule to $\Gamma\|\Delta\|\Lambda\|\Psi$ once, $\Gamma\|\Delta\|\Lambda\|\Psi$ becomes $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$.
- $\Gamma\|\Delta\|\Lambda\|\Psi \overset{+}{\Rrightarrow}_{\mathfrak{I}_{AGH}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, where $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$ is a disjunction of quadruples, means after applying some inference rules once or more than once, $\Gamma\|\Delta\|\Lambda\|\Psi$ becomes $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, namely $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$ is the set of all branches we get after applying one or more than once the rules in $\mathfrak{I}_{AGH}$ to $\Gamma\|\Delta\|\Lambda\|\Psi$.
- $\Gamma\|\Delta\|\Lambda\|\Psi \overset{*}{\Rrightarrow}_{\mathfrak{I}_{AGH}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, where $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$ is a disjunction of quadruples, means after zero or more steps, $\Gamma\|\Delta\|\Lambda\|\Psi$ becomes $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, namely $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$ is the set of all branches we get after applying zero or more times the rules in $\mathfrak{I}_{AGH}$ to $\Gamma\|\Delta\|\Lambda\|\Psi$.

Next, we give seven measures for proving termination:

- Let $H(\Gamma)$ be the number of h-terms in Γ. Since $H(\Gamma)$ is a natural number, $H(\Gamma)$ with standard $\leq$ on natural numbers is a well-founded ordering.
- Let $Coe(\Psi)$ be a multiset, $\{n_i : (n_i, x_i) \in \Psi\}$. Since every n_i is a natural number, the multiset order for $Coe(\Psi)$ is a well-founded ordering.
- Let $Vars(\Gamma) = \{x \mid x$ occurs in some equation in $\Gamma\}$. Since $|Vars(\Gamma)|$ can only be natural number, $|Vars(\Gamma)|$ with standard less than or equal on natural numbers is a well-founded ordering.
- $Cons(\Gamma)$ is the set of all constrained variables in Γ. Since $|Cons(\Gamma)|$ is natural number, $|Cons(\Gamma)|$ with standard $\leq$ on natural numbers is a well-founded ordering.
- We use $Reduce(\Gamma)$ to denote the set

  $$\{x \in \Gamma : x \text{ is a reduced unconstrained variable in } \Gamma\}.$$

  Let $RUncons(\Gamma)$ be $Uncons(\Gamma) \setminus Reduce(\Gamma)$. $|RUncons(\Gamma)|$ is non-negative because $|Uncons(\Gamma)|$ is always greater than or equal to $|Reduce(\Gamma)|$. So $|RUncons(\Gamma)|$ with standard $\leq$ on natural numbers is a well-founded ordering.
- Recall that $Sym(\Gamma)$ is the multiset of all symbols occurring in Γ. Obviously, the standard ordering of $|Sym(\Gamma)|$ based on natural numbers is a well-founded ordering on the set of equations.
- From the definition of Δ, we see that Δ only contains disequations of the form $s + (-t) \neq^? 0$ where s and t both have the same top function symbol and the coefficients of s and t are 1. So we let $Par(\Delta)$ be $\{(t, s) : t + (-s) \neq^? 0 \in \Delta\}$ and $Par(\Gamma)$ be $\{(t, s) : t, s$ occurs in Γ and t has the same top function symbol as $s\}$. We use $Par(\Gamma \setminus \Delta)$ to denote $Par(\Gamma) \setminus Par(\Delta)$. Since the number of terms in Γ is positive and finite, $|Par(\Gamma \setminus \Delta)|$ with standard $\leq$ on natural numbers is a well-founded ordering. This measure is used to count all possible disequations that could be placed in Δ by N-Decomposition, not including the ones that are already there.

We then define the measure of $\Gamma\|\Delta\|\Lambda\|\Psi$ as following:

$$\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)=\big(H(\Gamma),Coe(\Psi),\big|Vars(\Gamma)\big|,\big|Cons(\Gamma)\big|,\big|RUncons(\Gamma)\big|,\big|Sym(E)\big|,\big|Par(\Gamma\setminus\Delta)\big|\big).$$

The lexicographical order on this tuple is well founded because each element of this tuple with its corresponding order is well founded.

Since we apply purification whenever Γ is not a PURE SUM, we can always assume that the equation set Γ is a PURE SUM form before applying the inference rules.

We claim here that $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)$ decreases by every inference rule.

Lemma 24. *Let $\Gamma\|\Delta\|\Lambda\|\Psi$ and $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ be two quadruple sets, where Γ and Γ' are in* PURE SUM *form, such that $\Gamma\|\Delta\|\Lambda\|\Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\Psi'$, Then, $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)>\mathbb{M}_{AGH}(\Gamma',\Delta',\Lambda',\Psi')$*

Proof. Trivial: $Sym(\Gamma)$ decreases but other components do not change, so $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)>\mathbb{M}_{AGH}(\Gamma',\Delta',\Lambda',\Psi)$.

Variable Substitution: We already explained that the **UnconstrainedReduce** algorithm will halt. Here we only talk about the status of applying Variable Substitution.

According to the conditions for applying Variable Substitution, we only apply this rule on the variables which are not in $Reduce(\Gamma)$. During **UnconstrainedReduce**, if some unconstrained variable's coefficient's absolute value is one, this variable will be solved and no new variable will be introduced, hence $|Vars(\Gamma)|$ decreases.

If there is no h-term in this solved equation, then $H(\Gamma)$ and $Coe(\Psi)$ will not change. Now suppose there is an h-term in this equation, which has the form $\mathbf{S}'+x+h(t)=^? 0$, where x is the variable solved in this step, then every equation which contains unconstrained variables also has an h-term in it, which means all the equations $dx+h(t')+\mathbf{S}''$ will become $-\mathbf{dS}'+h(-dt)+h(t')+\mathbf{S}''$. In this case, our rewriting rules and **Purify** will be applied immediately. i.e. $-\mathbf{dS}'+h(-dt)+h(t')+\mathbf{S}''=^? 0$ will be rewritten to $h(-dt+t')+-\mathbf{dS}'+\mathbf{S}''=^? 0$ and then be purified to $\{h(x')+-\mathbf{dS}'+\mathbf{S}'', x'+dt+(-t')=^? 0\}$, where x' is a fresh variable. From this procedure, we see that $H(\Gamma)=H(\Gamma')+1$. If some solved variable makes two h-terms identical, **Singlize** will be applied. But in **Singlize**, we keep removing identical h-terms, so $H(\Gamma)$ decreases. So in both cases, $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)>\mathbb{M}_{AGH}(\Gamma',\Delta',\Lambda',\Psi)$.

If x's coefficient is -1, i.e. the form is $\mathbf{S}'+(-x)+h(t)=^? 0$, we can use the similar argument to above to get the same result.

If no unconstrained variable's coefficient's absolute value is one, some variable was introduced (it is an unconstrained variable), we will solve some preexisting variable and this new variable will still be an unconstrained variable. So one unconstrained variable will be replaced by another fresh unconstrained variable. Hence $|Vars(\Gamma)|$ will not increase.

If the equation to which we choose to apply Variable Substitution has no h-term in it, then $|H(\Gamma)|$ and $Coe(\Psi)$ will not increase. If the equation to which we choose to apply Variable Substitution has h-term in it, since terms are reduced by $\mathfrak{R}_{AGH}$, we know the only possible h-term form is a monic h-term, $h(t)$, where t is a monic form. Since no unconstrained variable's coefficient is one, suppose our equation is $c_1x_1+\cdots+c_nx_n+h(t)+c'_1t_1+\cdots+c'_mt_m=^? 0$, where x_j are unconstrained variables and no t_j is an unconstrained variable. If we choose to use a fresh variable x' to replace x_i, from our algorithm, we get:

$$\begin{aligned} x' &=^? \left\lfloor\frac{c_1}{c_i}\right\rfloor x_1+\cdots\left\lfloor\frac{c_{i-1}}{c_i}\right\rfloor x_{i-1}+x_i+\left\lfloor\frac{c_{i+1}}{c_i}\right\rfloor x_{i+1}+\cdots+\left\lfloor\frac{c_n}{c_i}\right\rfloor x_n+\left\lfloor\frac{1}{c_i}\right\rfloor h(t)+\left\lfloor\frac{c'_1}{c_i}\right\rfloor t_1+\cdots+\left\lfloor\frac{c'_m}{c_i}\right\rfloor t_m \\ &= \left\lfloor\frac{c_1}{c_i}\right\rfloor x_1+\cdots\left\lfloor\frac{c_{i-1}}{c_i}\right\rfloor x_{i-1}+x_i+\left\lfloor\frac{c_{i+1}}{c_i}\right\rfloor x_{i+1}+\cdots+\left\lfloor\frac{c_n}{c_i}\right\rfloor x_n+\left\lfloor\frac{c'_1}{c_i}\right\rfloor t_1+\cdots+\left\lfloor\frac{c'_m}{c_i}\right\rfloor t_m \end{aligned}$$

So we can see that the solved preexisting unconstrained variables will not contain any h-terms. $|H(\Gamma)|$ and $Coe(\Psi)$ will not increase.

We know **UnconstrainedReduce** will finally terminate when some new reduced unconstrained variable is generated. So $|Reduce(\Gamma)|$ increases. In this case, if no constrained variable becomes an unconstrained variable because of the algorithm, then $|Uncons(\Gamma)|$ will not increase. Thus $RUncons(\Gamma)$ decreases. If some constrained variable becomes an unconstrained variable, then $|Cons(\Gamma)|$ will decrease. In both cases, $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)>\mathbb{M}_{AGH}(\Gamma',\Delta',\Lambda',\Psi')$.

N-Decomposition: For Choice 1, in the decomposition rules, σ is $mgu(s_1=^? t_1, s_2=^? t_2,\cdots,s_n=^? t_n)$, which will not introduce new variables. If some variables were solved, and these variables make some h-terms zero, then $H(\Gamma)$ decreases. If no variable makes an h-term zero, but makes two h-terms identical, **Singlize** will be applied. In **Singlize**, we keep removing the identical h-terms, so $H(\Gamma)$ decreases. If no variable makes an h-term zero or two h-terms identical, then $H(\Gamma)=H(\Gamma')$, $Coe(\Psi)=Coe(\Psi')$ and $|Vars(\Gamma)|\geq|Vars(\Gamma')|$. If no variable was solved, then at least two fs are removed, which means $|Sym(\Gamma)|>|Sym(\Gamma')|$. Hence $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)>\mathbb{M}_{AGH}(\Gamma',\Delta',\Lambda',\Psi)$.

For Choice 2, no variable will be introduced, and no symbol in Γ is removed. But some disequality will be added into Δ, while two possible pairs are removed from Γ, which means $|Par(\Gamma)\setminus Par(\Delta)|=|Par(\Gamma')\setminus Par(\Delta')|+1$. Hence $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)>\mathbb{M}_{AGH}(\Gamma',\Delta',\Lambda',\Psi')$.

Factorization: In this rule, some variable x is replaced by a new variable y. Because we did not remove any h-term, $H(\Gamma)$ does not change. We replaced (n,x) by (c,y), where $c<n$, so $Coe(\Psi)$ decreases. Hence $\mathbb{M}_{AGH}(\Gamma,\Delta,\Lambda,\Psi)>\mathbb{M}_{AGH}(\Gamma',\Delta',\Lambda',\Psi')$.

Annulization: We removed an h-term, so $H(\Gamma)$ decreases. Hence $\mathbb{M}_{AGH}(\Gamma, \Delta, \Lambda, \Psi) > \mathbb{M}_{AGH}(\Gamma', \Delta', \Lambda', \Psi')$.
From the definition of Simplifier, we know $\mathbb{M}_{AGH}(\Gamma, \Delta, \Lambda, \Psi) > \mathbb{M}_{AGH}(\Gamma', \Delta', \Lambda', \Psi')$ for Simplifiers. □

Theorem 25. *For any quadruple set $\Gamma \| \Delta \| \Lambda \| \Psi$, there is a quadruple set $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$ such that $\Gamma \| \Delta \| \Lambda \| \Psi \overset{*}{\Rightarrow}_{\mathfrak{I}_{AGH}} \Gamma' \| \Delta' \| \Lambda' \| \Psi'$ and no rules in $\mathfrak{I}_{AGH}$ can be applied on $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$.*

Proof. This statement can be proven from Lemma 24 by induction. □

9. Soundness

The following lemma and theorem justify soundness disregarding Δ at the end of the inference procedure.

Lemma 26. *For two set quadruples $\Gamma \| \Delta \| \Lambda \| \Psi$ and $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$ satisfying*

$$\Gamma \| \Delta \| \Lambda \| \Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma' \| \Delta' \| \Lambda' \| \Psi',$$

let θ be a substitution such that $\theta \vDash \Gamma' \| \Lambda'$. Then $\theta \vDash \Gamma \| \Lambda$.

Proof. It is easy to prove the cases of Trivial and Simplifiers. So we only discuss the other cases:
Variable Substitution: Referring to Theorem 15, we know $\tilde{\Gamma} \cup \tilde{\Lambda}$ is a conservative extension of

$$\Gamma \cup \left\{c_1 x_1 + \cdots + c_n x_x + c'_1 t_1 + \cdots + c'_m t_m =^? 0\right\} \cup \Lambda.$$

But $\Gamma' = \tilde{\Gamma}$, $\sigma = \lambda_{\tilde{\Lambda}}$ and $\Lambda' = \Lambda \cup \tilde{\Lambda}$, so $\Gamma'\sigma \cup \Lambda'\sigma$ is also a conservative extension of

$$\Gamma \cup \left\{c_1 x_1 + \cdots + c_n x_x + c'_1 t_1 + \cdots + c'_m t_m =^? 0\right\} \cup \Lambda.$$

So the statement is true.
N-Decomposition: First Choice:

$$\frac{(\Gamma \cup \{\mathbf{S} + cf(s_1, s_2, \cdots s_m) + (-df(t_1, t_2, \cdots t_m)) =^? 0\}) \| \Delta \| \Lambda \| \Psi}{(\Gamma'\sigma) \| (\Delta\sigma) \| (\Lambda\sigma \cup \{[\sigma]\}) \| \Psi}$$

where $\Gamma' = \Gamma \cup \{\mathbf{S} + (c-d)f(s_1, s_2, \cdots, s_m) =^? 0\}$, and $\sigma = mgu(s_1 =^? t_1, s_2 =^? t_2, \cdots, s_m =^? t_m)$.
If $\theta \vDash \Gamma \cup \{\mathbf{S} =^? 0\} \| (\Lambda\sigma \cup [\sigma]) \| \Psi$, what we need to do is prove $\theta \vDash (\Gamma \cup \{\mathbf{S} + cf(s_1, s_2, \cdots s_m) + (-df(t_1, t_2, \cdots t_m)) =^? 0\}) \| \Lambda$. Because $\theta \vDash [\sigma]$, where $[\sigma] = \{s_1 =^? t_1, s_2 =^? t_2, \cdots, s_m =^? t_m\}$, we know $s_i\theta = t_i\theta$ and $\theta\sigma = \theta$, where $1 \le i \le m$. Then $f(s_1\theta, s_2\theta, \cdots, s_m\theta) = f(t_1\theta, t_2\theta, \cdots, t_m\theta)$. Because $\theta \vDash (\mathbf{S} + (c+d)f(s_1, s_2, \cdots, s_m) =^? 0\sigma)$ and $\sigma\theta = \theta$, we have

$$\begin{aligned}
\mathbf{S}\theta + (c-d)f(s_2, s_2, \cdots, s_m)\theta &= \mathbf{S}(\theta) + cf(s_1, s_2, \cdots, s_m)\theta + \left(-df(s_1, s_2, \cdots, s_m)\right)\theta \\
&= \mathbf{S}(\theta) + cf(s_1, s_2, \cdots, s_m)\theta + \left(-df(s_1\theta, s_2\theta, \cdots, s_m\theta)\right) \\
&= \mathbf{S}(\theta) + df(s_1, s_2, \cdots, s_m)\theta + \left(-df(t_1\theta, t_2\theta, \cdots, t_m\theta)\right) \\
&= 0.
\end{aligned}$$

So this statement is true.
The second choice is obvious.
Factorization: It is enough to show if

$$\theta \vDash \left\{cy' + \mathbf{S} + dx'' + h\left(x''\right) =^? 0,\ x =^? cx' + x'',\ y =^? y' + \left(-dx'\right) + \left(h\left(-x'\right)\right)\right\}$$

then $\theta \vDash \{cy + \mathbf{S} + dx + h(x) =^? 0\}$.
Because $x\theta = cx'\theta + x''\theta$ and $y\theta = y'\theta + (-dx')\theta + (h(-x'))\theta$, we get $x''\theta = x\theta + (-cx'\theta)$ and $y'\theta = y\theta + dx'\theta + h(x')\theta$. Thus

$$\begin{aligned}
0 &= cy'\theta + \mathbf{S}\theta + dx''\theta + h\left(x''\right)\theta \\
&= c\left(y\theta + dx'\theta + h\left(x'\right)\theta\right) + \mathbf{S}\theta + d\left(x\theta + \left(-cx'\theta\right)\right) + h\left(x\theta + \left(-cx'\theta\right)\right) \\
&= cy\theta + cdx'\theta + h\left(cx'\right)\theta + \mathbf{S}\theta + dx\theta + \left(-cdx'\theta\right) + h(x\theta) + \left(h\left(-cx'\theta\right)\right) \\
&= cy + \mathbf{S} + dx + h(x)
\end{aligned}$$

So the statement is true for Factorization.
Annulization: We only need to show $\theta \vDash t =^? 0$ implies $\theta \vDash h(t) =^? 0$. This is true from our rewriting rules.
From the definition of Simplifier, we know the statement is true for Simplifiers. □

Theorem 27. *For any two set quadruples $\Gamma\|\Delta\|\Lambda\|\Psi$ and $\Gamma'\|\Delta'\|\Lambda'\|\Psi$, satisfying*

$$\Gamma\|\Delta\|\Lambda\|\Psi \overset{*}{\Rightarrow}_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\Psi',$$

if there is a solution θ, satisfying $\theta \vDash \Gamma'\|\Lambda'$, then $\theta \vDash \Gamma\|\Lambda$.

Proof. We can prove it by induction from Lemma 26. □

From above theorem we have the following corollary:

Corollary 28 *(Soundness). Let Γ be an AGH-unification problem. Suppose after applying the inferences rules from $\mathfrak{I}_{AGH}$ to $\Gamma\|\emptyset\|\emptyset\|\Psi_\Gamma^{ini}$ exhaustively, we get $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, i.e.*

$$\Gamma\|\emptyset\|\emptyset\|\Psi_\Gamma^{ini} \overset{*}{\Rightarrow}_{\mathfrak{I}_{AGH}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$$

where for each i, no rules are applicable to $\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i$. Let $\Sigma = \{\lambda_{\Lambda_i} \mid \Gamma_i = \emptyset\}$. Then any member of Σ_σ is an AGH-unifier of Γ.

10. Completeness

In this section, we show that the inference rules never lose any solutions.

Lemma 29. *Let $\Gamma\|\Delta\|\Lambda\|\Psi$ be a set quadruple. If there exists another set quadruple $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$, such that $\Gamma\|\Delta\|\Lambda\|\Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\Psi'$ via a deterministic rule except Factorization and Annulization, then $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ is a directed conservative extension of $\Gamma\|\Delta\|\Lambda\|\Psi$.*

Proof. There are three cases: Trivial, Simplifiers and Variable Substitution.

Trivial: This case is trivial.

Simplifier: From the definition of Simplifier, the statement is true.

Variable Substitution: Similar to the proof in Lemma 26, we know $\Gamma'\sigma \cup \Lambda\sigma$ is a conservative extension of

$$\Gamma \cup \{c_1x_1 + \cdots + c_nx_x + c'_1t_1 + \cdots + c'_mt_m =^? 0\} \cup \Lambda.$$

All we need to prove is if there is a solution $\theta \vDash \Gamma\|\Delta\|\Lambda\|\Psi$, then the extension θ' of θ satisfies Δ and Ψ.

In Variable Substitution, we have proved that after applying **UnconstrainedReduce**, $\Gamma' \cup \Lambda'$ is a conservative extension of $\Gamma \cup \Lambda$. Since any solution θ of $\Gamma \cup \Lambda$ satisfies $(s\theta + t\theta)\downarrow \neq 0$, we claim each solution δ for $\Gamma' \cup \Lambda'$ satisfies $(s\delta + t\delta)\downarrow \neq 0$. Otherwise, since $\Gamma' \cup \Lambda'$ is a conservative extension of $\Gamma \cup \Lambda$, for any δ, there exists θ, which is a solution of $\Gamma \cup \Lambda$ and we can find a substitution σ, such that $\delta\sigma =_{\mathbf{AGH}} |_{Vars(\Gamma)}\theta$. Since $(s\delta + t\delta)\downarrow = 0$, we can get $(s\delta\sigma + t\delta\sigma)\downarrow = 0$, which is $(s\theta + t\theta)\downarrow = 0$. This is the contradiction. Hence Δ does not change after applying Variable Substitution. And the possible change of Ψ is from some equation being solved, and **Purify** and **Singlize** are applied.

If some equation is solved and an h-term is removed from Γ, then the corresponding pair will be removed from Ψ too. So it is trivially true.

Suppose after Variable Substitution, the solution is extended to θ', For **Purify**, suppose the old equation is $\mathbf{S} + ch(t) + dh(s) =^? 0$, and $(N_1, t), (N_2, s)$. After rewriting and **Purify**, $ch(t_1) + dh(t_2)$ is replaced by $h(x')$ and $(N_1, t), (N_2, s)$ is replaced by $(|cN_1 + dN_2|, x')$. From $(N_1, t), (N_2, s)$, we know $t\theta'\downarrow = \sum_{i=1}^k c_it_i$ and $s\theta'\downarrow = \sum_{i=1}^l d_is_i$, for all i, $c_i < N_1$, $d_i < N_1$ and for any i, j, $t_i \neq t_j$. We extend θ' to $\theta'' = \theta' \cup \{x' \longmapsto c(-t) + d(-s)\}$, then $x'\theta'' = c(-t)\theta'' + d(-s)\theta'' = c\sum_{i=1}^k c_it_i + d\sum_{i=1}^l d_is_i$. So for every i, we have $|cc_i| < |cN_1|$ and $|dd_i| < |dN_2|$. So for $x'\theta'' \sum_{i=1}^l d'r_i$, the largest absolute value of the coefficient is at most $cN_1 + dN_2$, which means θ'' satisfies Ψ.

And because if $\theta' \vDash \mathbf{S} + ch(t) + dh(s) =^? 0$, then $\theta'' \vDash \{\mathbf{S} + h(x') =^? 0, x' + (-ct) + (-ds) =^? 0\}$, θ'' is also a solution of $\Gamma' \cup \Lambda'$.

In **Singlize**, because we only remove the extra identical h-terms and the pair with a larger integer. (N, s) means for the solution θ, we have $s\theta = \sum_{i=i}^k c_is_i$, $c_i < N$ for every c_i and $t_i \neq t_j$ for any $i..j$. Therefore for two pairs (N_1, s) and (N_2, s), where $N_1 \leq N_2$, we have $c_i < N_1 \leq N_2$. So after removing (N_2, s), θ still satisfies (N_1, s).

So this statement is true. □

Lemma 30. *Let $\Gamma\|\Delta\|\Lambda\|\Psi$ be a set quadruple. If there exists another set quadruple $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$, such that $\Gamma\|\Delta\|\Lambda\|\Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\|\Psi'$ via Factorization,then $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$ is a directed conservative extension of $\Gamma\|\Delta\|\Lambda\|\Psi$.*

Proof. Our inference rules have priorities. Therefore if $\Gamma\|\Delta\|\Lambda\|\Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma'\|\Delta'\|\Lambda'\|\|\Psi'$ via Factorization, then Γ has an equation of the form

$$\mathbf{S} + cy + dx + h(x) =^? 0$$

where $(n, x) \in \Psi$ and $n > c$ (d might be 0). Suppose there is a solution θ for $\Gamma \| \Delta \| \Lambda \| \Psi$, such that $(x\theta) \downarrow = ct + s$ where $t \neq 0$ because of the pair (n, x). After applying Factorization, we get

$$\Gamma \cup \{\mathbf{S} + cy' + dx'' + h(x'') =^? 0\} \| \Delta \| \Lambda\theta \| \Psi \cup \{(c, x'')\}\theta.$$

So it is enough to show we can find a δ, such that

$$\theta\delta \vDash \{cy' + \mathbf{S} + dx'' + h(x'') =^? 0,\ x =^? cx' + x'', y =^? y' + (-dx') + h(-x')\}$$

and $\theta\delta$ satisfies Ψ'.

Let $\delta = \{x'' \longmapsto x + (-cx'), y' \longmapsto y + dx' + h(x')\}$, where x' satisfies: if $(x\theta) \downarrow = \sum_{i=1}^{k}(c_i t_i)$, where every t_i is a monic term and $t_i \neq t_j$ for every i, j, then $(x'\theta\delta) \downarrow = \sum_{i=1}^{k}(\lfloor \frac{c_i}{c} \rfloor t_i)$.

Then

$$\begin{aligned}
(cy' + \mathbf{S} + dx'' + h(x''))\theta\delta &= c(y + dx' + h(x'))\theta\delta + \mathbf{S}\theta\delta + d(x + (-cx'))\theta\delta + h(x + (-cx'))\theta\delta \\
&= cy\theta\delta + cdx'\theta\delta + ch(x')\theta\delta + \mathbf{S}\theta\delta + dx\theta\delta + (-cdx')\theta\delta + h(x)\theta\delta + h(-cx')\theta\delta \\
&= cy\theta\delta + \mathbf{S}\theta\delta + h(x)\theta\delta \\
&= 0
\end{aligned}$$

$$\begin{aligned}
(cx' + x'')\theta\delta &= cx'\theta\delta + x''\theta\delta \\
&= cx'\theta\delta + x\theta\delta + (-cx')\theta\delta \\
&= x\theta\delta \\
&= x\theta
\end{aligned}$$

$$\begin{aligned}
(y' + (-dx') + h(-x'))\theta\delta &= y'\theta\delta + (-dx')\theta\delta + h(-x')\theta\delta \\
&= y\theta\delta + dx'\theta\delta + h(x')\theta\delta + (-dx')\theta\delta + h(-x')\theta\delta \\
&= y\theta\delta \\
&= y\theta
\end{aligned}$$

Suppose $x\theta = \sum_{c_i}^{k} t_i$, where t_i is a monic term and $t_i \neq t_j$ for every i, j, because we have the constraint

$$(x'\theta\delta) \downarrow = \sum_{i=1}^{k}\left(\left\lfloor \frac{c_i}{c} \right\rfloor t_i\right)$$

where $(x\theta) \downarrow = \sum_{i=1}^{k}(c_i t_i)$, then

$$x''\theta\delta = x\theta\delta - cx'\theta\delta = \sum_{i=1}^{k}((c_i \bmod c)t_i)$$

So $\theta\delta$ satisfies Ψ'.

Thus this statement is true. □

Before we give the next lemma, we define a function called *(Lay)* which will count the layers of a term, when the term is represented as a tree.

Definition 31. If t is reduced with respect to $\mathfrak{R}_{AGH}$, then $Lay(t)$ has the following value:

- $Lay(t) = 0$, if t is a constant or variable.
- $Lay(f(c_1t_1, c_2t_2, \cdots, c_nt_n)) = \max\{Lay(t_1), Lay(t_2), \cdots, Lay(t_n)\} + 1$, where f is an uninterpreted symbol.
- $Lay(c_1t_1 + c_2t_2 + \cdots + c_nt_n) = \max\{Lay(t_1), Lay(t_2), \cdots, Lay(t_n)\}$.
- $Lay(h(t)) = 1 + Lay(t)$.

From the definition, we get $Lay(kt) = Lay(t)$, so sometimes we will omit the coefficient.

Lemma 32. *Let $\Gamma \| \Delta \| \Lambda \| \Psi$ be a set quadruple. If there exists another set quadruple $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$, such that $\Gamma \| \Delta \| \Lambda \| \Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma' \| \Delta' \| \Lambda' \| \Psi'$ via Annulization,then $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$ is a directed conservative extension of $\Gamma \| \Delta \| \Lambda \| \Psi$.*

Proof. Since Annulization is applicable, all the equations have the form $c_{y_i} y_i + c_{i1}x_{i1} + c_{i2}x_{i2} + \cdots + c_{im}x_{im} + h(t) =^? 0$, where x_{ij} is not unconstrained, $(c, t) \in \Psi$ and $n \le c$.

Annulization sets all the h-terms to zero. Annulization will not change Δ, and the corresponding pairs will be removed from Ψ if some h-terms are set to zero. So it is enough to show there is no solution θ, such that for some $h(t)$, $(h(t)\theta) \downarrow \ne 0$.

Suppose there is a ground reduced substitution θ and there exists an h-term $h(t)$, such that $(h(t)\theta) \downarrow \ne 0$.

θ is a ground reduced substitution, which is $\{x_1 \longmapsto T_1, x_2 \longmapsto T_2, \cdots x_n \longmapsto T_n, y_1 \longmapsto S_1, y_2 \longmapsto S_2, \cdots, y_l \longmapsto S_l\}$, where each x_i is a constrained variable and y_i is an unconstrained variable in Γ. Without loss of generality, we suppose $Lay(T_1) \ge Lay(T_2) \ge \cdots \ge Lay(T_n)$.

We claim that $Lay(T_1)$ is not zero. If it is, then all $Lay(T_i)$ are zero,which means all the T_i are variables or constants. Then for the equation $cy_i + c_{i1}x_{i1} + c_{i2}x_{i2} + \cdots + c_{im}x_{im} + h(t) =^? 0$, where $(h(t)\theta) \downarrow \ne 0$, we have:

$$\big(cy_i + c_{i1}x_{i1} + c_{i2}x_{i2} + \cdots + c_{im}x_{im} + h(t)\big)\theta = cy_i\theta + ca_1 + ca_2 + \cdots ca_m + h(t)\theta$$

where a_i are constants or variables. The only possibility to cancel $h(t)\theta$ is $cy_i\theta$. Because $h(t)$ is a pure term, t's top symbol is some uninterpreted function symbol or t is a variable.

If $t = f(s_1, s_2, \cdots, s_l)$, then $t\theta = f(s_1, s_2, \cdots, s_l)\theta$. Because y_i can cancel $t\theta$, $y_i = h(kf(s_1, s_2, \cdots, s_l))\theta + S$. Because we can only apply Annulization, c has the largest absolute value among all the coefficients, or Variable Substitution will apply. So $|c| > 1$. Therefore $cy_i\theta + h(t)\theta = ch(kf(s_1, s_2, \cdots, s_l))\theta + h(f(s_1, s_2, \cdots, s_l))\theta + S = h((ck+1)f(s_1, s_2, \cdots, s_l))\theta + S$, which is impossible to be zero.

If t is a variable x_j, suppose $T_j = \sum_{k=1}^{n} c_k t_k$, where t_k is a monic term $|c_k| < |c|$ and $t_{k_1} \ne t_{k_2}$ for every k_1, k_2, because $(c, x_j) \in \Psi$. So $y_i\theta = \sum_{k_1}^{n} c'_k t_k + S$. Also $cy_i\theta + h(x_j)\theta = \sum_{k=1}^{n}(cc'_k + c_k)t_k + S$. Because $|c_k < c|$, $\sum_{k=1}^{n}(cc'_k + c_k)t_k$ cannot to be zero.

So $Lay(T_1)$ is not zero. Since x_1 occurs in t in the equation $cy_i + c_{i1}x_{i1} + c_{i2}x_{i2} + \cdots + c_{im}x_{im} + h(t) =^? 0$, where x_{ij}'s are not unconstrained variables, if we want to cancel $h(t)\theta$, we need another variable $x_{ij}\theta$ or $y_i\theta$ to cancel it because there are no other h-terms in it. If $x_{ij}\theta$ can cancel $h(t)\theta$ and $x_{ij} = x_l$, then $x_l\theta = kh(t)\theta + T$. Suppose $Lay(T_l) = Lay(kh(t)\theta + T) \ge Lay(kh(t)\theta) \ge Lay(h(t)\theta) > Lay(x_1\theta) = Lay(T_1)$, which is a contradiction. If $y_i\theta$ can cancel $h(t)\theta$, then the case is same as above.

So in this case there is no solution θ, such that for some $h(t)$, $(h(t)\theta \downarrow) \ne 0$. Therefore the statement is true. □

Lemma 33. *Let $\Gamma \| \Delta \| \Lambda \| \Psi$ be a set quadruple. If there exists another set quadruple $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$, such that $\Gamma \| \Delta \| \Lambda \| \Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma' \| \Delta' \| \Lambda' \| \Psi'$ via N-Decomposition, then $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$ is a directed conservative extension of $\Gamma \| \Delta \| \Lambda \| \Psi$.*

Proof. Because N-Decomposition is a nondeterministic rule, there are two possible cases:

Case 1: Γ has an equation of the form:

$$\mathbf{S} + cf(s_1, s_2, \cdots, s_m) + \big(-df(t_1, t_2, \cdots, t_m)\big) =^? 0$$

which has a solution θ, such that $f(s_1, s_2, \cdots, s_m)\theta = f(t_1, t_2, \cdots, t_m)\theta$.

Since θ is a solution, so $\theta \vDash \Gamma \| \Delta \| \Lambda \| \Psi$. Let σ be the most general unifier of $s_1 =^? t_1, \cdots, s_m =^? t_m$. Since $f(s_1, s_2, \cdots, s_m)\theta = f(t_1, t_2, \cdots, t_m)\theta$, θ is a solution of $s_1 =^? t_1, \cdots, s_m =^? t_m$. There exists a substitution δ, such that $\sigma\delta = \theta$. So $\theta \vDash [\sigma]$ from $\theta \vDash [\sigma\delta]$. Hence for every equation Q, we have that if $\theta \vDash Q$ then $\theta \vDash Q\sigma$. Because $\sigma \vDash \mathbf{S} + cf(s_1, s_2, \cdots, s_m) + (-df(t_1, t_2, \cdots, t_m)) =^? 0$ and $\sigma \vDash f(s_1, s_2, \cdots, s_m) = f(t_1, t_2, \cdots, t_m)$, we can get $\sigma \vDash \mathbf{S} + (c-d)f(s_1, s_2, \cdots, s_m) =^? 0$. So $\sigma \vDash (\mathbf{S} + (c-d)f(s_1, s_2, \cdots, s_m) =^? 0)\sigma$. Hence σ satisfies every equation in $\Gamma\sigma \cup \{Q\}\sigma$ and $\Lambda\sigma \cup [\sigma]$.

Because Δ does not change, θ still satisfies Δ'. Ψ might be changed for **Singlize**, but we already proved θ still satisfies Ψ' after **Singlize** applies in the proof of Lemma 29. Thus $\theta \vDash (\Gamma\sigma \cup \{Q\}\sigma) \| \Delta\sigma \| (\Lambda\sigma \cup [\sigma]) \| \Psi\sigma$.

Case 2: Γ has an equation of the form

$$\mathbf{S} + cf(s_1, s_2, \cdots, s_m) + \big(-df(t_1, t_2, \cdots, t_m)\big) =^? 0$$

and a solution θ, such that $f(s_1, s_2, \cdots, s_m)\theta \ne f(t_1, t_2, \cdots, t_m)\theta$, there is no deterministic rule to be applied and $f(s_1, s_2, \cdots, s_m) + (-f(t_1, t_2, \cdots, t_m)) \ne^? 0 \notin \Delta$.

We apply N-Decomposition (the second choice) and add $f(s_1, s_2, \cdots, s_m) + (-f(t_1, t_2, \cdots, t_m)) \ne^? 0$ into the disequation set. It is trivially true that

$$\theta \vDash \Gamma \| \Delta \cup \big\{f(s_1, s_2, \cdots, s_m) + \big(-f(t_1, t_2, \cdots, t_m)\big) \ne^? 0\big\} \| \Lambda \| \Psi.$$

From these two cases, if there is some solution θ and N-Decomposition is applied, then there exists an extension θ' of θ such that θ' satisfies one of the choices of the conclusion after N-Decomposition is applied. □

Lemma 34. *Let $\Gamma \| \Delta \| \Lambda \| \Psi$ be a set quadruple. If there is not another set quadruple $\Gamma' \| \Delta' \| \Lambda' \| \Psi'$, such that $\Gamma \| \Delta \| \Lambda \| \Psi \Rightarrow_{\mathfrak{I}_{AGH}} \Gamma' \| \Delta' \| \Lambda' \| \Psi'$, and Γ is not empty, then $\Gamma \| \Delta \| \Lambda \| \Psi$ has no solution.*

Proof. Because there are no rules applicable including Annulization, there is some uninterpreted function symbol occurring as top function symbol, i.e. some equation has the form:

$$c_y y + c_1 x_1 + \cdots + c_n x_n + \mathbf{S} + cf(s_1, s_2, \cdots, s_m) =^? 0$$

where y is an unconstrained variable in Γ, c_y might be zero, each x_i is a constrained variable in Γ, and $\mathbf{S}$ contains no pure variables.

Assume there are no rules to apply to $\Gamma \| \Delta \| \Lambda \| \Psi$ and θ is a ground reduced substitution, which is $\{x_1 \longmapsto T_1, x_2 \longmapsto T_2, \cdots x_n \longmapsto T_n, y_1 \longmapsto S_1, y_2 \longmapsto S_2, \cdots, y_l \longmapsto S_l\}$, where each x_i is a constrained variable and each y_i is an unconstrained variable in Γ. Without loss of generality, we suppose $Lay(T_1) \geq Lay(T_2) \geq \cdots \geq Lay(T_n)$.

Here, we claim that $Lay(T_1)$ is not zero. If it is, then all $Lay(T_i)$s are zero. Then the equation

$$c_y y + c_1 x_1 + \cdots + c_n x_n + \mathbf{S} + cf(s_1, s_2, \cdots, s_m) =^? 0$$

becomes

$$c_y y\theta + c_1 a_1 + \cdots + c_n a_n + \mathbf{S}\theta + cf(s_1, s_2, \cdots, s_m)\theta = 0$$

where a_i are constants or variables. Because this equation is true, we need the inverse of $f(s_1, s_2, \cdots, s_m)\theta$ to cancel $cf(s_1, s_2, \cdots, s_m)\theta$. If there is some f-term, e.g. $df(t_1, t_2, \cdots, t_m)\theta$ in $\mathbf{S}\theta$ which can cancel $cf(s_1, s_2, \cdots, s_m)\theta$, then because here we have no rules to apply, $f(t_1, t_2, \cdots t_m) - f(s_1, s_2, \cdots, s_m) \neq 0 \in \Delta$, which means $f(t_1, t_2, \cdots t_m)\theta \neq f(s_1, s_2, \cdots, s_m)\theta$, so the only possibility to cancel an f-term is $c_y y\theta$. But $|c_y| > |c|$, or else we can apply Variable Substitution, so if $y\theta = kf(s_1, s_2, \cdots, s_m)\theta + T$, then $c_y y\theta + cf(s_1, s_2, \cdots, s_m)\theta = (c_y k + c) f(s_1, s_2, \cdots, s_m)\theta + T$, which is impossible to be zero.

So we can say $Lay(T_1)$ is not zero.

Because no rules in $\mathfrak{I}_{AGH}$ can be applied, from Variable Substitution's conditions, all unconstrained variables must be unconstrained reduced variables. And since x_1 is not an unconstrained variable, x_1 must occur under some uninterpreted function symbol or h-term. So we have two cases:

Case A: x_1 occurs under some uninterpreted function symbol. Suppose this equation is $\mathbf{S} + ct =^? 0$ where $x_1 \in t$ and $Top(t)$ is an uninterpreted function symbol or t is an h-term of the form $h(f(t_1, t_2, \cdots, t_n))$. Then if we want to cancel $x_1\theta$, we need another variable $x\theta$ to cancel it because there is no other t' in $\mathbf{S}$ with f on the top to cancel it or else N-Decomposition could be applied.

Case A.1: If $x = x_i$ for some x_i, then $x_i\theta = kt\theta + \mathbf{T}$, where T is a term. Suppose $\mathbf{S} = c_x x_i + S'$, then $c_x x_i\theta = c_x T_i = c_x(kt\theta + \mathbf{T})$. Then $Lay(T_i) = Lay(kt\theta + \mathbf{T}) \geq Lay(kt\theta) \geq Lay(t\theta) > Lay(x_1\theta) = Lay(T_1)$, which is a contradiction.

Case A.2: If x is an unconstrained variable, then $y_j\theta = kt\theta + \mathbf{T}'$ for some y_j, where T is a term. So $c_y y_j\theta + ct\theta = c_y(kt\theta + \mathbf{T}') + ct = (c_y k + c)t\theta + \mathbf{T}'$. Because all the unconstrained variables are reduced, and $|c_y| > |c|$, then $c_y k + c \neq 0$, which means there still some $t\theta$ left in the equation, we still need another variable to cancel it which can only be another constrained variable. This goes back to Case A.1.

Case B: x_1 occurs only in an h-term. If it is under some uninterpreted function symbol in this h-term, we know there is no solution from the analysis of Case A. So we can suppose this equation is $\mathbf{S} + h(x_1) =^? 0$. Because $Lay(T_1) \neq 0$, we can suppose $T_1 = ct + T_2'$, where $Lay(T_1) = Lay(ct) \geq Lay(T'2)$ and t is a monic term. Because only one h-term is in this equation and $h(s) + h(t) = h(s+t)$, we need another variable x to have the contribution to cancel $h(ct)$. We claim that no variable can cancel $h(ct)$. In the next two cases, we will first prove that no constrained variables can cancel $h(t)$, and we will show that no unconstrained variables can completely cancel $h(ct)$.

Case B.1: If some constrained variable x_i, whose coefficient is d, can cancel $h(t)$, then $x_i\theta = h(kt) + R$. So $Lay(T_i) = Lay(x_i\theta) = Lay(dx_i\theta) = Lay(dh(kt) + dR) \geq Lay(h(T_1)) > Lay(T_1)$ which is a contradiction.

Case B.2: Suppose the unconstrained variable y_j, which has the coefficient c_y, can cancel $h(ct)$. Then $y_j\theta = h(kt) + R$. Thus $c_y y_j\theta + h(ct) = c_y kh(t) + c_y R + h(ct) = h((c_y k + c)t) + c_y R$. So we need $(c_y k + c)t = 0$ or if $c_y k + c \neq 0$, there are still some $h(t)$s left, which we need another variable to cancel $h(t)\theta$, but we already proved in Case B.1 that this is impossible. So $(c_y k + c)t = 0$. But $Lay(t) > 0$, so $c_y k + c = 0$, which means $|c| > |c_y|$. We already know there is no rule applicable, and x_1 occurs only under an h-term, so $(n, x_1) \in \Psi$, where $n < c_y$, which means $x\theta = \sum_{k=1}^{m} d_k s_k$ and $d_k < n$. So in this case if $c > c_y$, then t needs to be zero, which is a contradiction of $Lay(t) > 0$.

In summary, if there is no rule we can apply for some quadruple, there is no solution for this quadruple. □

Then by combining Lemma 29, Lemma 30, Lemma 32, Lemma 33 and Lemma 34 we get:

Lemma 35. *Let $\Gamma \| \Delta \| \Lambda \| \Psi$ be a set quadruple and Γ is not empty. If there exists a set of set quadruples $\bigvee_i (\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$, such that $\Gamma \| \Delta \| \Lambda \| \Psi \Rightarrow_{\mathfrak{I}_{AGH}} \bigvee_i (\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$, then $\bigvee_i (\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$ is a directed conservative extension of $\Gamma \| \Delta \| \Lambda \| \Psi$. If there is no such a set of set quadruples, then $\Gamma \| \Delta \| \Lambda \| \Psi$ has no solution.*

Then by induction via Lemma 35, we get:

Theorem 36. *Let $\Gamma \| \Delta \| \Lambda \| \Psi$ be a set quadruple and Γ is not empty. If there exists a set of set quadruples $\bigvee_i (\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$, such that $\Gamma \| \Delta \| \Lambda \| \Psi \overset{+}{\Rightarrow}_{\mathfrak{I}_{AGH}} \bigvee_i (\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$, then $\bigvee_i (\Gamma_i' \| \Delta_i' \| \Lambda_i' \| \Psi_i')$ is a directed conservative extension of $\Gamma \| \Delta \| \Lambda \| \Psi$. If there is no such set quadruple, then $\Gamma \| \Delta \| \Lambda \| \Psi$ has no solution.*

From above theorem we have the following corollary:

Corollary 37 *(Completeness). Let Γ be an AGH-unification problem. Suppose after applying the inferences rules from $\mathfrak{I}_{AGH}$ to $\Gamma \| \emptyset \| \emptyset \| \Psi_\Gamma^{ini}$ exhaustively, we get $\bigvee_i (\Gamma_i \| \Delta_i \| \Lambda_i \| \Psi_i)$, i.e.*

$$\Gamma \| \emptyset \| \emptyset \| \Psi_\Gamma^{ini} \overset{*}{\Rightarrow}_{\mathfrak{I}_{AGH}} \bigvee_i (\Gamma_i \| \Delta_i \| \Lambda_i \| \Psi_i)$$

where for each i, no rules are applicable to $\Gamma_i \| \Delta_i \| \Lambda_i \| \Psi_i$. Let $\Sigma = \{\lambda_{\Lambda_i} \mid \Gamma_i = \emptyset\}$. Then for any AGH-unifier δ of Γ, there exist a $\sigma \in \Sigma$, such that $\sigma \leq_{AGH} \delta|_{Vars(\Gamma)}$.

11. Auxiliary rules for improving efficiency

For efficiency, we add some other inference rules to our system.

Dis-Trivial

$$\frac{\Gamma \| (\Delta \cup \{0 \neq^? 0\}) \| \Lambda \| \Psi}{\textbf{Fail}}.$$

Clash

$$\frac{(\Gamma \cup \{\mathbf{S} + c_1 f(t_{11}, \cdots, t_{1n}) + \cdots + c_m f(t_{m1}, \cdots, t_{mn}) =^? 0\}) \| \Delta \| \Lambda \| \Psi}{\textbf{Fail}}.$$

if there are no pure variables and f is not a top symbol in $\mathbf{S}$ and $\sum_{i=1}^m c_i$ is not zero.

Example 38. The equation:

$$2f(x) + 3f(y) + \big(-3f(a)\big) =^? 0$$

has no solution.

Clash-Annul

$$\frac{(\Gamma \cup \{\mathbf{S} + h(t) =^? 0\}) \| \Delta \| \Lambda \| \Psi}{(\Gamma \cup \{\mathbf{S} =^? 0\} \cup \{t =^? 0\}) \| \Delta \| \Lambda \| \Psi},$$

if there is no pure variable in $\mathbf{S}$.

Example 39. For the equation:

$$2f(x) + \big(-2f(y)\big) + \big(-2f(0)\big) + h(x) =^? 0$$

we can let $x \longmapsto 0$ before we use N-Decomposition. So we find out quickly that there is no solution.

Occur Check

$$\frac{(\Gamma \cup \{c_1 x_1 + c_2 x_2 + \cdots + c_n x_n + c'_1 t_1 + c'_2 t_2 + \cdots + c'_m t_m =^? 0\}) \| \Delta \| \Lambda \| \Psi}{\textbf{Fail}}.$$

where each t_i is a monic term, and for all x_i, there exists a pure term t_j, such that:

1. $x_i \in Vars(t_j)$ and $Top(t_j) \neq h$.
2. The sum of the coefficients of every term in $|Top(t_j; \{t_1, t_2, \cdots, t_m\})|$ is not zero.
3. x_i occurs in every pure term in $Top(t_j; \{t_1, t_2, \cdots, t_m\})$.

Example 40. The equation $x + 2f(x) =^? 0$ has no solution.

Occur Check-Annul

$$\frac{(\Gamma \cup \{cx + \mathbf{S} + h(t) =^? 0\})\|\Delta\|\Lambda\|\Psi}{(\Gamma \cup \{t =^? 0\} \cup \{\mathbf{S} =^? 0\})\|\Delta\|\Lambda\|\Psi},$$

if there is no pure variable in **S** and $x \in Vars(t)$.

Example 41. We have equation set $\{x + h(x) =^? 0,\ f(x) + (-f(0)) + f(y) + (-f(z)) =^? 0\}$. We use Occur Check-Annul to get $x \longmapsto 0$, and then get the solution $\{x \longmapsto 0,\ y \longmapsto z\}$. This rule can avoid applying N-Decomposition too early.

Decomposition-II

If all the pure variables x_i in **S** occur in some s_j (or t_j), and t_j and s_j are monic terms and no top symbol of a term of **S** is f, then:

$$\frac{(\Gamma \cup \{\mathbf{S} + cf(s_1, s_2, \cdots s_m) + (-c)f(t_1, t_2, \cdots t_m) =^? 0\})\|\Delta\|\Lambda}{(\Gamma_1\sigma)\|(\Delta\sigma)\|(\Lambda\sigma \cup \{[\sigma]\})}$$

where $\Gamma_1 = (\Gamma \cup \{\mathbf{S} =^? 0\})$, where $\sigma = mgu(s_1 =^? t_1, s_2 =^? t_2, \cdots, s_m =^? t_m)$.

Decomposition-III

If all the pure variables x_i in **S** occur in some s_j (or r_j, or t_j), and t_j, s_j and r_j are monic terms and no top symbol of a term of **S** is f, then:

$$\frac{(\Gamma \cup \{\mathbf{S} + c_1 f(s_1, \cdots, s_m) + c_2 f(t_1, \cdots, t_m) + c_3 f(r_1, \cdots, r_m) =^? 0\})\|\Delta\|\Lambda}{(\Gamma_1\sigma)\|(\Delta\sigma)\|(\Lambda\sigma \cup \{[\sigma]\})}$$

where $\Gamma_1 = (\Gamma \cup \{\mathbf{S} =^? 0\})$, $\sigma = mgu(s_1 =^? t_1, \cdots, s_m =^? t_m, s_1 =^? r_1, \cdots, s_m =^? r_m)$ and $c_1 + c_2 + c_3 = 0$.

Before we prove all of our auxiliary rules are simplifiers, we need the following lemma.

Lemma 42. *Suppose* **S** *has the reduced form $t_1 + \cdots + t_n$, where t_i is not a variable, and no f occurs as top symbol in t_i. Then*

$$\mathbf{S} + c_1 f(s_{11}, s_{12}, \cdots, s_{1n}) + c_2 f(s_{21}, s_{22}, \cdots s_{2n}) + \cdots + c_m f(s_{m1}, s_{m2}, \cdots s_{mn}) =^? 0$$

has no solution if $\sum_{i=1}^{m} c_i \neq 0$.

We omit the proof here since it can follow the property of the **AGH**.

Theorem 43. *All the auxiliary rules are (P, E, μ)-Simplifiers, where, P is in* PURE SUM *form, E is Abelian Group with Homomorphism and μ is $\mathbb{M}_{AGH}(\Gamma, \Delta, \Lambda, \Psi)$.*

Proof. In Lemma 29, we proved that the purification procedure will preserve Ψ. So for proving the auxiliary rules are E-equation preserving, we only need to prove two directions: a solution of the premise is a solution of the conclusion and a solution of the conclusion is also a solution of the premise.

For **Dis-Trivial** both directions are easily proved. From Lemma 42 we know that **Clash** is E-equation preserving.

For **Clash-Annul**, if $h(t) \neq 0$, then there is no rewriting rule to cancel $h(t)$ because there are no other h-terms and variables. So if there is a solution σ for the premise, $h(t)$ has to be zero and σ is a solution of the conclusion. The other direction is trivial.

For **Occur Check**, we can suppose there is a ground reduced solution

$$\theta = \{x_1 \longmapsto s_1, x_2 \longmapsto s_2, \cdots x_n \longmapsto s_n, y_1 \longmapsto s'_1, \cdots, y_k \longmapsto s'_k\}$$

Without loss of generality, we suppose $Lay(s_1) \geq Lay(s_2) \geq \cdots \geq Lay(s_n)$, $x_1 \in t_1$, and $Top(t_1) = f$, where f is an uninterpreted function symbol, $Top(t_1; \{t_1, t_2, \cdots, t_m\}) = \{t_1, t_2, \cdots, t_l\}$ where $\sum_{i=1}^{l} c'_i \neq 0$ from the condition of Occur Check.

Table 1
Experiment result.

Problems	Real Time	Steps	# Sol.	Min?
$f(x)+f(y)=^? f(a)+f(b)$	60 ms	6	2	YES
$f(x)+f(y)+f(z)=^? f(a)+f(b)+f(c)$	1391 ms	12	6	YES
$f(x)+f(y)+f(z)+f(w)=^? f(a)+f(b)+f(c)+f(d)$	43 s	20	24	YES
$x=^? f(x+y)$	9 ms	3	1	YES
$x=^? f(x+y-f(y))$	12 ms	3	1	YES
$x=^? h(x+y-h(y))$	17 ms	5	1	YES
$2y+w+h(z)=^? 3(v)+z+h(w)$	19 ms	4	1	YES
$2y+w+h(z)=^? 0$ $3v+z+h(w)=^? 0$	33 ms	3	1	YES
$2y+w+h(z)=^? 0$ $3v+z+2x+h(w)=^? 0$	22 ms	2	1	YES
$3x+2y+h(x+y)+3f(h(x+y+z),g(a))=^? 0$	47 ms	7	1	YES
$3f(h(x+y+z),g(a))+x=^? 0$	24 ms	5	1	YES
$2y+3f(w,a)+7f(z,a)+8x=^? 0$ $2y+4x+5f(w,a)+5f(z,a)=^? 0$	43 ms	4	1	YES
$f(y)+3x+3f(z)+4z=^? 0$	13 ms	2	1	YES
$x+f(y)-f(x_1)=^? 0$ $y+f(z)-f(x_2)=^? 0$ $z+f(x)-f(x_3)=^? 0$	81 ms	6	3	YES

Because $\sum_{i=1}^{l} c'_i \neq 0$, from the proof of Lemma 42, we know at least we need to find another term with f as top symbol to cancel some t_i which cannot be canceled by other elements in $\{t_1, t_2, \cdots, t_l\}$. This must be from some variable. Suppose this variable is x_i. Because every equation before applying the substitution is pure and θ is ground and reduced, then no subterm under an uninterpreted function symbol can be canceled after applying the substitution, which means $x_i\theta = s_i = kt_i[x_1]\theta + S\theta = kt_i[s_1]\theta + S\theta$, where S is a term. This means $Lay(s_1) < Lay(s_i)$ which is a contradiction. So in this case, there is no solution. This means **Occur Check** is E-equation preserving.

For **Occur Check-Annul**, if there is a solution θ, then $cx\theta + \mathbf{S}\theta + h(t\theta) = 0$. Because $cx + \mathbf{S} + h(t\theta) =^? 0$ is a PURE SUM and no pure variables occur in $\mathbf{S}$, there are two possible ways to cancel $h(t\theta)$ from our rewriting rules: one is from $cx\theta = h(t\theta) + T \neq 0$ and the other one is $t\theta = 0$. For the first case, because x occurs in t, then $Lay(x\theta) = Lay(h(t\theta)) > Lay(x\theta)$, which is a contradiction. So the only possible case is the second case, which means θ is also a solution of the conclusion. If the h-term has the form of $h(t[x])$ where $Top(t)$ is an uninterpreted function symbol, $t =^? 0$ will fail from Lemma 42. If the h-term has the form of $h(x)$, then $x = 0$. The other direction is trivial.

For **Decomposition II**, if $cf(s_1, s_2, \cdots, s_m) - cf(t_1, t_2, \cdots, t_m) \neq 0$, then there is no rewriting rule to cancel $f(s_1, s_2, \cdots s_m)$ and $-f(t_1, t_2, \cdots t_m)$ because there are no other terms with top symbol f. So if there is a solution θ for the premise, then the only possibility to cancel them is from the pure variables. But we already know every pure variable occurs in some s_i or t_j. Suppose the pure variable set is $\{x_1, x_2, \cdots, x_k\}$ and $Lay(x_1\theta) \geq Lay(x_2\theta) \geq \cdots \geq Lay(x_k)$. If $Lay(x_1\theta) = 0$, then $Lay(x_i) = 0$ for all i. There will be no solution because $f(s_1, s_2, \cdots, s_m)$ cannot be canceled. Assume $x_1 \in Vars(s_i)$, and some $x_i\theta = kf(s_1, s_2, \cdots, s_m)$, then we should have $Lay(x_i\theta) \geq Lay(f(s_1, s_2, \cdots, s_m)\theta) > Lay(x_1\theta)$, which is a contradiction. If $x_1 \in Vars(t_j)$, the discussion is the same. So if there is some solution, we need to have $f(s_1, s_2, \cdots, s_m)\theta = f(t_1, t_2, \cdots, t_m)\theta$. The other direction is trivial.

Decomposition III is the same as **Decomposition II**.

We will apply the purification procedure whenever Γ is not in PURE SUM form, so all the rules preserve PURE SUM form.

Next we prove all the rules are μ-reduced. **Trivial**, **Clash** and **Occurs Check** are trivial because they go to **fail**. For **Clash-Annul** and **Occur Check-Annul**, an h-term is solved, so $H(\Gamma)$ decreases. For **Decomposition II** and **III**, either some variables are solved or two fs are removed, and $\mathbb{M}_{AGH}(\Gamma, \Delta, \Lambda, \Psi)$ decreases. So the conclusion is μ-decreasing.

Hence all of the above rules are simplifiers. $\square$

12. Implementation

Due to the high nondeterminism of the combination algorithm [11], as far as we know, there is no public implementation for the general AG-unification algorithm yet.

Thanks to Maude [18], we have implemented our algorithm. The resulting program has solved many AGH-unification problems. The running environment is on Windows 7, Intel Core2 Duo 2.26GHz, and 5 GB RAM. Table 1 shows some of our results.

In Table 1, we count one application of a rule (we also count **Singlize** and **Purify** as rules too) as one step. The number of steps is the number of steps of the branch which has the maximum number of steps among all the branches. As we can see in the table, in all of the cases, we got the minimal complete set of unifiers (we cannot guarantee this is true for arbitrary examples). Except for the second and third one, we got the final result in less than 0.1 second. It makes sense that the second and third one need more time to get the final result since their minimal complete set of unifiers contains more solutions. So the result is very satisfactory.

13. Conclusion and future work

We introduced inference rules for general E-unification problems modulo a homomorphism over an Abelian group. We proved these inference rules to be sound, complete and terminating. We also introduced auxiliary rules to avoid applying N-Decomposition, and to make the inference system more efficient. These inference rules also apply to an Abelian group without a homomorphism. In this case, the Variable Substitution rule becomes simpler, because the conditions involving h symbols are trivially true, N-Decomposition remains the same, and Factorization and Annulization never apply. The auxiliary rules are the same, but Clash-Annul and Occur-Check-Annul no longer apply.

This AGH-unification algorithm is an extension of the unification algorithm for XOR with a homomorphisms as given in [15]. It is based on the same framework, but with more sophisticated inference rules. The Purify rules are similar for XOR, and the Singlize rule is not needed in the XOR case. Variable Substitution is much more sophisticated here. N-Decomposition is easily adapted from XOR to Abelian groups. Factorization is new here. Annulization is similar to the XOR case. We believe we have a useful framework that could be extended to other theories.

AGH is an important theory in cryptographic protocols, and that is the focus of our research. The algorithm is simple to implement, and is implemented into Maude already. Now it is being incorporated in the Maude NPA. The inference rules have the benefit that the N-Decomposition rule is not applied often, so the inference system is mostly deterministic. This makes the algorithm more efficient and the complete set of unifiers smaller.

For future work, we will also combine AG and AGH with convergent rewrite theories like cancellation and extend our algorithm by adding arbitrary many homomorphic operators.

References

[1] C. Meadows, Formal verification of cryptographic protocols: a survey, in: J. Pieprzyk, R. Safavi-Naini (Eds.), ASIACRYPT, in: Lect. Notes Comput. Sci., vol. 917, Springer, ISBN 3-540-59339-X, 1994, pp. 135–150.
[2] S. Escobar, C. Meadows, J. Meseguer, Maude-NPA: cryptographic protocol analysis modulo equational properties, in: A. Aldini, G. Barthe, R. Gorrieri (Eds.), FOSAD, in: Lect. Notes Comput. Sci., vol. 5705, Springer, ISBN 978-3-642-03828-0, 2007, pp. 1–50.
[3] P.Y.A. Ryan, D. Bismark, J. Heather, S. Schneider, Z. Xia, Prêt à voter: a voter-verifiable voting system, IEEE Trans. Inf. Forensics Secur. 4 (4) (2009) 662–673.
[4] S. Kremer, M. Ryan, B. Smyth, Election verifiability in electronic voting protocols, in: D. Gritzalis, B. Preneel, M. Theoharidou (Eds.), ESORICS, in: Lect. Notes Comput. Sci., vol. 6345, Springer, ISBN 978-3-642-15496-6, 2010, pp. 389–404.
[5] D. Lankford, G. Butler, B. Brady, Abelian group unification algorithms for elementary terms, Contemp. Math. 29 (1984) 193–199.
[6] Q. Guo, P. Narendran, D.A. Wolfram, Unification and matching modulo nilpotence, in: M.A. McRobbie, J.K. Slaney (Eds.), Proceedings of the 13th International Conference on Automated Deduction on Automated Deduction, New Brunswick, NJ, USA, CADE-13, July 30–August 3, 1996, in: Lect. Notes Comput. Sci., vol. 1104, Springer, Berlin, ISBN 3-540-61511-3, 1996, pp. 261–274 .
[7] K.U. Schulz, A criterion for intractability of E-unification with free function symbols and its relevance for combination algorithms, in: H. Comon (Ed.), RTA, in: Lect. Notes Comput. Sci., vol. 1232, Springer, ISBN 3-540-62950-5, 1997, pp. 284–298.
[8] M. Hermann, P.G. Kolaitis, Unification algorithms cannot be combined in polynomial time, in: M.A. McRobbie, J.K. Slaney (Eds.), Proceedings of the 13th International Conference on Automated Deduction on Automated Deduction, New Brunswick, NJ, USA, CADE-13, July 30–August 3, 1996, in: Lect. Notes Comput. Sci., vol. 1104, Springer, Berlin, ISBN 3-540-61511-3, 1996, pp. 246–260 .
[9] F. Baader, Unification in commutative theories, J. Symb. Comput. 8 (5) (1989) 479–497.
[10] W. Nutt, Unification in monoidal theories, in: M.E. Stickel (Ed.), CADE, in: Lect. Notes Comput. Sci., vol. 449, Springer, ISBN 3-540-52885-7, 1990, pp. 618–632.
[11] M. Schmidt-Schauß, Unification in a combination of arbitrary disjoint equational theories, J. Symb. Comput. 8 (1/2) (1989) 51–99.
[12] A. Boudet, J.-P. Jouannaud, M. Schmidt-Schauß, Unification in Boolean rings and abelian groups, J. Symb. Comput. 8 (5) (1989) 449–477.
[13] N. Dershowitz, D.A. Plaisted, Rewriting, in: J.A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning (in 2 volumes), Elsevier and MIT Press, 2001, pp. 535–610 , ISBN 0-444-50813-9, ISBN 0-262-18223-8.
[14] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C.L. Talcott (Eds.), All About Maude – A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, Lect. Notes Comput. Sci., vol. 4350, Springer, Berlin, ISBN 978-3-540-71940-3, 2007.
[15] Z. Liu, C. Lynch, Efficient general unification for XOR with homomorphism, in: N. Bjørner, V. Sofronie-Stokkermans (Eds.), CADE, in: Lect. Notes Comput. Sci., vol. 6803, Springer, ISBN 978-3-642-22437-9, 2011, pp. 407–421.
[16] F. Baader, W. Snyder, Unification theory, in: J.A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning (in 2 volumes), Elsevier and MIT Press, 2001, pp. 535–610, ISBN 0-444-50813-9, ISBN 0-262-18223-8.
[17] D. Knuth, The Art of Computer Programming, vol. 2, 3rd edition, Addison–Wesley, 2004.
[18] Maude: http://maude.cs.uiuc.edu/.