CLARKSON UNIVERSITY

Dealing Efficiently with Exclusive OR, Abelian Groups and Homomorphism in Cryptographic Protocol Analysis

A Dissertation by

Zhiqiang Liu

Department of Mathematics and Computer Science

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Mathematics

September 2012

The committee below have examined the dissertation entitled“ **Dealing Efficiently with Exclusive OR, Abelian Groups and Homomorphism in Cryptographic Protocol Analysis**” presented by **Zhiqiang Liu**, a candidate for the degree of **Doctor of Philosophy (Mathematics)**, and have certified that it is worthy of acceptance.

September 5th, 2012	Christopher Lynch
Date	Advisor
	James Lynch
	Examining Committee
	Alexis Maciel
	Examining Committee
	Paliath Narendran
	Examining Committee
	Christino Tamon
	Examining Committee

Abstract

The focus of this thesis is to extend the capability and increase the efficiency of formal methods for cryptographic protocol analysis.

When cryptographic protocols are analyzed by formal methods, messages are abstracted by symbolic terms. Normally, terms do not take into account algebraic properties of the cryptographic algorithms. We extend the capability of formal methods tools by developing equational unification algorithms to deal with algebraic properties of cryptographic algorithms, with a goal of making the equational unification algorithms efficient.

Exclusive OR, Abelian groups and homomorphism are common algebraic properties of cryptographic algorithms. So far there are no practically efficient algorithms for solving general E-unification problems for these theories. In the first part of this thesis, we give efficient algorithms for E-unification modulo exclusive OR with homomorphism and Abelian groups with homomorphism, and prove they are terminating, sound and complete. Since these algorithms are based on inference rules, they have been naturally implemented in Maude.

The Maude-NPA is a tool for analyzing cryptographic protocols which adopts abstract state exploration and backward search to find attacks on cryptographic protocols. It has a good mechanism for incorporating different equational theories. It uses strand spaces to represent protocols, and deals well with nonces. It uses many techniques to reduce the search space, but the techniques it uses have conflicts with traditional E-unification algorithms. It uses a new unification algorithm which is called an asymmetric unification algorithm for helping detect infeasible states. In the second part of this thesis, we devise two efficient asymmetric unification algorithms for exclusive OR and Abelian groups and also prove they are terminating, sound and complete. We implemented the algorithm for exclusive OR in Maude and it is being built into the Maude-NPA. The preliminary results are very encouraging.

Acknowledgments

This dissertation would not have been possible without the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of my study at Clarkson University.

First and foremost, I am sincerely and heartily grateful to my advisor, Professor Christopher Lynch, for the support and guidance he showed me throughout not only my dissertation writing, but also my whole graduate study at Clarkson University. He gave me the opportunity to study in the U.S., taught me the knowledge patiently, led me to this interesting research area, helped me writing the academic paper, and became one of my friends.

I would like to show my gratitude to my academic committee members, Professor James Lynch, Professor Alexis Maciel, Professor Paliath Narendran and Professor Christino Tamon. They gave me not only many precious suggestions on this dissertation, but also they helped during my study of computer science. No matter in classes, in conferences or any other place, they were so kind to answer all kinds of questions and helped me enter a higher level of computer science. I would especially like to thank Tino, who not only taught me much knowledge, but also gave me much help in my Clarkson life with his wife Siew Hwee Lee.

I would like to give my thanks to the faculty in the department of Computer Science and Mathematics, such as Professor Michael Felland, Professor Kathleen Fowler, Professor Aaron Luttman, Professor Jeanna Matthews, Professor Kevin Dempsey etc. They taught me knowledge and taught me how to teach and talk.

I would like to thank Hai Lin, Eric McGregor and Yuefeng Tang for their discussions about research with me. They all were Ph.D. students of Chris. I thank them for letting me know more about this research area.

In addition, I would like to thank my family, my wife Bin Xiang, my parents and my son Andrew Liu. They supported me emotionally and made my writing of my thesis peaceful.

I would like to thank Clarkson University, especially the department of Mathematics and Computer Science, for offering me a good place to study.

Last but not least, I would like to thank my friends at Clarkson University, Ahmad Almomani, Jay Appleton, Ye Chen, Claudette Foisy, Michael Fowler, Wenjin Hu, Na Li, Chenyu Lin, Qin Lou, Lu Lu, Tian Ma, Vidoje Mihajlovik, Sumona Modal, Xunyang Shen, Jie Sun, Yilin Tian, Hao Wu, Tao Yang, Guangming Yao, Long Zhang, Jiongxuan Zheng etc., in addition to except their support, they make my life in Potsdam more interesting and colorful.

Table of Contents

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 Cryptographic Protocol Analysis

A protocol is a set of rules which are used by computers to communicate with each other across a network. A cryptographic protocol (security protocol or encryption protocol) is a protocol to provide security [44, 58]. Normally, the properties that a cryptographic protocol needs to ensure are secrecy and authentication. For example, since 1891, elections in the United States are mostly held by *secret ballot*, in which a voter's choice is supposed to be kept secret, i.e., confidential - this is called the *secrecy property.* The server needs to guarantee that the votes are from legal voters - this is called *authentication.* The secrecy property means that the encrypted message can only be viewed by the principal who is supposed to see it, and the authentication property means that the principal knows exactly who he/she is communicating with.

The security of a protocol strongly relies on the cryptographic primitives, like encryption (symmetric and asymmetric), in which any party cannot know an encrypted message unless he/she knows the key. We can find many kinds of algorithms which are used in cryptographic protocols in [30, 58].

However, the security of a cryptographic protocol depends on not only the primitives but also the way the protocols are written, i.e., the way of communication. After several decades of protocol development, researchers found that even for short protocols, even if the encryption method(cryptographic primitive) is unbreakable, potential flaws still may exist in the protocols. A famous example is the Needham-Schroeder Public key protocol [46], which was initially published in 1978, but G. Lowe first found a *man-in-the-middle* attack in 1995, 17 years later [37](indeed, during these years, there was a replay attack identified by Denning and Sacco [18], but fixed by Needham and Schroeder using timestamps [47]). So proving a cryptographic protocol secure is not an easy mathematical problem.

In all cryptographic systems, keys are needed for encryption and decryption. We call the key an *encryption key* if it is used for encrypting a message and a *decryption key* if it is used for decrypting a message. If a message is encrypted, it is supposed to be computationally hard to get the message without knowing the key.

If we assume that an attacker can see, encrypt, remove and insert any messages in the network, we call this kind of attack an *active attack*, and if an attacker can only see the messages and do some calculation, this is called a *passive attack*. Obviously, an active attack can be more harmful, and this is a more realistic assumption for protocols. Lowe's attack of the Needham-Schroeder protocol is a classical example [37] of an active attack.

In analysis of cryptographic protocols, a complete run of a protocol is called a *session.* Many cryptographic protocol analysis tools assume that a protocol may run only a bounded number of times. However, some of them deal with an unbounded number of sessions, because sometimes intruders may find some vulnerable points of the protocol through a very large number of runs of the protocol.

1.1.2 Algebraic Properties in Cryptographic Protocols

Since Dolev and Yao initiated the use of formal methods [21] for analyzing cryptographic protocols in the 1980's, formal methods have been used in the analysis of the security of protocols. Verification tools based on formal methods have been used to automate the proof of the existence of attacks, and to discover new attacks in published cryptographic protocols [37, 45, 53].

In Dolev and Yao's model [21], there are two basic assumptions: The first one is that the malicious intruder can listen, encrypt, remove and insert messages in the network and the second one is that the cryptographic primitives are perfect which means only the people who knows the decryption key can decrypt the message. The second one is called the *perfect encryption assumption.* Modern approaches of analysis of security protocols based on formal methods are more or less from the idea of Dolev and Yao's model.

At the beginning of formal analysis of cryptographic protocols, cryptographic primitives were considered as black boxes. The attacker can only compute using these abstract functions, which means the properties of these function are not taken into account. We call this method the free algebra method [41]. 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 for attacks, a message pattern representing an expected 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, where the *syntactic unification algorithm* is an algorithm to find a substitution of variables to make two terms identical.

However, considering only abstract algebraic functions is not always sufficient in reality. The attackers can still get some information out of an encrypted message without knowing the key since the cryptographic primitives have some algebraic properties.

We give here an example of an attack based on the exclusive OR properties. This example is from [12]. This protocol is to authenticate the identities between two principals. For convenience, we use A and B to denote the two principals, and I to denote the intruder. We assume A is Alice, B is Bob and I is an intruder. Without ambiguity, as parts of the messages, we also use A, B and I to denote the identities of the principals and the intruder.

K_A and K_B: Public-key of Alice and Bob respectively.

$\{m\}^p_{K_X}$: the encryption of message m with the public key of agent X.

N_A and N_B: two random numbers generated by A and B respectively.

1. $A \rightarrow B$: $\{N_A, A\}^p_{K_B}$
2. $B \rightarrow A$: $\{N_B, N_A \oplus B\}^p_{K_A}$
3. $A \rightarrow B$: $\{N_B\}^p_{K_B}$

The operator $\oplus$ has the following properties:

- $x \oplus y \approx y \oplus x$
- $(x \oplus y) \oplus z \approx x \oplus (y \oplus z)$

- $x \oplus x \approx 0$
- $x \oplus 0 \approx x$

where x, y and z are variables and 0 is a constant, which is called the *identity*.

We explain this protocol here. In this protocol, K_A and K_B are the public keys of A and B respectively. N_A and N_B are called *nonces* which are generated freshly by the agents A and B respectively. A *nonce* is a random message which is often used by a principal to show that a message is fresh. In the first step, A sends message $\{N_A, A\}^p_{K_B}$, which is encrypted by K_B, to B. This message can only be decrypted by B. So when B gets this message, he can decrypt it and get N_A and A. After that, in the second step, B encrypts N_B and $N_A \oplus B$ by A's public key, K_A, and sends them to A. Because only A knows what N_A is, after getting this message, only A can use $N_A \oplus B \oplus N_A$ to test whether this message is from B. After making sure this message is from B, in Step 3, A sends N_B encrypted with K_B back to B to show that A knows N_B now.

So in this protocol, A uses $N_A \oplus B$ to identify where the message in the second step is from(or who generated N_B) by using the property that $N_A \oplus B \oplus N_A = B$. We can see that only A is supposed to know N_B except for B himself because it is encrypted by K_A when B sends it out, and A is the person with whom B wants to talk. So B will believe that the principal who sends him N_B is the one he wants to talk with. If N_B is known by somebody else, the protocol is not secure anymore, because this person can pretend to be A to talk with B by sending $\{N_B\}^p_{K_B}$ to B

If the $\oplus$ function is interpreted as a free symbol(without the exclusive OR properties we listed above), in [38], the protocol was proved to be secure, in the sense that the intruder cannot learn N_B. But, if the $\oplus$ function has the properties of exclusive OR(Associativity, Commutativity, Unity and Nilpotence), there will be an attack:

1. $A \to I$: $\{N_A, A\}^p_{K_I}$

1'. $I[A] \rightarrow B$: $\{N_A \oplus B \oplus I, A\}^p_{K_B}$

2'. $B \rightarrow I[A]$: $\{N_B, N_A \oplus B \oplus I \oplus B\}^p_{K_A}$

2. $I \rightarrow A$: $\{N_B, N_A \oplus B \oplus I \oplus B\}^p_{K_A}$

3. $A \rightarrow I$: $\{N_B\}^p_{K_I}$

Where $I[A]$ means that the intruder I pretends to be A.

In this attack, $N_A \oplus B \oplus I \oplus B = N_A \oplus I$ because of the properties of exclusive OR. Because B does not know what N_A is supposed to be, when he receives anything of the form of $\{M, A\}^p_{K_B}$ for any value of M, he just sends $\{N_B, M \oplus B\}^p_{K_A}$ back, and in this attack M is $N_A \oplus B \oplus I$. Using the exclusive OR properties, A believes N_B was originally generated by the intruder and sends it back to the intruder to show her identity to the intruder. The intruder knows N_B finally. Thus after taking the exclusive OR properties into account, this protocol is not secure anymore.

We can find many examples of exclusive OR attacks in [53, 15].

Besides the theory of exclusive OR, Homomorphisms and Abelian groups also play important roles in cryptographic primitives.

For example, let's look at the NEEDHAM-SCHROEDER-LOWE Protocol [37] with ECB, where ECB(Electronic Code Book) is an encryption algorithm with the homomorphic property of $\{(M_1, M_2)\}_K = (\{M_1\}_K, \{M_2\}_K)$, this protocol is similar to the previous protocol:

K_A and K_B: Public-key of Alice and Bob respectively.

$\{m\}^p_{K_X}$: the encryption of message m with the public key of agent X.

N_A and N_B: two random numbers generated by A and B respectively.

1. $A \rightarrow B$: $\{N_A, A\}^p_{K_B}$

2. $B \rightarrow A$: $\{N_A, N_B, B\}^p_{K_A}$

3. $A \rightarrow B$: $\{N_B\}^p_{K_B}$

The attack is as follows [15]:

1. $A \to I$: $\{N_A, A\}^p_{K_I}$

1'. $I[A] \to B$: $\{N_A, A\}^p_{K_B}$

2'. $B \to I[A]$: $\{N_A, N_B, B\}^p_{K_A}$

2. $I \to A$: $\{N_A, N_B, I\}^p_{K_A}$

3. $A \to I$: $\{N_B\}^p_{K_I}$

3'. $I[A] \to B$: $\{N_B\}^p_{K_B}$

The intruder uses the property of $\{(M_1, M_2)\}^p_{K_x} = (\{M_1\}^p_{K_x}, \{M_2\}^p_{K_x})$ to extract $\{N_A, N_B\}^p_{K_A}$ from $\{N_A, N_B, B\}^p_{K_A}$ and combine $\{N_A, N_B\}^p_{K_A}$ and $\{I\}^p_{K_A}$ to construct $\{N_A, N_B, I\}^p_{K_A}$, such that A believes N_B is generated by the intruder and decrypts it for him.

Another famous example using the homomorphism property is a protocol based on RSA(which stands for Rivest, Shamir and Adleman) [50, 11]. In the RSA algorithm, there are a private key d and a public key e between A and B, which satisfies $d \equiv e^{-1} \mod \phi(n)$, where n is a multiplication of two big prime numbers, ϕ is Euler's totient function and $\equiv$ denotes the congruence relation. The principals use $\overline{M^e}$ ($\overline{M^e} \equiv M^e \mod n$ and $0 < \overline{M^e} \leq M^e$) to encrypt the message M and $(\overline{N^d})$ to decrypt the encrypted message N, because $(M^e)^d \equiv M \mod n$ from Euler's theorem. However, because M^e has the property of $M_1^e M_2^e = (M_1 M_2)^e$, if the intruder wants to know M when he knows $\overline{M^e}$, he can choose a constant C and send $\overline{C^e M^e}$ which is $\overline{(CM)^e}$ to the holder of the private key d. The holder will decrypt this message by d and send $\overline{CM}$ back to the intruder. Because the intruder knows what C is, he can get C^{-1} efficiently by the extended Euclidean algorithm. Then he can get M by multiplying $\overline{CM}$ by C^{-1}.

Next let's look at the Diffie-Hellman key exchange protocol [20, 57], which uses the properties of Abelian groups. A and B agree on a finite cyclic group G(a multiplicative group of integers modulo a prime number p) and a generator element g of G. The protocol is as follows:

1. A: generates a secret number a randomly.

1'. B: generates a secret number b randomly.

2. $A \to B$: $g^a \mod p$.

2'. $B \to A$: $g^b \mod p$.

3. A: computes $(g^b)^a \mod p$

3'. B: computes $(g^a)^b \mod p$

It uses Abelian groups here because exponentiation has the property that $(x^y)^z = (g^z)^y \mod p$. Hence $(g^a)^b = (g^b)^a = g^{ab} \mod p$ will be the secret key between A and B. But on the other hand, the intruder can also use this property to find an attack:

1. A: generates a secret number a randomly.

1'. B: generates a secret number b randomly.

1". I: generates a secret number i randomly.

2. $A \to I[B]$: $g^a \mod p$.

2'. $I[A] \to B$: $g^i \mod p$.

2". $B \to I[A]$: $g^b \mod p$.

2"'. $I[B] \to A$: $g^i \mod p$.

3. A: computes $(g^i)^a \mod p$

3'. B: computes $(g^i)^b \mod p$

This attack is called a *Man-in-the-Middle* attack. The intruder intercepts the public messages and replaces them by his own messages. In this attack, the intruder exploits the properties of Abelian groups to make A and B believe that they are talking to each other but not I. So finally, the intruder has two secret keys g^{ai} and g^{bi}. One is used between him and A and the other between him and B. After that, he can easily listen, decrypt, encrypt, and modify the messages between A and B.

We can find many other examples using Abelian groups and homomorphism properties in [15].

So the algebraic properties of the primitives need to be taken into account in protocol analysis in reality. But how does a protocol analysis method handle these properties?

Here we use backward search, which is often used to search for attacks, to explain how these properties are dealt with.

For example, in the NEEDHAM-SCHROEDER-LOWE Protocol [37] with the exclusive OR function, we set up a goal first: whether the intruder has the ability to know N_B. Because the intruder knows the private key K_I, if he/she knows $\{N_B\}^p_{K_I}$, then he/she knows N_B. The problem can be converted to a further problem: can the intruder know $\{N_B\}^p_{K_I}$? This is only one possible line of the attack, there are other possibilities.

If A receives a message of the form $\{N_B, N_A \oplus I\}^p_{K_A}$, she will send $\{N_B\}^p_{K_I}$ to the intruder. Thus the problem becomes whether the intruder can know $\{N_B, N_A \oplus I\}^p_{K_A}$.

Continuing to look at the protocol, if B receives a message of the form $\{X, A\}^p_{K_B}$(no matter what X is), he will send back the message $\{N_B, X \oplus B\}^p_{K_A}$ to A and this message can be heard by the intruder. So if we can find a proper X such that $\{N_B, X \oplus B\}^p_{K_A} = \{N_B, N_A \oplus I\}^p_{K_A}$, the problem will become whether the intruder can know $\{X, A\}^p_{K_B}$.

So here, we need to solve X from $\{N_B, X \oplus B\}^p_{K_A} = \{N_B, N_A \oplus I\}^p_{K_A}$. This kind of problem is called a *unification problem.* If $\oplus$ is a free function symbol, by the syntactic unification algorithm, there are no solutions because $I \neq B$. If $\oplus$ has the properties of exclusive OR, we could set $X = B \oplus N_A \oplus I$ as a solution, because $B \oplus N_A \oplus I \oplus B = B \oplus B \oplus N_A \oplus I = 0 \oplus N_A \oplus I$. Here we used the properties of the $\oplus$ function. This kind of unification problem is called an *E-unification problem*, where here E means the exclusive OR theory.

Let us go back to our protocol analysis. After solving M, our problem now is whether the intruder can know $\{B \oplus N_A \oplus I, A\}^p_{K_B}$. Considering the abilities of intruders, if he/she knows $B \oplus N_A \oplus I$ and A, he/she will know $\{B \oplus N_A \oplus I, A\}^p_{K_B}$ since K_B is public here. A is public information, so the problem becomes whether the intruder can know $B \oplus N_A \oplus I$.

Still considering the abilities of intruders. $\oplus$ is also a function which the intruder can use. The names I and B are public, so the problem is whether I can know N_A. Because A is public, the problem is whether I can know the pair of $\{N_A, A\}$. And K_I is known to the intruder, and the problem can be converted into whether the intruder can get $\{N_A, A\}^p_{K_I}$. This will be easy to get if A wants to talk to I from the first step of the protocol.

If we track the procedure, we can find the attack. However, when we do the backward search, there are many possible paths we need to consider and here we only consider the one that leads to an attack. Indeed, the search space will be huge, actually sometimes infinite. For example, other possible ways to know N_A are knowing $A \oplus N_A$, knowing $B \oplus N_A$, knowing $I \oplus N_A$ etc. How to reduce the search space is a very important problem we need to face. Finding an *E-unification algorithm* to efficiently generate a minimum complete set of unifiers is a way to reduce the search space.

E-unification algorithms are used to solve the E-unification problem, i.e. find the solution to make two terms equal modulo some theory E. For instance, suppose we have two terms $a \oplus b$ and $x \oplus y$, where a and b are constants and x and y are variables. A unification algorithm is used to find a solution of x and y such that $a \oplus b = x \oplus y$. If $\oplus$ is a free function symbol, the solution is $\{x \leftarrow a, y \leftarrow b\}$(we call this a *unifier* of the unification problem). But if $\oplus$ has the properties of exclusive OR, $\{x \leftarrow b, y \leftarrow a\}$ is also a solution because of commutativity. From the above analysis, we need to consider two possible subproblems separately. But there are many solutions, $\{x \leftarrow a \oplus z, y \leftarrow b \oplus z\}$ where z is a variable, is also a solution. How can we know these are all the possible solutions? We need to find a *complete set* of solutions, of which each unifier is one of the instance of one of the elements.

On the other hand, we can see $\{x \leftarrow a, y \leftarrow b\}$ is a special case of $\{x \leftarrow a \oplus z, y \leftarrow b \oplus z\}$, and so is $\{x \leftarrow b, y \leftarrow a\}$. If we consider all these three solutions, we will waste time on solving the two instance problems and our search space will blow up very quickly. So we need to find a *general* solution to make our search space as small as possible. Hence, finding a complete and small set of unifiers is very important in practice. We will give the technical definition of unifiers in Chapter 2.

In conclusion, unification algorithms for the theory of exclusive OR (with homomorphism) and Abelian groups (with homomorphism) are essential for cryptographic protocol analysis. It is important that these algorithms are efficient. Efficient unification algorithms have been developed for these theories [36, 28, 32, 33]. However, cryptographic protocol analysis also must deal with uninterpreted function symbols. For example, besides the encryption function, the hash function and concatenation functions are also commonly used for building other cryptographic primitives.

So it is important to have unification algorithms for these theories in combination with uninterpreted function symbols, which are called *general E-unification algorithms*. When uninterpreted function symbols occur in combination with these theories, the complete set of unifiers is not always a singleton, but it is finite [6]. 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 equational unification algorithms for these theories that are both efficient and create a small complete set of unifiers. The problem of deciding solvability of XOR/AG-unification problems is NP-complete [55]. An extra homomorphic operator will not decrease the complexity, since homomorphism may not happen in the problem. An extra homomorphic operator will not increase the complexity either by using the technique in [55], i.e. the problem of deciding solvability of XORH/AGh-unification problems is also NP-complete. The problem of counting the minimal complete set of unifiers of the XOR/AG-unification problem is in #P [29].

In [3] and [48], the authors present a more general idea of solving unification problems modulo monoidal theories, which include XOR/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 [54] and [10], the authors introduced a general algorithm to solve general unification problems. It can solve a general XORH/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.

Another standard technique for dealing with general AGh-unification is to create a convergent equational theory and apply Narrowing to solve the unification problem [19]. Similar to the first technique, this method is also highly nondeterministic and builds a highly redundant complete set of unifiers. Also termination of narrowing is also an issue if the convergent rewriting system cannot be created.

In Chapter 3 and Chapter 4, we try to overcome these problems about the efficiency in practice 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.

1.1.3 E-Unification in Maude-NPA

As we said above, adopting a good E-Unification algorithm is a way of reducing the search space. However, even if a small complete set of unifiers is guaranteed, the search space may grow fast during the procedure. Theoretically, searching for an attack is undecidable even for a bounded number of sessions with a theory for which the E-unification problem is decidable. For example, [17] presents an undecidability result for Abelian groups with homomorphism. so for a specific protocols analysis tool, the authors make efforts to reduce the search space, for example, by discarding infeasible branches as early as possible. The asymmetric unification algorithms presented in Chapters 5 and 6 are two unification algorithms used in Maude-NPA for helping to discard infeasible branches where the traditional efficient algorithms do not work.

Before explaining what the asymmetric unification problem is, let us briefly review the formal approaches to analyze cryptographic protocols.

Formal Method in Analysis of Cryptographic Protocols

Here we introduce three main formal approaches: the methods based on belief logic, the methods based on concrete state exploration and the methods based on abstract state exploration.

- Methods Based on Belief Logic

Belief logic was the first attempt to automate the verification of security protocols [49]. BAN (short for Burrows-Abadi-Needham) is the most significant verification tool based on belief logic. By using belief logic, BAN formalizes a protocol rule by a rule saying that if a principal received a message under some conditions, he/she may believe that the message is trustworthy.

This method is very simple and straightforward. The proof of the existence of an attack can be very short and found very fast. But because its simplicity is from not considering the intruder's abilities on the network, like computing and modifying messages, it has limited application. This is its big disadvantage in reality [49]. There are some other extended methods of BAN to deal with the intruder, but they are too complicated to be implemented [49].

- Methods Based on Concrete State Exploration

In the concrete state exploration approach, a protocol is characterized as the set of all its possible traces between concrete states(without variables). Each state represents some knowledge the intruder knows. Given a security protocol as input, the verification tool based on state exploration explores as many execution paths of the protocol as possible, checking at each state reached if some conditions hold [49]. Depending on the search method, the input can be some transition rules with initial states if forward search is adopted or a goal if backward search is adopted. Because the transition between states depends on some concrete logic conditions, the method of exploring states may be fully automatic. However, because every state is concrete, i.e. without variables, the number of states is finite. So this method can only deal with a bounded number of sessions.

- Methods Based on Abstract State Exploration

In abstract state exploration, a protocol is formalized similarly to the concrete state exploration approach, but the difference is that variables are allowed in describing states, which means that a state may represent an infinite number of concrete states. So the verification tools based on this method can deal with an unbounded number of sessions. But also because of the variables, the search space may blow up very fast, as we mentioned in the last section. Some verification tools based on abstract state exploration use Higher-Order Logic and some use First-Order Logic to formalize the protocol. The Higher-Order

Logic method is not automatic and needs interaction with a person, but the First-Order Logic method can be fully automatic.

Abstract exploration has more general application, but with lower efficiency. Here we focus on abstract state exploration. After introducing methods of analyzing protocols, we still need to consider two things: how to express a protocol and how to explore the states? We introduce a method to express the protocol and several ways to explore the states.

Strand spaces are one of the main methods to formalize protocols based on state exploration approaches.

A *strand* is a sequence of actions that a principal(including the intruder) can do in the protocol. If some message M is known by some principal, $+M$ denotes that the principal sends this message out or this principal computes this message. $-M$ denotes that the principal receives this message. For example, consider the NEEDHAM-LOWE protocol with exclusive OR:

1. $A \rightarrow B$: $\{N_A, A\}^p_{K_B}$
2. $B \rightarrow A$: $\{N_B, N_A \oplus B\}^p_{K_A}$
3. $A \rightarrow B$: $\{N_B\}^p_{K_B}$

B will send out a message $\{N_B, M \oplus B\}^p_{K_X}$ if he receives the message $\{M, X\}^p_{K_B}$ from any other party, where M, X and ${K_X}^p$ are variables. Because the intruder can listen, insert and delete any message in the network, the label of the message's source does not need to be indicated. So one strand for B is the sequence of $\{-\{M, X\}^p_{K_B}, +\{N_B, M \oplus B\}^p_{K_X}, -\{N_B\}^p_{K_B}\}$, which means if B receives message $\{M, X\}^p_{K_B}$, he will send out $\{N_B, M \oplus B\}^p_{K_X}$ and after that he may receive $\{N_B\}^p_{K_B}$.

Besides setting up the strands of the parties, we also need to give the strands of the intruder. For example, because all the identities and the public keys of the principals are public, the intruder can know them. So two of the strands of I are $\{+A\}$ and $\{-A, -X, \{X\}^p_{K_A}\}$, which means the intruder can know the identity of A automatically, and if the intruder knows A and any message X(a variable), then, he can encrypt this message by A's public key to get $\{X\}^p_{K_A}$.

In the runs of the protocol, strands are used to show what is happening in the channel. In the above protocol, a complete run of the protocol can be represented as $\{+\{N_A, A\}^p_{K_B}, -\{N_A, A\}^p_{K_B}, +\{N_B, N_A \oplus B\}^p_{K_A}, -\{N_B, N_A \oplus B\}^p_{K_A}, +\{N_B\}^p_{K_B}, -\{N_B\}^p_{K_B}\}$. Also the intruder can use his abilities to modify the messages. For example if the intruder wants to modify K_A, and the strand becomes $\{+\{N_A, A\}^p_{K_B}, -\{N_A, A\}^p_{K_B}, +\{\{N_A, A\}^p_{K_B}\}^p_{K_A}\}$, which means A sends out $\{N_A, A\}^p_{K_B}$, and the intruder receives it, then after that, the intruder encrypts $\{N_A, A\}^p_{K_B}$ with K_A and sends it out. So if the goal is to see whether the intruder can know N_B, we just need to know whether N_B appears in the strand.

In the previous section, we mentioned backward search for protocol analysis. Backward search is one of the main strategies for doing inference between states. After the states are given, including the initial states and a goal state, backward search starts from the goal to find a path to connect the goal state with some initial state. The big disadvantage of backward search is the search space's blowup. For example, if we want to know whether an intruder can know M, we can convert the problem to whether he/she can know $\{M\}^p_{K_I}$. And the problem can be converted further to whether he/she can know $\{\{M\}^p_{K_I}\}^p_{K_I}$.

Besides backward search, *forward search* and *Ordered Resolution* are two other main strategies for exploring the states. Ordered Resolution is used in ProVerif [8]

Forward search is a similar search strategy to backward search, except it starts from some initial state and works toward the goal. Like backward search, it also has the disadvantage that the search space can blow up. For example, suppose we have an intruder ability rule of $\{-A, -X, +\{X\}^p_{K_A}\}$ which means if the intruder knows message A and X, he/she can use the encryption function to get $\{X\}^p_{K_A}$. So if an intruder knows M, then because the encryption key of A is public, he can know $\{M\}^p_{K_A}$, and after that he can know $\{\{M\}^p_{K_A}\}^p_{K_A}$ because he/she knows A, and so on. This procedure will go on forever. From the intruder's other abilities, he also has other choices to search. The search space will become huge very quickly. Theoretically, the search space will blow up quickly just like backward search does, but because it is not goal oriented, it is not as good as backward search in practice.

Ordered Resolution is a fast inference system for dealing with first order sentences. If we use $I(X)$, which is a predicate to say that the intruder knows some knowledge X, then we can rewrite the intruder's ability $\{-A, -X, \{X\}^p_{K_A}\}$ to $I(A) \wedge I(X) \to I(\{X\}^p_{K_A})$, which is a first order sentence. Here $\to$ is the logical symbol for implication. So we can regard every strand as a first order sentence. If we negate the goal, the problem becomes an unsatisfiability problem and a good inference system can help solving it. Ordered Resolution is an efficient inference system to solve this kind of problem if we can convert the strands to first order sentences. Indeed, forward search and backward search are both based on Resolution but not ordered.

ProVerif [8] uses Ordered Resolution with selection to analyze cryptographic protocols. It is a fast and fully automatic verification tool which can deal with an infinite number of sessions. But ProVerif cannot deal with primitives with algebraic properties except for some special ones and this is a big disadvantage. It performs approximation to deal with nonces, so it can produce false negatives [8], which means the attacks ProVerif finds may not be true attacks. As we mentioned above, this is a sacrifice for improving efficiency.

The Maude-NPA is another verification tool for protocol analysis which was designed and implemented by Meadows [43]. It was first written in Prolog and later rewritten in Maude. It is one of the first provers which used automated theorem proving to analyze the security of protocols [49]. It uses strand spaces to express the runs of protocols and the backward search strategy to search for attacks. This tool can be used to test not only secrecy but also authentication and handle the algebraic properties of cryptographic primitives. But the search space may blow up quickly.

Searching in Maude-NPA

For improving the searching efficiency, the author of Maude-NPA adopts many techniques for automatically identifying unreachable states and redundant states and introducing the super-lazy intruder [25]. These techniques [25] includes

- the generation of *formal grammars* [24, 42, 23];
- early detection of inconsistent states;
- reducing the number of logical variables present in a state;
- giving priority to input messages in strands;
- a relation of transition subsumption (to discard transitions and states already being processed in another part of the search space); and
- delaying the generation of substitution instances as much as possible by defining a super-lazy intruder.

Though the search methods and algorithms are tool-dependent, the above techniques have common syntax and semantics such that they can be adapted into other tools [25]. Indeed, some of the above techniques also can be seen in [56, 16, 7] etc.

But these techniques may be incompatible with the unification methods. Here we want to talk about the incompatibility between early detection of inconsistent states and the E-unification algorithms.

Inconsistent states are the states which are not supposed to be there in real runs of the protocol.

For example [22], in the following protocol, Bob wants to talk to Alice by sending his own nonce N_B encrypted by Alice's public key and Alice sends her own nonce N_A encrypted by Bob's public key to Bob upon receiving his request. Finally, Bob will send Alice back $N_A \oplus N_B$ to prove he is Bob by showing he knows N_A already. Here they use the property of exclusive OR $N_A \oplus N_B \oplus N_A \approx N_B$.

Example 1.1

1. $B \to A$: $\{N_B\}^p_{K_A}$
2. $A \to B$: $\{N_A\}^p_{K_B}$
3. $B \to A$: $N_A \oplus N_B$

So from Alice's perspective, informally, her strand space is:

$$\{-\{X\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus X)\}$$

If we find an instance of the protocol by applying the substitution $X \leftarrow N_A \oplus Y$, we will get

$$\{-\{N_A \oplus Y\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus N_A \oplus Y)\}$$

Here we can see N_A is supposed to be generated when A sends out $\{N_A\}^p_{K_B}$. But N_A appears before A sends out $\{N_A\}^p_{K_B}$. So if any state containing the above strand is an inconsistent state. Maude-NPA will discard the state to avoid redundant searching.

However, if we further instantiate Y by $Y \leftarrow N_A \oplus N_B$, then we can get:

$$\{-\{N_A \oplus N_A \oplus N_B\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus N_A \oplus N_A \oplus B)\}$$

which is:

$$\{-\{N_B\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus B)\}$$

We can see any state containing this strand is a legal execution of the protocol. This means discarding inconsistent states may prune a legal branch, which makes the search algorithm incomplete.

Hence, though standard exclusive OR unification algorithm (here we call the traditional unification algorithm like we introduced in Section 1.1.2 a *standard one*) gives a small complete set of unifiers, it hinders the optimization of detecting inconsistent states.

Without using the standard exclusive OR unification algorithm, Maude-NPA has its own way to overcome this difficulty, which is called *folding variant narrowing* [27]. Before talking about this, we need to briefly introduce the *finite variant property*.

E-variants and the finite variant property were first introduced by Hubert Comon-Lundh and Stéphanie Delaune in [14]. In [27] and [26], the authors expressed the method to compute the compete set of unifiers of E-unification problems by using variant narrowing which is based on the finite variant property of E.

We explain this concept by taking the exclusive OR theory. The exclusive OR theory is first decomposed into (R, E'), where E' is the $\mathcal{AC}$ theory and R is a set of rewrite rules for the properties $\{X \oplus 0 \approx X,\ \ X \oplus X = 0\}$, which is $\{X \oplus 0 \rightarrow X,\ \ X \oplus X \rightarrow 0, X \oplus X \oplus Y \rightarrow Y\}$. An R, E*-variant* [22] of term t is a pair (t', θ), where t' is a term and θ is a substitution, if $t\theta \downarrow_{R,E} =_E t'$.

For example, for the term $N_A \oplus X$, we have four R, E-variants: $(N_A \oplus X, id)$, $(0, [X \leftarrow N_A])$, $(N_A, [X \leftarrow 0])$, and $(Y, [X \leftarrow N_A \oplus Y])$. These four R, E-variants compose *the complete set*[22, 27, 26] of R, E-variants of $N_A \oplus X$. This means for any R, E-variant (t'', θ'') of $N_A \oplus X$, we always can find one of these four R, E-variants, (t', θ'), and a substitution σ, such that $t'' =_E t'\sigma$ and $\theta'' =_E \theta'\sigma$. If $t'' = N_B$ and $\theta'' = [X \leftarrow N_A \oplus N_B]$, then we can choose $(t', \theta') = (Y, [X \leftarrow N_A \oplus Y])$ and let $\sigma = Y \leftarrow N_B$.

A theory has the *finite variant property* if a complete set of variants is finite for every term.

Roughly, in [26] and [27], if we have an exclusive OR unification problem, $t \overset{?}{=} t'$, we will compute the complete set of R, E-variants of t and t' and get $\{(t_1, \theta_1), \cdots, (t_n, \theta_n)\}$ and $\{(t'_1, \theta'_1), \cdots, (t'_m, \theta'_m)\}$, (the exclusive OR theory has the finite variant property [14]) then we can compute all the E-unification problems (namely $\mathcal{AC}$-unification problems) $t'_i \overset{?}{=} t'_j$ for all $0 \leq i \leq n$ and $0 \leq j \leq m$. Then all the unifiers compose the complete set of unifiers of the exclusive OR unification problem $t \overset{?}{=} t'$.

Thanks to the finite variant property, Maude-NPA can remove the inconsistent state without losing the completeness of the search algorithm. Let us go into the details as follows by taking on Example 5.1.

The key idea [22] of removing the inconsistent state without losing the completeness of the search algorithm is to divide $N_A \oplus X$ into two possible forms $N_A \oplus X$ and Z (it has four forms according to the formal definition, here, for simplicity, we use two of them to explain the idea). Then we can assume neither of them are reducible. The state becomes two independent states:

One contains the following strand:

$$\{-\{X\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus X)\}$$

with the substitution *id* and the other one contains the following strand:

$$\{-\{N_A \oplus Y\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus N_A \oplus Y)\}$$

with the substitution $X \leftarrow N_A \oplus Y$.

The second state is an inconsistent state, and Maude-NPA can safely discard it, because in the future, we do not allow any substitution to make Y contain N_A which will cancel N_A in $N_A \oplus Y$. So Maude-NPA can only focus on the first state to continue the search.

As we discussed in the last section, we need a unification algorithm to unify the states and transition rules. But here we need the states to be irreducible for any substitution. So the problem becomes finding a complete set of unifiers U for E-unification problem $t \overset{?}{=}_E t'$, where t is a term from some transition rule and t' is a term in some state, such that for every $\sigma \in U$, $t\sigma =_E' t'\sigma$ and $t'\sigma$ is irreducible by R for theory E, where (R, E') is the decomposition of E.

For example, if we have an exclusive OR unification problem $y \oplus a \overset{?}{=} x \oplus b$ and the theory (R, E') which are mentioned above, then although the unifier $\theta = [x \leftarrow y \oplus a \oplus b]$ is a unifier of $y \oplus a \overset{?}{=} x \oplus b$, it does not satisfies the requirement that $(x \oplus b)\theta$ is reducible, because $x \oplus b$ becomes $y \oplus a \oplus b \oplus b$, which can be reduced to $y \oplus a$. but $\sigma = [y \leftarrow x \oplus b \oplus a]$, which is equivalent to θ, is a solution. For clarity, we call this kind of problem an asymmetric unification problem and the unifiers are called asymmetric unifiers. We write the problem as $t =_\downarrow t'$. To distinguish traditional unification problems, we call them standard unification problems.

The asymmetric unification problem can be solved by variant narrowing: just convert the problem to a set of standard unification problems $t_i \overset{?}{=}_E t'$, where (t_i, θ_i) is the E-variant of t.

Though this technique can overcome the problem of detecting inconsistent states, it still has some disadvantages:

- Computing variants is not very efficient. This is totally based on narrowing, which is highly non-deterministic.

- The complete set of unifiers directly got by this method is usually not a small set.

There are two reasons: the first one is the complete set of variants generated by narrowing may be much larger than the minimal complete set of variants of the term in the right side [27]; the second one is $\mathcal{AC}$ unification problems may have doubly exponential many unifiers [31].

In Chapters 5 and 6, we introduce two terminating, sound and complete asymmetric unification algorithms for exclusive OR and Abelian groups, to avoid the above disadvantages. We already implemented the algorithms for exclusive OR in Maude and it is being built into Maude-NPA. The preliminary results of comparison between it with other methods is very encouraging.

1.2 Work Done in This Thesis

1.2.1 E-Unification Algorithms for Exclusive OR with Homomorphism and Abelian Groups with Homomorphism

In Chapter 3 and 4, we solve the problems of finding a small complete set of unifiers for the E-unification problem with uninterpreted function symbols modulo exclusive OR with homomorphism and Abelian groups with homomorphism by devising two sets of inference rules which are simple, easy to be implemented, very deterministic in practice, and produce a small complete set of unifiers for unification problems.

We introduced the concept of *unconstrained variable* in the exclusive OR unification problem, and *purified* the unification problem such that all the terms are pure to avoid generating loops. Intuitively, an *unconstrained variable* is a variable which never occurs under an uninterpreted function symbol and a *pure* term is a term in which no exclusive

OR operator occurs.

For example, we will purify $x \oplus f(x+y) \overset{?}{=} 0$ by converting it into $x \oplus f(v) \overset{?}{=} 0$ and $v \oplus x \oplus y \overset{?}{=} 0$, since $f(x+y)$ is not pure. After purifying the equations, we will solve the unconstrained variables one by one. For the above example, we solve x from $x \oplus f(v) \overset{?}{=} 0$ and get $x \leftarrow f(v)$. The only equation left $v \oplus x \oplus y \overset{?}{=} 0$ becomes $v \oplus f(v) \oplus y \overset{?}{=} 0$. At last, we solve y and get $y \leftarrow v \oplus f(v)$.

Based on this main idea, we give four rules which are easy to be implemented and efficient in practice. For generating a smaller complete set of unifiers and avoiding unnecessary non-deterministic steps, we also give several auxiliary rules which can be used to increase the efficiency without losing soundness, completeness and termination.

Because of the similarity between exclusive OR and Abelian groups, we transplanted our algorithm into Abelian groups with Homomorphism. Because we only consider the free Abelian groups, we borrow the technique from [34] for solving linear diophantine equations in our algorithm to solve the unconstrained variables.

We also prove that all the inference systems are terminating, sound and complete. We also give some examples to explain how our inference system works.

Although our algorithms consider homomorphism, as a special case, our two inference systems can be used in the corresponding theory without homomorphism without losing efficiency, soundness, completeness and termination.

We have implemented our algorithms in Maude.

1.2.2 Asymmetric E-Unification Algorithms for Exclusive Or and Abelian Groups

In Chapters 5 and 6, we present an algorithm for solving asymmetric unification problems modulo exclusive OR and Abelian groups.

In the algorithm for asymmetric exclusive OR unification, we start from a standard unifier of the unification problem (we can get this by using the algorithm presented in Chapter 3), and search for corresponding asymmetric unifiers fulfilling the constraints from the asymmetric unification problem.

For example, suppose we have the problem $\{a \oplus y \oplus z =_{\downarrow} x \oplus a, a \oplus z =_{\downarrow} z \oplus a\}$ and we got a standard unifier $\sigma = [x \leftarrow a \oplus b \oplus y \oplus z]$. In this case, since $x \leftarrow a \oplus b \oplus y \oplus z$ will make $x \oplus a$ reducible, we need another unifier to fit this constraint. The idea is to non-deterministically try $y \leftarrow a \oplus v$ and $z \leftarrow a \oplus v'$, where v and v' are fresh variables. We can see here $\sigma_1 = [y \leftarrow a \oplus v, x \leftarrow v \oplus b \oplus z]$ is an asymmetric unifier but $\sigma_2 = [z \leftarrow a \oplus v', x \leftarrow v' \oplus b \oplus y]$ is not an asymmetric unifier because it does not fulfill the constraint which is that $z \oplus a$ should be irreducible. Since σ_1 and σ_2 are both equivalent to σ modulo exclusive OR, we can just chose σ_1 as the final asymmetric unifier.

The above idea is the essential part of our algorithm, but we still need to deal with other issues:

- If we could not find any equivalent unifiers after trying every possibility, does this mean there is no equivalent asymmetric unifier?

- If there is no equivalent asymmetric unifier, does that mean there is no instance of the standard unifier which is an asymmetric unifier?

- If there are some instances of the standard unifier which are asymmetric unifiers, how do we find them?

In Chapter 5, we completely solve these issues and use inference rules *Useless Branching, Decomposition Instantiation* and *Elimination Instantiation* to help us find all the possible instances. Formally, we give six rules which are easy to be implemented and efficient in practice. For generating a smaller complete set of unifiers and avoiding unnecessary non-deterministic steps, we also give several auxiliary rules which can be used to increase efficiency without losing soundness, completeness and termination.

For solving the asymmetric unification problem modulo Abelian groups, in Chapter 6, we use similar techniques to the branching rules used in solving the asymmetric exclusive OR unification problem. But the branching rules will not terminate. For avoiding loops, we adopt a *bound set* to remember the depth of the branch such that we can prevent the looping by setting a bound of the depth of the branch by a user. The disadvantage of this is that it will make our algorithm incomplete. When this happens, we will call the variant narrowing algorithm to get the complete set of unifiers(the theory of Abelian groups has a term rewriting system which has the finite variant property [14]). These ideas will be presented in Chapter 7. We also prove our whole algorithm (including calling the variant narrowing algorithm) will terminate, and is sound and complete.

1.3 Overview of This Thesis

In Chapter 2, we give the basic notation which is used in following chapters. In Chapter 3, we introduce the new inference system for general unification for exclusive OR with homomorphism and give the proof of the termination, soundness and completeness of the inference system. In Chapter 4, we introduce the new inference system for general unification for Abelian groups with homomorphism and also give the proof of the termination, soundness and completeness. In Chapter 5 and Chapter 6, we present the algorithms for finding an asymmetric unifier from a standard unifier of an asymmetric exclusive OR / Abelian group unification problem and the same as before, we give the proof of the termination, soundness and completeness of our algorithms. In Chapter 7, we give conclusions and future work.

Chapter 2

Basic Notation

Here we give some basic terminology which we will use in the following chapters. Most of the standard definition are from [6, 19, 22].

2.1 Standard Unification Preliminaries

A *signature*, $\mathbb{F}$, is a finite set of function symbols with fixed-arity, where a constant is a function with 0 arguments. Let $\mathbb{V}$ be a set of variables. We say t is a term over $\mathbb{V}$ and $\mathbb{F}$ 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. We say that a term u is a *subterm* of t and write $t[u]$ if either $t = u$ or if $t = f(t_1, \cdots, t_n)$ and u is a subterm of t_i for some i. If t and s are two terms, then we use $t[s]$ to denote that s occurs in t as a subterm.

When needed, it is customary to specify positions of subterms as sequences of integers indicating the path to the subterm in the tree representation of the term. If p is a position and t is a term, we would write $t|_p$ for the subterm of t at position p; otherwise, $f(t_1, \cdots, t_n)|_{i:p} = t_i|_p$. For example, if $t = f(x, g(a, b))$, then $t_2 = g(a, b)$, $t_{21} = a$ and $t_{22} = b$. We use $Pos_{\mathbb{F}}(t)$ to denote the set of all the positions of t. So

$Pos_{\mathbb{F}(f(x,g(a,b)))} = \{\epsilon, 1, 2, 21, 22\}$, where ϵ is the empty sequence. If we have two terms t and s, and a position p, then we use $t[s]|_p$ to denote the term t', which is the same as t except at the position p, $t|_p$ is replaced by s. For example, if $t = f(x, y)$, then $t[g(a)]_{11} = f(g(a), y)$.

We use $\mathbb{T}_{\mathbb{F}}(\mathbb{V})$ to denote the set of all terms over $\mathbb{V}$ and $\mathbb{F}$. 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 $Top(t)$.

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. $\{x_1 \leftarrow t_1, x_2 \leftarrow t_2, \cdots, x_n \leftarrow t_n\}$ 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 $Dom(\sigma) = \{x \mid x \leftarrow t \in \sigma \text{ and } x\sigma \neq x\}$ and $Range(\sigma) = \{t \mid x \leftarrow t \in \sigma \text{ and } x \in Dom(\sigma)\}$. 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 $\sigma \vDash \Gamma$. *Composition* of two substitutions is written $\sigma\theta$ and defined by $t\sigma\theta = (t\sigma)\theta$.

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 $\mathbb{T}_{\mathbb{F}}(\mathbb{V}) \times \mathbb{T}_{\mathbb{F}}(\mathbb{V})$. 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 $\mathbb{T}_{\mathbb{F}}(\mathbb{V})$ that is closed under substitution and contains E.

Example 2.1 In the Exclusive OR theory (XOR), the operator($\oplus$) has four properties: Associativity, Commutativity, Existence of Unity and Nilpotence. We can model them as

$$\begin{aligned} \{\ \ & x \oplus (y \oplus z) \approx (x \oplus y) \oplus z, \\ & x \oplus y \approx y \oplus x, \\ & x \oplus 0 \approx x, \\ & x \oplus x \approx 0 \ \ \} \end{aligned}$$

If two terms, t and s, are equal with respect to a theory $=_E$, we write it as $t =_E s$. For example, though $a \oplus b \neq b \oplus 0 \oplus a$ syntactically, $a \oplus b =_{XOR} b \oplus 0 \oplus a$.

Definition 2.2 (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*

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

*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** if and only if $U_E(\Gamma) \neq \emptyset$. If E is an empty set, we call the E-unification problem a **syntactic unification problem**.*

Let $\Gamma = \{t_1 \overset{?}{=}_E s_1, \cdots, t_n \overset{?}{=}_E s_n\}$. We use $\mathbb{V}_\Gamma$ to denote the set of variables in Γ. And we use $[\sigma]$ to denote the set $\{x_1 \overset{?}{=} t_1, x_2 \overset{?}{=} t_2, \cdots, x_n \overset{?}{=} t_n\}$ if $\sigma = \{x_1 \leftarrow$

$t_1, x_2 \leftarrow t_2, \cdots, x_n \leftarrow t_n\}$. For example, if we have $\sigma = x \leftarrow f(g(a), b), y \leftarrow g(f(x, a))$, then $[\sigma] = \{x \overset{?}{=} f(g(a), b), y \overset{?}{=} g(f(x, a))\}$.

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

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

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 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 2.3 (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 some equational theory $=_E$, we will call two E-unification problems *equivalent modulo E* if they have the same set of unifiers modulo theory $=_E$.

Definition 2.4 (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 $s \neq^? t$.

A *rewrite rule* is an oriented pair $l \to r$ where $l \notin \mathbb{V}$ and $l, r \in \mathbb{T}_{\mathbb{F}}(\mathbb{V})$. A *rewrite system* $\mathbf{R}$ is set of rewrite rules. The *rewriting relation* on $\mathbb{T}_{\mathbb{F}}(\mathbb{V})$, written $t \mapsto t'$ holds between t and t' if and only if there exist $p \in Pos_{\mathbb{T}}(t)$, a rule $l \to r \in \mathbf{R}$ and a substitution σ, such that $t|_p = l\sigma$ and $t' = t[r\sigma]|_p$. We call t' is *reduced* from t. If no such t' exist, we say t is *irreducible*.

For example, if we have a term $t = f(g(a \oplus b \oplus a, 0))$ and a rewrite rule $X \oplus Y \oplus X \to Y$ in $\mathbf{R}$, then we have $f(g(a \oplus b \oplus a, 0)) \mapsto f(g(b, 0))$. Here $\sigma = [Y \leftarrow b, X \leftarrow a]$.

If we have a sequence, $t_1 \mapsto t_2 \mapsto \cdots \mapsto t_n$, such that t_n is irreducible, then t_n is called the *normal form* of t_1, and is denote by $t_1 \downarrow$. For simplification, we write this sequence as $t_1 \mapsto^* t_n$ if $n \geq 0$. So for an irreducible term t, $t \downarrow = t$. For a rewrite system $\mathbf{R}$, if for any term t, $t \downarrow$ exists, then we say $\mathbf{R}$ is terminating. If for any two terms t and t' and another term s such that $s \mapsto^* t$ and $s \mapsto^* t'$, there exists another term s', such that $t \mapsto^* s'$ and $t' \mapsto^* s'$, then we call $\mathbf{R}$ is *confluent*. A rewrite system is *convergent* if it is terminating and confluent.

A convergent rewrite system can help us easily to compare two syntactically different terms modulo a theory. For a simple example, if we have $f(x \oplus x)$ and $f(a \oplus a)$, we do not know whether these two terms are equivalent in the theory of $\{X \oplus X \approx 0\}$. If we have rule $X \oplus X \to 0$, then we can rewrite $f(x \oplus x)$ to $f(0)$ and rewrite $f(a \oplus a)$ to $f(0)$. In this case, we can just direct compare their normal forms $f(0)$ and $f(0)$ to see whether they are equal syntactically.

We sometimes omit E, $\mathbb{F}$, and Γ, if they are clear from the context.

2.2 Asymmetric Unification Preliminaries

Let theory E be set of identities. We can divide E into two parts: E' and E''. Suppose there exist a convergent rewrite system R for E'', then we call (R, E') is a *decomposition* of E. For convenience, sometimes, we use $R \cup E'$ to denote the theory E.

Definition 2.5 (Asymmetric Unification Problem, Standard Unifiers and Asymmetric Unifiers) *Let (R, E') be a decomposition of the theory E. An **asymmetric E-unification problem** is a set of equations $\Gamma = \{s_1 =_{\downarrow} t_1, \cdots, s_n =_{\downarrow} t_n\}$. A **standard E-Unifier** of Γ is a substitution σ, such that $s_1\sigma \downarrow =_{E'} t_1\sigma \downarrow$, $\cdots$, $s_n\sigma \downarrow =_{E'} t_n\sigma \downarrow$, namely, it is a $(R \cup E')$ unifier. An **asymmetric E-unifier** of Γ is a standard E-unifier δ, such that $s_1\delta \downarrow =_{E'} t_1\delta$, $\cdots$, $s_n\delta \downarrow =_{E'} t_n\delta$, where $t_1\delta, \cdots, t_n\delta$ are in normal form in terms of R.*

Here the standard E-unifier is as same as the E-unifier in the last section. We see that an asymmetric XOR unifier is also a standard XOR unifier. The only difference between these two unifiers are whether the result makes the right hand sides of the unification problem reducible or not. So, in Chapter 5 and 6, if we talk about a unifier σ without indicating asymmetric or standard, σ may be an asymmetric unifier or standard unifier.

If R and E' are clear, for convenience, we will omit R, E' and only write the normal form.

Example 2.6 We divide the exclusive OR theory to E' and R, where E' is $\mathcal{AC}$, namely:

- $x \oplus y \approx y \oplus x$ [commutativity];
- $x \oplus (y \oplus z) \approx (x \oplus y) \oplus z$ [associativity].

and R the rewrite system in terms of UN, namely

- $x \oplus 0 \to x$ [Unity];

- $x \oplus x \to 0$ [Nilpotence];
- $x \oplus y \oplus x \to y$ [Extension of Nilpotence].

Then if we have a unification problem $x \oplus a =^? y \oplus b$, then $y \leftarrow b \oplus a \oplus x$ is a standard unifier, but not asymmetric unifier. But $x \leftarrow y \oplus b \oplus a$ is a standard unifier and an asymmetric unifier.

For an asymmetric unifier σ, without ambiguity, we still have $[\sigma] = \{x \overset{?}{=} T \mid x \leftarrow T \in \sigma\}$.

For a substitution σ and a variable assignment $x \leftarrow S$, define $[x \leftarrow S]\sigma = [x \leftarrow S\sigma]$. If δ is a unifier of Γ, define $(\delta\sigma)_\Gamma = \{x \leftarrow S\sigma \mid x \leftarrow S \in \delta\} \cup \{y \leftarrow T \mid y \notin Dom(\delta), y \in Vars(\Gamma)$ and $y \leftarrow T \in \sigma\}$. If Γ is clear in the context, we will omit Γ and only write $\delta\sigma$.

Here $\delta\sigma$ is a little different from the traditional substitution composition, because here we will not add all the variable assignments into the result if the variables are not in the domain of δ. For clarity, in Chapter 5 and Chapter 6, we will use $\delta \circ \sigma$ to denote traditional substitution composition.

Example 2.7 In solving exclusive OR unification problems, if we have :

$$\begin{aligned} \Gamma &= \{x \oplus f(x \oplus y \oplus z) =_\downarrow 0\} \\ \delta &= [x \leftarrow f(v),\ y \leftarrow z \oplus v \oplus f(v)] \\ \sigma &= [z \leftarrow a, v \leftarrow b] \end{aligned}$$

Then

$$\delta\sigma = \{x \leftarrow f(b), y \leftarrow a \oplus b \oplus f(b), z \leftarrow a\}$$

and

$$\delta \circ \sigma = \{x \leftarrow f(b), y \leftarrow a \oplus b \oplus f(b), z \leftarrow a, \mathbf{v} \leftarrow \mathbf{b}\}$$

Definition 2.8 (Equivalence of Unifiers Modulo E) *For an asymmetric unification problem* Γ*, an* E*-unifier* σ *is* ***more general modulo E*** *than another* E*-unifier* θ*, denoted* $\sigma \leq_E^\Gamma \theta$ *if there exists a substitution* τ*, such that for every variable* x *in* Γ*,* $x(\sigma \circ \tau) =_E x\theta$*. Note that the relation* $\leq$ *is a quasi-ordering, i. e. reflexive and transitive. We say two E-unifier* δ_1 *and* δ_2 *for* Γ *are* ***equivalent modulo*** E *if* $\delta_1 \leq_E^\Gamma \delta_2$ *and* $\delta_2 \leq_E^\Gamma \delta_1$.

Definition 2.9 (Complete Set of Standard Unifiers) *A* ***complete set of standard unifiers*** *of* Γ *is a set of substitutions* Σ*, such that each* $\sigma \in \Sigma$ *is a standard* E*-unifier of* Γ *and for any unifier* δ *of* Γ*, there exists* $\theta \in \Sigma$*, such that* θ *is more general modulo* E *than* δ*, namely* $\theta \leq_E^\Gamma \delta$*.*

Definition 2.10 (Complete Set of Asymmetric Unifiers) *A* ***complete set of asymmetric unifiers*** *of* Γ *is a set of substitutions* Σ*, such that each* $\sigma \in \Sigma$ *is an asymmetric* E*-unifier of* Γ *and for any asymmetric unifier* δ *of* Γ*, there exists* $\theta \in \Sigma$*, such that* θ *is more general modulo* E *than* δ*, namely* $\theta \leq_E^\Gamma \delta$*.*

Chapter 3

Efficient General Unification for Exclusive OR with Homomorphism

3.1 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 [41]. 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 [24]. The example we gave in Section 1.1.2 from [12] gave an attack of the protocol based on the property of the exclusive OR theory. Abelian groups are also important, because they can model products, such as the product of exponents in Diffie Hellman. Another common property of cryptographic algorithms is a homomorphisms over an exclusive OR or an Abelian group operator. For example, RSA has the property ${m_1}^e{m_2}^e = (m_1m_2)^e$, 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 [52, 35] . 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.

In conclusion, unification algorithms for the theories of exclusive OR (with homomorphism) and Abelian groups (with homomorphism) are essential for cryptographic protocol analysis. It is important that these algorithms are efficient. Efficient unification algorithms have been developed for these theories [36, 28, 32, 33]. However, cryptographic protocol analysis also must deal with uninterpreted function symbols, because in one protocol, there may be many uninterpreted function symbol involved, like the hash function. So it is important to have unification algorithms for these theories in combination with uninterpreted function symbols. Without uninterpreted function symbols, a minimum complete set of unifiers for the unification problem modulo exclusive OR or Abelian groups is always a singleton. When uninterpreted function symbols occur in combination with these theories, the minimum complete set of unifiers is not always a singleton, but it is finite.

There are two standard approaches for dealing with these algebraic properties in combination with uninterpreted function symbols. One way is to find an efficient preliminary unification algorithm without considering uninterpreted function symbol and combine it with an efficient syntactic unification algorithm for uninterpreted function symbols. This combination algorithm for exclusive OR can be found in [5]. The second technique is to convert the algebraic properties to a convergent rewriting system and apply narrowing to solve the unification problem [19]. But both of these techniques are highly nondeterministic and generate a highly redundant complete set of unifiers which is worse in cryptographic protocols analysis.

In the following two chapters, 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 can compare our work to [62]. That work is based on the combination method, and also has the goal of an efficient unification algorithm for exclusive OR unification. We think our inference rules are simpler and easily extended to other equational theories.

In this chapter, we have developed a sound, complete and terminating set of inference rules for exclusive OR with homomorphism(ACUNh), along with uninterpreted function symbols. We have implemented our inference rules in Maude [13]. These inference rules also apply to exclusive OR without homomorphism. We have designed them in such a way that they can be extended to Abelian groups.

Generally, our algorithm introduces the concept of unconstrained variables and solve the unconstrained variables based on the property that $x \oplus t =_{ACUNh} 0$ if and only if $x =_{ACUNh} t$. The inference system includes four necessary rules: Trivial, Variable Substitution, N-Decomposition, and Annulization. A simpler version of these inference rules (without Annulization and with simpler conditions for other rules) also applies to exclusive OR without homomorphism.

Here we give the outline of this chapter. In Section 2, we give preliminary knowledge about the ACUNh theory. In Section 3, we present a convergent rewriting system modulo $\mathcal{AC}$ for ACUNh. Next, we present our inference system in two parts: 1.a preprocessing step that transforms a unification problem into a purified form is given in Section 4; 2. the inference rules which are used to solve the unification problem are given in Section 5 and 6, where section 5 presents the necessary rules and Section 6 presents a kind of auxiliary rules. The proofs of termination, soundness and completeness of our inference system are given in Sections 7, 8 and 9 respectively. Section 10 gives some concrete auxiliary rules. Section 11 shows some of our implementation results. The conclusion is given in Section 12.

3.2 Preliminaries

Next, we introduce the basic notation related to the theory of exclusive OR with a homomorphism. We simplify this theory as **ACUNh**.

Here the signature $\mathbb{F}$ is composed of $\{0, \oplus, h\} \cup \mathbb{F}'$, where 0 is a constant, $\oplus$ a binary symbol, h a unary function symbol and $\mathbb{F}'$ a collection of uninterpreted function symbols, with the following properties:

- $(x \oplus y) \oplus z \approx x \oplus (y \oplus z)$ [Associativity - $\mathcal{A}$ for short];
- $x \oplus y \approx y \oplus x$ [Commutativity - $\mathcal{C}$ for short];
- $x \oplus 0 \approx x$ [Existence of Unity - $\mathcal{U}$ for short];
- $x \oplus x \approx 0$ [Nilpotence - $\mathcal{N}$ for short];
- $h(x \oplus y) \approx h(x) \oplus h(y)$ [Homomorphism of h - $\mathcal{H}$ for short] .

We say a term t is *pure*, if $\oplus \notin Sym(t)$ and h does not occur under the top symbol of t. We call a term *h-term* if the top function symbol of this term is h. 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. A term t is in PURE SUM form if t has the form $t_1 \oplus t_2 \oplus \cdots t_n$, and: (i) for every i, t_i is a pure term, (ii) in $\{t_1, \cdots, t_n\}$, there is at most one h-term and (iii) $n > 0$. We say an equation $\mathbf{S} \overset{?}{=} 0$ is in PURE SUM form, if $\mathbf{S}$ is a PURE SUM.

Because of the properties of exclusive OR, we only consider equations of the form $s =^? 0$ where s is a term. So for convenience, we will write an **ACUNh**-unification problem as $\{s_1 \overset{?}{=} 0, s_2 \overset{?}{=} 0, \cdots s_n \overset{?}{=} 0\}$, where each s_i is a term.

3.3 Rewriting System $\mathfrak{R}_{ACUNh}$

Before we introduce our inference system, we give a convergent rewriting system $\mathfrak{R}_{ACUNh}$ for **ACUNh** modulo associativity and commutativity:

- $x \oplus x \to 0$;
- $x \oplus 0 \to x$;
- $h(x) \oplus h(y) \to h(x \oplus y)$;
- $h(0) \to 0$;
- $x \oplus y \oplus x \to y$;
- $h(x) \oplus y \oplus h(z) \to h(x \oplus z) \oplus y$.

The last two rewriting rules are extensions of $x \oplus x \to 0$ and $h(x) \oplus h(y) \to h(x \oplus y)$.

In our inference system, all terms will get reduced by $\mathfrak{R}_{ACUNh}$ modulo **AC**. So unless stated, all the terms are in reduced form by $\mathfrak{R}_{ACUNh}$ modulo **AC**.

3.4 Initialization

Lemma 3.1 *Any set of equations* Γ *can be purified to be a set of equations* Γ' *in* PURE SUM *form, which is a conservative extension of* Γ*.*

Proof. We can use the following inference rule, which we call **Purify**, to prove this:

$$\frac{\Gamma \cup \{S \oplus t[s] \stackrel{?}{=} 0\}}{\Gamma \cup \{S \oplus t[x] \stackrel{?}{=} 0\} \cup \{x \oplus s \stackrel{?}{=} 0\}}$$

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

Purification (the exhaustive application of **Purify**) on a unification problem Γ will halt in a PURE SUM Γ', which is a conservation extension of Γ.

□

Without difficulty, we have the following lemma:

Lemma 3.2 *Let* Γ *and* Γ' *are two unification problems.* Γ' *is the result of applying* ***Purify*** *on* Γ*. Then* Γ' *is Conservative Extension of* Γ*.*

Proof. It is enough to prove (i) if we have $\sigma \vDash \mathbf{S} \oplus t[s] \stackrel{?}{=} 0$, we can extend σ to σ' such that $\sigma' \vDash \{\mathbf{S} \oplus t[x] \stackrel{?}{=} 0, x \oplus s \stackrel{?}{=} 0\}$ and (ii) if we have $\theta \vDash \{\mathbf{S} \oplus t[x] \stackrel{?}{=} 0, x \oplus s \stackrel{?}{=} 0\}$, then $\theta \vDash \oplus t[s] \stackrel{?}{=} 0$.

For (i), we only need to add extend σ by adding $x \leftarrow s$.

For (ii), it is easy to prove. □

Example 3.3 The equation

$$f(h(x)) \oplus h(a \oplus b) \oplus g(h(y \oplus a), f(b)) \oplus h(f(x)) \stackrel{?}{=} 0$$

is not in pure. From our purification rule, firstly, we use v_1 to replace $h(x)$ to get

$$\begin{aligned}\{&f(v_1) \oplus h(a \oplus b) \oplus g(h(y \oplus a), f(b)) \oplus h(f(x)) \overset{?}{=} 0,\\ &v_1 \oplus h(x) \overset{?}{=} 0\}.\end{aligned}$$

Next we use v_2 to replace $a \oplus b$ and get

$$\begin{aligned}\{&f(v_1) \oplus h(v_2) \oplus g(h(y \oplus a), f(b)) \oplus h(f(x)) \overset{?}{=} 0,\\ &v_1 \oplus h(x) \overset{?}{=} 0,\\ &v_2 \oplus a \oplus b \overset{?}{=} 0\}.\end{aligned}$$

Next we use v_3 to replace $h(y \oplus a)$ and get

$$\begin{aligned}\{&f(v_1) \oplus h(v_2) \oplus g(v_3, f(b)) \oplus h(f(x)) \overset{?}{=} 0,\\ &v_1 \oplus h(x) \overset{?}{=} 0,\\ &v_2 \oplus a \oplus b \overset{?}{=} 0,\\ &v_3 \oplus h(y \oplus a) \overset{?}{=} 0\}.\end{aligned}$$

Because $h(y \oplus a)$ still has $\oplus$ in it, we need another fresh variable v_4 to replace it to get the equation set in pure:

$$\begin{aligned}\{&f(v_1) \oplus h(v_2) \oplus g(v_3, f(b)) \oplus h(f(x)) \overset{?}{=} 0,\\ &v_1 \oplus h(x) \overset{?}{=} 0,\\ &v_2 \oplus a \oplus b \overset{?}{=} 0,\\ &v_3 \oplus h(v_4) \overset{?}{=} 0,\\ &v_4 \oplus y \oplus a \overset{?}{=} 0\}.\end{aligned}$$

From now on, we will assume that every equation has the form $\mathbf{S} \overset{?}{=} 0$, 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, x_i, y, y_i, v, v_i, etc. to denote variables, σ, θ to denote substitutions.

3.5 Inference System $\mathfrak{I}_{ACUNh}$

3.5.1 Problem Format

For efficiency and convenience, in our inference procedure, we will use a triple $\Gamma \| \Delta \| \Lambda$, where $\|$ is used to separate these three sets. In $\Gamma \| \Delta \| \Lambda$, Γ is a unification problem, a set of the form $\{S_1 \overset{?}{=} 0, S_2 \overset{?}{=} 0, \cdots, S_n \overset{?}{=} 0\}$, where each S_i is a PURE SUM. Δ is a set of disequations, and Λ is a set of equations. Every disequation in Δ has the form $f(s_1, \cdots, s_n) \oplus f(t_1, \cdots, t_n) \neq^? 0$ or $0 \neq^? 0$, where f is an uninterpreted symbol from Γ and s_i, t_i are terms. Δ is used to track non-deterministic choices in our inference system. Every time we make a choice, we will add a disequation into Δ. Some equation will be added to Λ if a variable in Γ was solved during the inference procedure. All the equations in Λ will have the form $x \overset{?}{=} S$ where x is a *solved variable*, which means that x does not occur in Γ. Δ and Λ are both empty initially.

For convenience, we call Γ an *equation set*, Δ a *disequation set*, Λ a *solved equation set* and $\Gamma \| \Delta \| \Lambda$ a set triple. We say a substitution θ satisfies the set triple $\Gamma \| \Delta \| \Lambda$, if θ satisfies every equation in Γ and Λ, and every disequation in Δ, and write that relation as $\theta \vDash \Gamma \| \Delta \| \Lambda$. 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 triple with no solution.

3.5.2 Necessary Rules

$\mathfrak{I}_{ACUNh}$ contains four *necessary rules*: Trivial, Variable Substitution, N-Decomposition and Annulization; and six *auxiliary rules* used for efficiency. Trivial, Variable Substitution, Annulization and the auxiliary rules are deterministic, and N-Decomposition is nondeterministic. In our inference procedure, there are three 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 lowest priority is Annulization. Rules can only be applied if no rules with higher priority can be applied.

If no rules can be applied and $\Gamma = \emptyset$, then Λ is a solution. Exhaustively applying the inference rules to $\Gamma||\emptyset||\emptyset$ yields a complete set of **ACUNh**-unifiers of Γ.

In this section, we introduce the necessary rules.

Trivial

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

Trivial

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

Variable Substitution

The second necessary rule is used to solve variables. For example, if we have $x \oplus f(y) \overset{?}{=} 0$, we can get $x \leftarrow 0$. It is based on the property of $x \oplus x \approx 0$. It also has the highest priority.

Variable Substitution

$$\frac{(\Gamma \cup \{x \oplus \mathbf{S} \stackrel{?}{=} 0\})\|\Delta\|\Lambda}{(\Gamma\sigma)\|(\Delta\sigma)\|\Lambda\sigma \cup \{x \stackrel{?}{=} \mathbf{S}\}}$$

where $\sigma = [x \leftarrow \mathbf{S}]$. Here we need the conditions: x is unconstrained in $\Gamma \cup x \oplus \mathbf{S} \stackrel{?}{=} 0$ and either

- S has no h-term, or
- all equations of the form $x \oplus T \stackrel{?}{=} 0$ in Γ contain an h-term.

Note:

- If $\mathbf{S}$ is empty, then we set $\sigma = [x \leftarrow 0]$.
- If the resulting equation set is not pure[1], Purification is immediately applied to the conclusion of the inference.

Example 3.4 Find the solution of $\{x \oplus f(z) \stackrel{?}{=} 0, x \oplus y \oplus z \stackrel{?}{=} 0\}$.

For convenience, we omit Δ and Λ here. Here, x and y are unconstrained and z is not unconstrained. If we solve x first, we get $[x \leftarrow f(z)]$ and $\{f(z) \oplus y \oplus z \stackrel{?}{=} 0\}$. At this time, y is the only unconstrained variable, so we solve y and get the answer $[x \leftarrow f(z), y \leftarrow z \oplus f(z)]$. If we had chosen y first, we would get the same result.

If we fail to check the first condition of the inference rule, the procedure may not terminate. We give an example to explain this. For convenience, we omit Δ and Λ here.

[1]This can only happen by application of the rewrite rule $h(x) \oplus h(y) \to h(x \oplus y)$

Example 3.5 Consider the unification problem

$$\begin{aligned}\Gamma = \{&x \oplus h(v_1) \stackrel{?}{=} 0, v_1 \oplus y \oplus z \stackrel{?}{=} 0,\\ &y \oplus h(v_2) \stackrel{?}{=} 0, v_2 \oplus x \oplus z \stackrel{?}{=} 0,\\ &z \oplus h(v_3) \stackrel{?}{=} 0, v_3 \oplus x \oplus y \stackrel{?}{=} 0\}.\end{aligned}$$

We see x is unconstrained and from the condition, we can solve x from $v_2 \oplus x \oplus z \stackrel{?}{=} 0$ or $v_3 \oplus x \oplus y \stackrel{?}{=} 0$. Suppose we choose the second one. Then we have $x \leftarrow v_3 \oplus y$ and the equation set becomes:

$$\begin{aligned}\Gamma = \{&v_3 \oplus y \oplus h(v_1) \stackrel{?}{=} 0, v_1 \oplus y \oplus z \stackrel{?}{=} 0,\\ &y \oplus h(v_2) \stackrel{?}{=} 0, v_2 \oplus v_3 \oplus y \oplus z \stackrel{?}{=} 0,\\ &z \oplus h(v_3) \stackrel{?}{=} 0\}.\end{aligned}$$

Then we solve z from $v_1 \oplus y \oplus z \stackrel{?}{=} 0$ and get $z \leftarrow v_1 \oplus z$. The equation set becomes:

$$\begin{aligned}\Gamma = \{&v_3 \oplus y \oplus h(v_1) \stackrel{?}{=} 0, y \oplus h(v_2) \stackrel{?}{=} 0,\\ &v_2 \oplus v_3 \oplus v_1 \stackrel{?}{=} 0, y \oplus v_1 \oplus h(v_3) \stackrel{?}{=} 0\}.\end{aligned}$$

From the condition of Variable Substitution, every equation containing y contains an h-term, we can choose any one of them. Suppose we choose $y \oplus h(v_2) \stackrel{?}{=} 0$ and get $y \leftarrow h(v_2)$. The equation set becomes:

$$\begin{aligned}\Gamma = \{&v_3 \oplus h(v_2) \oplus h(v_1) \stackrel{?}{=} 0, v_2 \oplus v_3 \oplus v_1 \stackrel{?}{=} 0,\\ &h(v_2) \oplus v_1 \oplus h(v_3) \stackrel{?}{=} 0\}.\end{aligned}$$

This equation set is not pure anymore. We rewrite the equations by $\mathfrak{R}_{ACUNh}$ and purify them:

$$\begin{aligned}\Gamma = \{&v_3 \oplus h(v_5) \stackrel{?}{=} 0, v_5 \oplus v_2 \oplus v_1 \stackrel{?}{=} 0,\\ &v_2 \oplus v_3 \oplus v_1 \stackrel{?}{=} 0, h(v_6) \oplus v_1 \stackrel{?}{=} 0,\\ &v_6 \oplus v_2 \oplus v_3 \stackrel{?}{=} 0\}.\end{aligned}$$

Continue this procedure, we will finally get

$$\Gamma = \{v_4 \oplus h(v_4) \stackrel{?}{=} 0, v_4 \oplus v_5 \oplus hv_5 \stackrel{?}{=} 0\}.$$

Because there is no unconstrained variables, we can do nothing so far.

However, if we use Variable Substitution without the condition, we may choose $x \oplus h(v_1) \stackrel{?}{=} 0$ which gives $x \leftarrow h(v_1)$, the problem becomes:

$$\begin{aligned}\Gamma = \{&v_1 \oplus y \oplus z \stackrel{?}{=} 0,\\ &y \oplus h(v_2) \stackrel{?}{=} 0, v_2 \oplus h(v_1) \oplus z \stackrel{?}{=} 0,\\ &z \oplus h(v_3) \stackrel{?}{=} 0, v_3 \oplus h(v_1) \oplus y \stackrel{?}{=} 0\}.\end{aligned}$$

Then we choose $y \oplus h(v_2) \stackrel{?}{=} 0$ giving $y \leftarrow h(v_2)$, the problem becomes:

$$\begin{aligned}\Gamma = \{&v_1 \oplus h(v_2) \oplus z \stackrel{?}{=} 0, v_2 \oplus h(v_1) \oplus z \stackrel{?}{=} 0,\\ &z \oplus h(v_3) \stackrel{?}{=} 0, v_3 \oplus h(v_1) \oplus h(v_2) \stackrel{?}{=} 0\}.\end{aligned}$$

Then we reduce and purify $v_3 \oplus h(v_1) \oplus h(v_2) \stackrel{?}{=} 0$ by introducing new variable v_4 and get

$$\begin{aligned}\Gamma = \{&v_1 \oplus h(v_2) \oplus z \stackrel{?}{=} 0, v_2 \oplus h(v_1) \oplus z \stackrel{?}{=} 0,\\ &z \oplus h(v_3) \stackrel{?}{=} 0, v_3 \oplus h(v_4) \stackrel{?}{=} 0,\\ &v_4 \oplus v_1 \oplus v_2 \stackrel{?}{=} 0\}.\end{aligned}$$

Next we choose $z \oplus h(v_3) \overset{?}{=} 0$ to get $z \leftarrow h(v_3)$ and the problem becomes:

$$\Gamma = \{v_1 \oplus h(v_2) \oplus h(v_3) \overset{?}{=} 0, v_2 \oplus h(v_1) \oplus h(v_3) \overset{?}{=} 0,$$
$$v_3 \oplus h(v_4) \overset{?}{=} 0, v_4 \oplus v_1 \oplus v_2 \overset{?}{=} 0\}.$$

Then we reduce and purify $v_1 \oplus h(v_2) \oplus h(v_3) \overset{?}{=} 0$ and $v_2 \oplus h(v_1) \oplus h(v_3) \overset{?}{=} 0$ by introducing the variables v_5, v_6, we get:

$$\Gamma = \{v_1 \oplus h(v_5) \overset{?}{=} 0, v_2 \oplus v_3 \oplus v_5 \overset{?}{=} 0,$$
$$v_2 \oplus h(v_6) \overset{?}{=} 0, v_6 \oplus v_1 \oplus v_3 \overset{?}{=} 0,$$
$$v_3 \oplus h(v_4) \overset{?}{=} 0, v_4 \oplus v_1 \oplus v_2 \overset{?}{=} 0\}.$$

This problem is a renaming of the original problem, which means the procedure will not terminate.

So from this example we see that the first condition of Variable Substitution is important. In the next section, we will prove under the condition above, the procedure is terminating. We must note here is, in using this rule, we may generate an equation which is not pure, and in this case we will apply the purification procedure eagerly. So the result will still be pure.

The above two inference rules have the highest priority.

N-Decomposition

The next inference rule N-Decomposition is nondeterministic. This is to solve a case, like $f(x) \oplus f(y) \oplus f(z) \oplus f(a) \overset{?}{=} 0$. In this case there are several possible guesses: $\{f(x) = f(y) = f(a) \neq f(z)\}$, $\{f(z) = f(y) = f(a) \neq f(x)\}$, $\{f(x) = f(z) = f(a) \neq f(y)\}$, $\{f(z) \neq f(x) = f(a), f(x) \neq f(y), f(y) \neq f(x)\}$, and $\{f(x) = f(y) = f(z) = f(a)\}$ etc.

Since our problems is in PURE SUM form, these guesses are several syntactical unification problem 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

If $f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \neq^? 0 \notin \Delta$,

$$\frac{\Gamma \cup \{\mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \stackrel{?}{=} 0\} \| \Delta \| \Lambda}{(\Gamma\sigma \cup \{\mathbf{S} \stackrel{?}{=} 0\}\sigma) \| (\Delta\sigma) \| (\Lambda\sigma \cup [\sigma]) \qquad \bigvee \qquad \Gamma_1' \| \Delta_1' \| \Lambda}.$$

where

- $\sigma = mgu(s_1 \stackrel{?}{=} t_1, s_2 \stackrel{?}{=} t_2, \cdots, s_m \stackrel{?}{=} t_m)$
- $\Gamma_1' = \Gamma \cup \{\mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \stackrel{?}{=} 0\}$. (iii) $\Delta_1' = \Delta \cup \{f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \neq^? 0\}$

Example 3.6

$$\Gamma = \{x \oplus f(y) \oplus f(x_1) \stackrel{?}{=} 0, y \oplus f(z) \oplus f(x_2) \stackrel{?}{=} 0, z \oplus f(x) \oplus f(x_3) \stackrel{?}{=} 0\}$$

We can do nothing via Variable Substitution, so choose two of the pure terms in one equation to apply the N-Decomposition rule.

$\{x \oplus f(y) \oplus f(x_1) \overset{?}{=} 0, y \oplus f(z) \oplus f(x_2) \overset{?}{=} 0, z \oplus f(x) \oplus f(x_3) \overset{?}{=} 0\} \| \emptyset \| \emptyset$

$\Longrightarrow$ (N-Decomposition)

$\{x \overset{?}{=} 0, x_1 \oplus f(z) \oplus f(x_2) \overset{?}{=} 0, z \oplus f(x) \oplus f(x_3) \overset{?}{=} 0\} \| \emptyset \| \{y \overset{?}{=} x_1\}$

$\bigvee \Gamma_1 \| \Delta_1 \| \emptyset$

$(\Gamma_1 = \{x \oplus f(y) \oplus f(x_1) \overset{?}{=} 0, y \oplus f(z) \oplus f(x_2) \overset{?}{=} 0, z \oplus f(x) \oplus f(x_3) \overset{?}{=} 0\}$

$\Delta_1 = \{f(y) \oplus f(x_1) \neq^? 0\})$

$\Longrightarrow$ (Variable Substitution)

$\{x_1 \oplus f(z) \oplus f(x_2) \overset{?}{=} 0, z \oplus f(0) \oplus f(x_3) \overset{?}{=} 0\} \| \emptyset \| \{y \overset{?}{=} x_1, x \overset{?}{=} 0\}$

$\bigvee \Gamma_1 \| \Delta_1 \| \emptyset$

$\Longrightarrow$ (Variable Substitution)

$\{z \oplus f(0) \oplus f(x_3) \overset{?}{=} 0\} \| \emptyset \| \{y \overset{?}{=} f(z) \oplus f(x_2), x \overset{?}{=} 0, x_1 \overset{?}{=} f(z) \oplus f(x_2)\}$

$\bigvee \Gamma_1 \| \Delta_1 \| \emptyset$

$\Longrightarrow$ (Variable substitution)

$\emptyset \| \emptyset \| \{y \overset{?}{=} f(f(0) \oplus f(x_3)) \oplus f(x_2), x \overset{?}{=} 0,$

$x_1 \overset{?}{=} f(f(0) \oplus f(x_3)) \oplus f(x_2), z \overset{?}{=} f(0) \oplus f(x_3)\} \bigvee \Gamma_1 \| \Delta_1 \| \emptyset$

Here we get one solution $\sigma_1 = [y \leftarrow f(f(0) \oplus f(x_3)) \oplus f(x_2), x \leftarrow 0, x_1 \leftarrow f(f(0) \oplus f(x_3)) \oplus f(x_2), z \leftarrow f(0) \oplus f(x_3)]$ and another equation set Γ_1 and disequations set Δ_1. So for $\Gamma_1 \| \Delta_1$, we have

$\{x \oplus f(y) \oplus f(x_1) \overset{?}{=} 0,\ y \oplus f(z) \oplus f(x_2) \overset{?}{=} 0,\ z \oplus f(x) \oplus f(x_3) \overset{?}{=} 0\} \| \{y \oplus x_1 \neq^? 0\} \| \emptyset$

$\Longrightarrow$ (N-Decomposition)

$\{y \overset{?}{=} 0,\ x \oplus f(y) \oplus f(x_1) \overset{?}{=} 0,\ x_2 \oplus f(x) \oplus f(x_3) \overset{?}{=} 0\} \| \Delta_1 \| \{z \overset{?}{=} x_2\}$

$\bigvee \Gamma_2 \| \Delta_2 \| \emptyset$

$(\Gamma_2 = \{x \oplus f(y) \oplus f(x_1) \overset{?}{=} 0, y \oplus f(z) \oplus f(x_2) \overset{?}{=} 0, z \oplus f(x) \oplus f(x_3) \overset{?}{=} 0\}$

$\Delta_2 = \{y \oplus x_1 \neq^? 0, z \oplus x_2 \neq^? 0\})$

$\Longrightarrow$ (Using a similar procedure)

$\ldots$

$\Longrightarrow$

$\emptyset \| \Delta_1' \| \Lambda' \bigvee \Gamma_2 \| \Delta_2 \| \emptyset$

So, we get a second solution $\sigma_2 = [z \leftarrow x_2, y \leftarrow 0, x_2 \leftarrow f(x) \oplus f(x_3), x \leftarrow f(0) \oplus f(x_1)]$ (We will prove that the Δ can be thrown away at the end) and another set triple: $\Gamma_2 \| \Delta_2 \| \Lambda_2$. Here Δ_1 becomes $\Delta_1' = \{x_1 \neq^? 0\}$.

Similarly, we can get a third solution $\sigma_3 = [x \leftarrow x_3, z \leftarrow 0, x_3 \leftarrow f(y) \oplus f(x_1), y \leftarrow f(0) \oplus f(x_2)]$ and another set triple of equations and disequations:

$$\begin{aligned}&\{x \oplus f(y) \oplus f(x_1) \overset{?}{=} 0, y \oplus f(z) \oplus f(x_2) \overset{?}{=} 0, z \oplus f(x) \oplus f(x_3) \overset{?}{=} 0\}\\ &|| \{y \oplus x_1 \neq^? 0, z \oplus x_2 \neq^? 0, x \oplus x_3 \neq^? 0\}\\ &|| \emptyset\end{aligned}$$

Because we can do nothing about this equation set, we fail for the last equation set.

So the complete set of unifiers is:

$$\begin{aligned}
\{\sigma_1 &= [y \leftarrow f(f(0) \oplus f(x_3)) \oplus f(x_2), x \leftarrow 0, z \leftarrow f(0) \oplus f(x_3)],\\
\sigma_2 &= [z \leftarrow f(f(0) \oplus f(x_1)) \oplus f(x_3), y \leftarrow 0, x \leftarrow f(0) \oplus f(x_1)],\\
\sigma_3 &= [x \leftarrow f(f(0) \oplus f(x_2)) \oplus f(x_1), z \leftarrow 0, y \leftarrow f(0) \oplus f(x_2)]\}
\end{aligned}$$

In the following tree, we give an overview to see how this nondeterministic rule is applied.

$$\begin{tikzcd}
& f(y) \oplus f(x_1) \overset{?}{=} 0 \arrow[dl, dash, "\text{YES}"'] \arrow[dr, dash, "\text{NO}"] & & & \\
y = x_1 \Rightarrow \text{Get solution } \sigma_1 & & f(z) \oplus f(x_2) \overset{?}{=} 0 \arrow[dl, dash, "\text{YES}"'] \arrow[dr, dash, "\text{NO}"] & & \\
& z = x_2 \Rightarrow \text{Get solution } \sigma_2 & & f(x) \oplus f(x_3) \overset{?}{=} 0 \arrow[dl, dash, "\text{YES}"'] \arrow[dr, dash, "\text{NO}"] & \\
& & x = x_3 \Rightarrow \text{Get solution } \sigma_3 & & \text{No Rules Applied } \textbf{Fail}.
\end{tikzcd}$$

Annulization

The fourth rule is based on the homomorphism property $h(0) =_{ACUNh} 0$ and it has lowest priority. It will solve the problem like $h(x) \oplus x \overset{?}{=} 0$.

Annulization

$$\frac{(\Gamma \cup \{\mathbf{S} \oplus h(t) \overset{?}{=} 0\}) \| \Delta \| \Lambda}{(\Gamma \cup \{\mathbf{S} \overset{?}{=} 0\} \cup \{t \overset{?}{=} 0\}) \| \Delta \| \Lambda}$$

if there are no uninterpreted function symbol on the top in Γ and $\mathbf{S}^2$, Here $\mathbf{S}$ may be empty.

Note:

- The process will fail eventually if there is an uninterpreted symbol on the top of Γ or $\mathbf{S}$, because no inference rules will be applicable.
- This rule can continued being applied until no h-terms left.

Now we give an example of how to apply Annulization. For convenience we will ignore Δ and Λ.

Example 3.7

$$\Gamma = \{x \oplus h(y) \oplus h(z) =^? 0, y \oplus h(x) \oplus h(z) \overset{?}{=} 0, z \oplus h(x) \oplus h(y) \overset{?}{=} 0\}$$

These equations are not reduced yet. So we need to reduce via $\mathfrak{R}_{ACUNh}$, then we can get

$$\Gamma = \{x \oplus h(y \oplus z) \overset{?}{=} 0, y \oplus h(x \oplus z) \overset{?}{=}, z \oplus h(x \oplus y) \overset{?}{=} 0\}$$

We can see these equations are not pure according to our definition of pure, we need to purify them first and get

$$\begin{aligned}\Gamma = \{&x \oplus h(v_1) \overset{?}{=} 0, v_1 \oplus y \oplus z \overset{?}{=} 0,\\ &y \oplus h(v_2) \overset{?}{=} 0, v_2 \oplus x \oplus z \overset{?}{=} 0,\\ &z \oplus h(v_3) \overset{?}{=} 0, v_3 \oplus x \oplus y \overset{?}{=} 0\}.\end{aligned}$$

Next, we can use our inference rules to solve Γ. First, we find x is unconstrained, but according to Variable Substitution, we cannot chose $x \oplus h(v_1) \overset{?}{=} 0$ to solve x because $v_2 \oplus x \oplus z \overset{?}{=} 0$ has no h in it. So we choose $v_2 \oplus x \oplus z \overset{?}{=} 0$ and get $x \leftarrow v_2 \oplus z$, Γ becomes

$$\begin{aligned}\Gamma = \{&v_2 \oplus z \oplus h(v_1) \overset{?}{=} 0, v_1 \oplus y \oplus z \overset{?}{=} 0,\\ &y \oplus h(v_2) \overset{?}{=} 0, z \oplus h(v_3) \overset{?}{=} 0, v_3 \oplus v_2 \oplus z \oplus y \overset{?}{=} 0\}.\end{aligned}$$

Then solve z via $v_1 \oplus y \oplus z \stackrel{?}{=} 0$:

$$\Gamma = \{v_2 \oplus z \oplus h(v_1) \stackrel{?}{=} 0, v_1 \oplus z \oplus h(v_2) \stackrel{?}{=} 0,$$
$$z \oplus h(v_3) \stackrel{?}{=} 0, v_3 \oplus v_2 \oplus v_1 \stackrel{?}{=} 0\}.$$

Here, we can find the only unconstrained variable is z, and every equation which contains s has an h-term in it. So we can choose any of these equations to solve z. Here we choose $z \oplus h(v_3) \stackrel{?}{=} 0$ and Γ becomes:

$$\Gamma = \{v_2 \oplus h(v_3) \oplus h(v_1) \stackrel{?}{=} 0, v_1 \oplus h(v_2) \oplus h(v_3) \stackrel{?}{=} 0, v_3 \oplus v_2 \oplus v_1 \stackrel{?}{=} 0\}.$$

So we need to reduce Γ and purify it again:

$$\begin{aligned}\Gamma &= \{v_2 \oplus h(v_3 \oplus v_1) \stackrel{?}{=} 0, v_1 \oplus h(v_2 \oplus v_3) \stackrel{?}{=} 0, v_3 \oplus v_2 \oplus v_1 \stackrel{?}{=} 0\} \\ &= \{v_2 \oplus h(v_4) \stackrel{?}{=} 0, v_4 \oplus v_1 \oplus v_1 \stackrel{?}{=} 0, \\ &\quad v_1 \oplus h(v_5) \stackrel{?}{=} 0, v_5 \oplus v_2 \oplus v_3 \stackrel{?}{=} 0, v_3 \oplus v_2 \oplus v_1 \stackrel{?}{=} 0\}.\end{aligned}$$

Then use a similar procedure to solve v_1 via $v_3 \oplus v_2 \oplus v_1 \stackrel{?}{=} 0$:

$$\Gamma = \{v_2 \oplus h(v_4) \stackrel{?}{=} 0, v_4 \oplus v_2 \stackrel{?}{=} 0,$$
$$v_2 \oplus v_3 \oplus h(v_5) \stackrel{?}{=} 0, v_5 \oplus v_3 \oplus v_2 \stackrel{?}{=} 0\}.$$

Solve v_2 via $v_4 \oplus v_2 \stackrel{?}{=} 0$:

$$\Gamma = \{v_4 \oplus h(v_4) \stackrel{?}{=} 0, v_4 \oplus v_3 \oplus h(v_5) \stackrel{?}{=} 0, v_5 \oplus v_3 \oplus v_4 \stackrel{?}{=} 0\}.$$

Solve v_3 via $v_5 \oplus v_3 \oplus v_4 \stackrel{?}{=} 0$:

$$\Gamma = \{v_4 \oplus h(v_4) \stackrel{?}{=} 0, v_5 \oplus h(v_5) \stackrel{?}{=} 0\}.$$

Now, no rules can be applied except Annulization. So we use Annulization to let Γ be $\{v_4 \overset{?}{=} 0, v_5 \oplus h(v_5) \overset{?}{=} 0\}$ We can solve $v_4 = 0$. Then no rules can be applied except Annulization again, so we can get $\{v_5 \overset{?}{=} 0\}$ and easily get the solution:

$$[x \leftarrow 0, y \leftarrow 0, z \leftarrow 0].$$

3.6 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. First we define directed conservative extension:

Definition 3.8 (Directed Conservative Extension) *Let* $\Gamma \| \Delta \| \Lambda$ *and* $\Gamma' \| \Delta' \| \Lambda'$ *be two set triples.* $\Gamma' \| \Delta' \| \Lambda'$ *is called a* ***directed conservative extension*** *of* $\Gamma \| \Delta \| \Lambda$*, if for any substitution* θ*, such that* $\theta \vDash \Gamma \| \Delta \| \Lambda$*, then there exists* σ*, whose domain is the variables in* $Vars(\Gamma' \cup \Lambda')/Vars(\Gamma \cup \Lambda)$*, such that* $\theta\sigma \vDash \Gamma' \| \Delta' \| \Lambda'$*.*

From the definition, we can see this is the one direction of the conservative extension, but applying on set triple but not only Γ. We will used it to prove the inference rules never lose any solutions.

Next, we give a definition about an abstract inference rule called *Simplifier*, which covers all our concrete auxiliary rules, given later.

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

Definition 3.9 (Preservative rules and Simplifiers) *Let* E *be an equational theory. Let* I

be an inference rule of the following form:

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

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

3.7 Termination

Next, we give four well-founded orderings for proving termination:

Let Γ be a set of equations then $Vars(\Gamma) = \{x \mid x \text{ occurs in some equation in } \Gamma\}$, and $|Vars(\Gamma)|$ 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 sets.

From the definition of Δ, we see that Δ only contains disequations of the form $s \oplus t \neq^? 0$ where s and t have the same top function symbol. So we let $Par(\Delta)$ be $\{(t, s) \mid t \oplus s \neq^? 0 \in \Delta\}$ and $Par(\Gamma)$ be $\{(t, s) \mid t, s \in \Gamma \text{ and } t \text{ has the same top function symbol as } s\}$.

We use $Par(\Gamma/\Delta)$ to denote $Par(\Gamma) - Par(\Delta)$. Since the number of terms in Γ is finite, $|Par(\Gamma/\Delta)|$ is a well-founded ordering. This ordering is used to count all possible disequations that could be placed in Δ by N-Decomposition, but not including the ones that are already there.

We also need a well-founded ordering $H(\Gamma)$, which is the number of h-terms in Γ.

We then define the measure of $\Gamma\|\Delta\|\Lambda$ as the lexicographically ordered quadruple $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda) = (H(\Gamma), |Vars(\Gamma)|, |Sym(E)|, |Par(\Gamma/\Delta)|)$. Obviously, this measure is well founded because the four arguments are well-founded.

For two set triples, $\Gamma\|\Delta\|\Lambda$ and $\Gamma'\|\Delta'\|\Lambda'$, we use $\Gamma\|\Delta\|\Lambda \Rightarrow_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$ to mean we can obtain $\Gamma'\|\Delta'\|\Lambda'$ from $\Gamma\|\Delta\|\Lambda$ by applying a rule with the following form from $\mathfrak{I}_{ACUNh}$ once. Note that for N-Decomposition, $\Gamma'\|\Delta'\|\Lambda'$ could represent either of the choices. Let $\Gamma\|\Delta\|\Lambda \overset{*}{\Rightarrow}_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$ mean that $\Gamma'\|\Delta'\|\Lambda'$ can be obtained by applying zero or more rules from $\mathfrak{I}_{ACUNh}$.

Lemma 3.10 *If* $\Gamma\|\Delta\|\Lambda \overset{*}{\Rightarrow}_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$ *and* Γ *is in* PURE SUM *form, then the equations in* Γ' *are also in* PURE SUM *form.*

Proof. For Variable Substitution, as we stated after the rule, we will apply the purification procedure if the result is not a PURE SUM. So the result will be a PURE SUM. For other rules, it is trivially true. □

Hence, we can always assume that the equation set Γ is in PURE SUM form. So we claim here that $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda)$ is reduced by every inference rule.

Lemma 3.11 *Let* $\Gamma\|\Delta\|\Lambda$ *and* $\Gamma'\|\Delta'\|\Lambda'$ *be two triple sets, such that* $\Gamma\|\Delta\|\Lambda \Rightarrow_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$*, Then,* $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda) > \mathbb{M}_{ACUNh}(\Gamma', \Delta', \Lambda')$

Proof. **Trivial:** $|Sym(E)|$ decreases and the others do not change. So $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda)$ decreases.

Variable Substitution: For the substitution $\sigma = [x \leftarrow \mathbf{S}]$, if $\mathbf{S}$ does not contain an h-term, then after we replace all the positions where the variable x occurs by $\mathbf{S}$, we will not introduce any variables, so $|Vars(\Gamma)| = |Vars(\Gamma')| + 1$. Hence $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda) > \mathbb{M}_{ACUNh}(\Gamma', \Delta', \Lambda')$.

If $\mathbf{S} = \mathbf{S}' \oplus h(t)$, from the condition of Variable Substitution, we know every equation which contains x, has an h-term in it. This means all the equations $x \oplus h(t') \oplus \mathbf{S}''$ will become $\mathbf{S}' \oplus h(t) \oplus h(t') \oplus \mathbf{S}''$. In this case, our rewriting rules and purification rule will be applied immediately. i.e. $\mathbf{S}' \oplus h(t) \oplus h(t') \oplus \mathbf{S}'' \overset{?}{=} 0 \Rightarrow h(t \oplus t') \oplus \mathbf{S}' \oplus \mathbf{S}'' \overset{?}{=} 0 \Rightarrow \{h(x') \oplus \mathbf{S}' \oplus \mathbf{S}'', x' \oplus t \oplus t' \overset{?}{=} 0\}$, where x' is a fresh variable. From this procedure, we see that $H(\Gamma) = H(\Gamma') + 1$. Hence $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda) > \mathbb{M}_{ACUNh}(\Gamma', \Delta', \Lambda')$.

N-Decomposition: For Choice 1, in the decomposition rules, σ is $mgu(s_1 \overset{?}{=} t_1, s_2 \overset{?}{=} t_2, \cdots, s_n \overset{?}{=} t_n)$, which will not introduce new variables. If some variables were solved, and these variables make some h-terms zeros, then $H(\Gamma)$ decrease. If no variable makes h-terms zeros, then $H(\Gamma) = H(\Gamma')$ 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 $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda) > \mathbb{M}_{ACUNh}(\Gamma', \Delta', \Lambda')$.

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 $|Par(\Gamma) - Par(\Delta)| = |Par(\Gamma') - Par(\Delta')| + 1$. Hence $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda) > \mathbb{M}_{ACUNh}(\Gamma', \Delta', \Lambda')$ decreases.

Annulization: $H(\Gamma)$ decreases, so $\mathbb{M}_{ACUNh}(\Gamma, \Delta, \Lambda)$ decreases.

Simplifiers: it is trivially true from the definition.:

□

Theorem 3.12 *For any triple set* $\Gamma \| \Delta \| \Lambda$*, there is a triple set* $\Gamma' \| \Delta' \| \Lambda'$ *such that* $\Gamma \| \Delta \| \Lambda \overset{*}{\Rightarrow}_{\mathfrak{I}_{ACUNh}} \Gamma' \| \Delta' \| \Lambda'$ *and no rules in* $\mathfrak{I}_{ACUNh}$ *can be applied on* $\Gamma' \| \Delta' \| \Lambda'$*.*

Let us estimate how many steps we need to solve a unification problem in the worst case.

The purification procedure is linear with respect to the size of the unification problem.

For Variable Substitution, assume after purification from the original unification problem, we have m equations and n h-terms, where m is smaller than the size of the original unification problem. From the proof of Lemma 3.11, every time we apply Variable Substitution, some equation will be removed from Γ. Because N-Decomposition and Annulization will not add any new equations, the only possibility that new equations are added is from the purification after Variable Substitution. In this case, the newly added equations will not contain any h-terms and the number of new equations will be one less than the current number of h-terms. So in the worst case, we need to apply Variable Substitution at most $m + (n-1)(n-2)/2$ times, where $(n-1)(n-2)/2$ is the number of newly added equations.

For N-Decomposition, in the worst case we might need to compare all the possible uninterpreted function terms. Assume we have k different function terms. Because our rules never increase the size of the set of uninterpreted function terms, at most we need to compare $k(k-1)/2$ times, which means at most we need to apply N-Decomposition $k(k-1)/2$ times.

We apply Annulization at most n times, where n is the number of h-terms.

So the upper bound of inference steps we might apply is non-deterministically quadratic with respect to the size of the problem. Because applying every inference rule is in polynomial time, our algorithm is bounded by non-deterministic polynomial time.

3.8 Soundness

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

Lemma 3.13 *For two set triples $\Gamma\|\Delta\|\Lambda$, $\Gamma'\|\Delta'\|\Lambda'$ satisfying $\Gamma\|\Delta\|\Lambda \Rightarrow_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$, let θ be a substitution such that $\theta \vDash \Gamma'\|\Lambda'$. Then $\theta \vDash \Gamma\|\Lambda$.*

Proof. We prove it rule by rule.

Trivial. This is trivially true.

Variable Substitution:

$$\frac{(\Gamma \cup \{x \oplus \mathbf{S} \overset{?}{=} 0\})\|\Delta\|\Lambda}{(\Gamma\sigma)\|(\Delta\sigma)\|\Lambda\sigma \cup \{x \overset{?}{=} \mathbf{S}\}}$$

If $\theta \vDash (\Gamma\sigma)\|\Lambda\sigma \cup \{x \overset{?}{=} \mathbf{S}\}$, what we need to do is to prove $\theta \vDash (\Gamma \cup \{x \oplus \mathbf{S} \overset{?}{=} 0\})\|\Lambda$.

Because, we have $x\theta = \mathbf{S}\theta$, $\theta \vDash x \oplus \mathbf{S} \overset{?}{=} 0$. For an arbitrary equation $\mathbf{T} \overset{?}{=} 0$ in $\Gamma \cup \Lambda$, we have $\mathbf{T}\sigma\theta = 0$. Because $x\theta = \mathbf{S}\theta$ and $\sigma = [x \leftarrow \mathbf{S}]$, we have for variable x, $x\sigma\theta = S\theta = x\theta$, and for any variable $y \neq x$, $y\sigma\theta = y\theta$. So $\sigma\theta = \theta$, which means $\mathbf{T}\theta = 0$, i.e. θ satisfies $\mathbf{T} \overset{?}{=} 0$. Thus $\theta \vDash (\Gamma \cup \{x \oplus \mathbf{S} \overset{?}{=} 0\})\|\Lambda$.

N-Decomposition:

First Choice:

$$\frac{(\Gamma \cup \{\mathbf{S} \oplus f(s_1, s_2, \cdots s_m) \oplus f(t_1, t_2, \cdots t_m) \overset{?}{=} 0\})\|\Delta\|\Lambda}{(\Gamma'\sigma)\|(\Delta\sigma)\|(\Lambda\sigma \cup [\sigma])}$$

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

If $\theta \vDash \Gamma\sigma \cup \{\mathbf{S} \overset{?}{=} 0\}\sigma\|(\Lambda\sigma \cup [\sigma])$, what we need to do is prove $\theta \vDash (\Gamma \cup \{\mathbf{S} \oplus f(s_1, s_2, \cdots s_m) \oplus f(t_1, t_2, \cdots t_m) \overset{?}{=} 0\})\|\Lambda$.

Because $\theta \vDash [\sigma]$, we have $\sigma\theta = \theta$ and $s_i\theta = t_i\theta$, where $0 < i < m + 1$. Then $f(s_1\theta, s_2\theta, \cdots, s_m\theta) = f(t_1\theta, t_2\theta, \cdots, t_m\theta)$, which means $\theta \vDash f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \overset{?}{=} 0$. Because $\theta \vDash \mathbf{S}\sigma \overset{?}{=} 0$ and $\sigma\theta = \theta$, we get $\theta \vDash \mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \overset{?}{=} 0$. And because $\sigma\theta = \theta$ and $\theta \vDash \Gamma'\sigma\|(\Lambda\sigma \cup [\sigma])$, we have $\theta \vDash \Gamma\|\Lambda$, which means $\theta \vDash (\Gamma \cup \{\mathbf{S} \oplus f(s_1, s_2, \cdots s_m) \oplus f(t_1, t_2, \cdots t_m) \overset{?}{=} 0\})\|\Lambda$.

The second choice is obvious.

Annulization: We only need to show $\theta \vDash t \overset{?}{=} 0$ implies $\theta \vDash h(t) \overset{?}{=} 0$. This is true from our rewriting rules.

Simplifier: This is true from the definition of Simplifier.

□

Theorem 3.14 *For any two set triples* $\Gamma\|\Delta\|\Lambda$ *and* $\Gamma'\|\Delta'\|\Lambda'$*, satisfying* $\Gamma\|\Delta\|\Lambda \overset{*}{\Rightarrow}_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$*, if there is a solution* θ*, satisfying* $\theta \vDash \Gamma'\|\Lambda'$*, then* $\theta \vDash \Gamma\|\Lambda$*.*

From above theorem we have the following corollary:

Corollary 3.15 (Soundness) Let Γ be an ACUNh-unification problem. Suppose after applying the inferences rules from $\mathfrak{I}_{ACUNh}$ to $\Gamma\|\emptyset\|\emptyset$ exhaustively, we get $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i)$, i.e.

$$\Gamma\|\emptyset\|\emptyset \overset{*}{\Rightarrow}_{\mathfrak{I}_{ACUNh}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i)$$

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

3.9 Completeness

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

Lemma 3.16 *Let* $\Gamma\|\Delta\|\Lambda$ *be a set triple. If there exists another set triple* $\Gamma'\|\Delta'\|\Lambda'$*, such that* $\Gamma\|\Delta\|\Lambda \Rightarrow_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$ *via a deterministic rule except Annulization, then* $\Gamma'\|\Delta'\|\Lambda'$ *is a directed conservative extension of* $\Gamma\|\Delta\|\Lambda$*.*

Proof. There are three cases: Trivial, Simplifiers and Variable Substitution and this lemma is trivially true for the first two cases. Let use look at Variable Substitution.

For Variable Substitution, we might generate new variables from purification whenever we generate non pure term. From Lemma 3.2, we know the Γ' is the conservative extension of Γ. So it is enough to show that for any substitution θ, $\theta \vDash \Gamma'\|\Delta'\|\Lambda'$ whenever $\theta \vDash \Gamma\|\Delta\|\Lambda$.

Γ has an equation Q of the form $x \oplus \mathbf{S} \overset{?}{=} 0$, where x is unconstrained in Γ. Γ has a solution θ, such that $x\theta \oplus \mathbf{S}\theta = 0$, which implies $x\theta = \mathbf{S}\theta$. Because $\sigma = [x \leftarrow \mathbf{S}]$, $x\sigma\theta = \mathbf{S}\theta = x\theta$. For other variables y, from the definition of substitution, we have $y\sigma\theta = y\theta$. This means for every variable v in Γ, we have $v\sigma\theta = v\theta$. So $\theta = \sigma\theta$. $\Gamma\sigma\theta \| \Delta\sigma\theta \| \Lambda\sigma\theta \cup \{x\theta \overset{?}{=} \mathbf{S}\theta\} = \Gamma\theta\|\Delta\theta\|\Lambda\theta \cup \{x\theta \overset{?}{=} \mathbf{S}\theta\} = (\Gamma/\{x \oplus \mathbf{S} \overset{?}{=} 0\})\theta\|\Delta\theta\|\Lambda\theta \cup \{x \overset{?}{=} \mathbf{S}\}\theta$. So θ satisfies $\Gamma/\{x \oplus \mathbf{S} \overset{?}{=} 0\}\|\Delta\|\Lambda \cup \{x \overset{?}{=} \mathbf{S}\}$. □

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 3.17 *If t is reduced, then* $\mathbf{Lay}(t)$ *has the following value:*

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

Lemma 3.18 *Let $\Gamma\|\Delta\|\Lambda$ be a set triple, if there exists another set triple $\Gamma'\|\Delta'\|\Lambda'$, such that $\Gamma\|\Delta\|\Lambda \Rightarrow_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$ via Annulization, then $\Gamma'\|\Delta'\|\Lambda'$ is a directed conservative extension of $\Gamma\|\Delta\|\Lambda$.*

Proof. In the procedure of applying Annulization, we will not generate new variables, so it is enough to show for any substitution θ, $\theta \vDash \Gamma'\|\Delta'\|\Lambda'$ whenever $\theta \vDash \Gamma\|\Delta\|\Lambda$.

Since Annulization is applicable, every equation has the form

$$x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{in} \oplus h(t_i) \overset{?}{=} 0,$$

where all the x_{ij}'s are not unconstrained. Annulization sets all the h-terms to zero. From the rule, we know Annulization will not change Δ. So it is enough to show there is no solution θ, such that for some $h(t_i)$, $(h(t_i)\theta \downarrow) \neq 0$.

Suppose there is a ground reduced substitution θ and there exists an h-term $h(t_j)$, such that $(h(t_j)\theta) \downarrow\neq 0$. Suppose θ is $\{x_1 \leftarrow T_1, x_2 \leftarrow T_2, \cdots x_n \leftarrow T_n, y_1 \leftarrow S_1, y_2 \leftarrow S_2, \cdots, y_l \leftarrow S_l\}$, where each x_i is a non-unconstrained variable and y_i is a unconstrained variable in Γ. Without loss of generality, we suppose $Lay(T_1) \geq Lay(T_2) \geq \cdots \geq Lay(T_n)$.

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 $x_{j1} \oplus x_{j2} \oplus \cdots \oplus x_{jn} \oplus h(t_j) \overset{?}{=} 0$, where $(h(t_j)\theta) \downarrow\neq 0$, we have:

$$\begin{aligned} & x_{j1} \oplus x_{j2} \oplus \cdots \oplus x_{jn} \oplus h(t_j)\theta \\ = \; & a_{j1} \oplus a_{j2} \oplus \cdots \oplus a_{jn} + h(t_j)\theta, \end{aligned}$$

where a_{ji} are constants or variables. There are no way to cancel $h(t_j)\theta$ if $h(t)\theta \downarrow\neq 0$. This equation cannot be true.

So we can say $Lay(T_1)$ is not zero. Since x_1 occurs in t_i in the equation $x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{im} \oplus h(t_i) \overset{?}{=} 0$, where x_{ij}'s are not unconstrained variables, if we want to cancel $h(t_i)\theta$, we need another variable $x_{ij}\theta$ cancel it because there are no other h-terms in it. Suppose $x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{ik}$ can cancel $h(t_i)\theta$. Then we have $(x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{ik})\theta = h(t_i)\theta + T$. So $Lay((x_{i1} \oplus x_{i2} \oplus \cdots \oplus x_{ik})\theta) = \max\{Lay(t_{i1}), Lay(t_{i2}), \cdots, Lay(t_{ik})\} = Lay(h(t_i)) > Lay(x_1\theta) = Lay(T_1)$, which is a contradiction.

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

Lemma 3.19 *Let $\Gamma \| \Delta \| \Lambda$ be a set triple. If there exists another set triple $\Gamma' \| \Delta' \| \Lambda'$, such that $\Gamma \| \Delta \| \Lambda \Rightarrow_{\mathfrak{I}_{ACUNh}} \Gamma' \| \Delta' \| \Lambda'$ via N-Decomposition, then $\Gamma' \| \Delta' \| \Lambda'$ is a directed conser-*

vative extension of $\Gamma\|\Delta\|\Lambda$.

Proof. In the procedure of applying N-Decomposition, we will not generate new variables, so it is enough to show that if there exists another two set triples $\Gamma'\|\Delta'\|\Lambda'$ and $\Gamma''\|\Delta''\|\Lambda''$, such that $\Gamma\|\Delta\|\Lambda \Rightarrow_{\Im_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$ and $\Gamma\|\Delta\|\Lambda \Rightarrow_{\Im_{ACUNh}} \Gamma''\|\Delta''\|\Lambda''$ via N-Decomposition, then for any substitution θ, either $\theta \vDash \Gamma'\|\Delta'\|\Lambda'$ or $\theta \vDash \Gamma''\|\Delta''\|\Lambda''$, whenever $\theta \vDash \Gamma\|\Delta\|\Lambda$.

We have two cases if we can apply N-Decomposition.

Case 1: Γ has an equation of the form

$$\mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \stackrel{?}{=} 0,$$

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

Recall what σ is in N-Decomposition. Let $x\sigma = r$ where x is in the domain of σ. Then $x \stackrel{?}{=} r \in [\sigma]$. Because $\theta \vDash f(s_1, s_2, \cdots, s_m) \stackrel{?}{=} f(t_1, t_2, \cdots, t_m)$, we have $\theta \vDash [\sigma]$. So $x\sigma\theta = r\theta = x\theta$. For every variable y in Γ which is not in the domain of σ, $y\sigma\theta = y\theta$, which means $\theta \vDash (\Gamma\sigma \cup \{\mathbf{S} \stackrel{?}{=} 0\})\|\Delta\sigma\|(\Lambda\sigma \cup [\sigma])$.

Case 2: Γ has an equation of the form

$$\mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \stackrel{?}{=} 0$$

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

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

$$\theta \vDash \Gamma\|\Delta \cup \{f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \neq^? 0\}\|\Lambda.$$

$\square$

Lemma 3.20 *Let $\Gamma \| \Delta \| \Lambda$ be a set triple. If there is not another set triple $\Gamma' \| \Delta' \| \Lambda'$, such that $\Gamma \| \Delta \| \Lambda \Rightarrow_{\Im_{ACUNh}} \Gamma' \| \Delta' \| \Lambda'$, then $\Gamma \| \Delta \| \Lambda$ 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:

$$x_{i1} \oplus \cdots \oplus x_{ik} \oplus \mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \stackrel{?}{=} 0,$$

where each x_{ij} is a non-unconstrained variable in Γ, and $\mathbf{S}$ contains no pure variables.

Assume there are no rules to apply to $\Gamma \| \Delta \| \Lambda$ and θ is a ground reduced substitution, which is $\{x_1 \leftarrow T_1, x_2 \leftarrow T_2, \cdots x_n \leftarrow T_n, y_1 \leftarrow S_1, y_2 \leftarrow S_2, \cdots, y_l \leftarrow S_l\}$, where each x_i is a non-unconstrained variable and each y_i is a 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

$$x_{i1} \oplus \cdots \oplus x_{ik} \oplus \mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \stackrel{?}{=} 0$$

becomes

$$a_{i1} \oplus \cdots \oplus a_{ik} \oplus \mathbf{S}\theta \oplus f(s_1, s_2, \cdots, s_m)\theta = 0,$$

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 it. If there is some f-term, e.g. $f(t_1, t_2, \cdots, t_m)\theta$ in $\mathbf{S}\theta$ which can cancel it, then because here we have no rules to apply, $f(t_1, t_2, \cdots t_m) \oplus f(s_1, s_2, \cdots, s_m) \neq 0 \in \Delta$, which means $f(t_1, t_2, \cdots t_m)\theta \neq f(s_1, s_2, \cdots, s_m)\theta$. So this equation can not be zero.

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

Because no rules in $\mathfrak{I}_{ACUNh}$ can be applied and from Variable Substitution's conditions, there are no unconstrained variable in Γ. And since x_1 is not a 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 $\mathbf{S} \oplus t \overset{?}{=} 0$ 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.

Suppose this x is some x_i, then $x_i\theta = t[x_1]\theta + T$. Then $Lay(x_i\theta) = Lay(T_i) = Lay(t[x_1]\theta+T) \geq Lay(t[x_1]\theta) > Lay(x_1\theta) = Lay(T_1)$, which is a contradiction with $Lay(T_1)$ is the biggest during all the T_i's.

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 $\mathbf{S} \oplus h(x_1) =^? 0$. Because $Lay(T_1) \neq 0$, we can suppose $T_1 = t_1 \oplus T_2'$, where $Lay(T_1) = Lay(t_1) \geq Lay(T'2)$ and t_1 is not a sum. Because only one h-term is in this equation and $h(s) + h(t) = h(s+t)$, we need another variable x to has the contribution to cancel $h(t_1)$. we claim that no variable can cancel $h(t_1)$.

If some non-unconstrained variable x_i , can cancel $h(t_1)$, then $x_i\theta = h(t_1) + R$. So $Lay(T_i) = Lay(x_i\theta) = Lay(h(t_1) + R) \geq Lay(h(T_1)) > Lay(T_1)$ which is a contradiction.

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

$\square$

Then by combining Lemma 3.16, Lemma 3.18, Lemma 3.19 and Lemma 3.20 we get:

Lemma 3.21 *Let* $\Gamma \| \Delta \| \Lambda$ *be a set triple. If there exists another set triple* $\Gamma' \| \Delta' \| \Lambda'$, *such that* $\Gamma\|\Delta\|\Lambda \Rightarrow_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$*, then* $\Gamma'\|\Delta'\|\Lambda'$ *is a directed conservative extension of* $\Gamma\|\Delta\|\Lambda$*. If there is no such set triple, then* $\Gamma\|\Delta\|\Lambda$ *has no solution.*

Then by induction via Lemma 3.21, we get:

Theorem 3.22 *Let* $\Gamma\|\Delta\|\Lambda$ *be a set triple. If there exists another set triple* $\Gamma'\|\Delta'\|\Lambda'$*, such that* $\Gamma\|\Delta\|\Lambda \overset{+}{\Rightarrow}_{\mathfrak{I}_{ACUNh}} \Gamma'\|\Delta'\|\Lambda'$*, then* $\Gamma'\|\Delta'\|\Lambda'$ *is a directed conservative extension of* $\Gamma\|\Delta\|\Lambda$*. If there is no such set triple, then* $\Gamma\|\Delta\|\Lambda$ *has no solution.*

From above theorem we have the following corollary:

Corollary 3.23 (Completeness) Let Γ be an ACUNh-unification problem. Suppose after applying the inferences rules from $\mathfrak{I}_{XOR}$ to $\Gamma\|\emptyset\|\emptyset$ exhaustively, we get $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i)$, i.e.

$$\Gamma\|\emptyset\|\emptyset \overset{*}{\Rightarrow}_{\mathfrak{I}_{ACUNh}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i)$$

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

So far, we proved our inference system $\mathfrak{I}_{ACUNh}$ is terminating, sound and complete.

3.10 Auxiliary Rules for Improving Efficiency

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

Dis-Trivial

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

Clash

$$\frac{(\Gamma \cup \{\mathbf{S} \oplus f(t_{11}, \cdots, t_{1n}) \oplus \cdots \oplus f(t_{m1}, t_{m2}, \cdots, t_{mn}) \stackrel{?}{=} 0\}) \| \Delta \| \Lambda}{\mathbf{Fail}}.$$

if there is neither a pure variable nor a term with uninterpreted function symbol f as top symbol in $\mathbf{S}$, and m is odd.

Example 3.24 Solve the equation:

$$g(x) \oplus g(y) \oplus f(x) \oplus f(y) \oplus f(z) \stackrel{?}{=} 0.$$

From Clash, this results in a fail case.

Clash-Annul

$$\frac{(\Gamma \cup \{\mathbf{S} \oplus h(t) \stackrel{?}{=} 0\}) \| \Delta \| \Lambda}{(\Gamma \cup \{\mathbf{S} \stackrel{?}{=} 0\} \cup \{t \stackrel{?}{=} 0\}) \| \Delta \| \Lambda}.$$

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

Occur Check

$$\frac{(\Gamma \cup \{x_1 \oplus x_2 \oplus \cdots \oplus x_n \oplus t_1 \oplus t_2 \oplus \cdots \oplus t_m \stackrel{?}{=} 0\}) \| \Delta \| \Lambda}{\mathbf{Fail}}.$$

where, $i > 0$, and for all x_i, there exists a term t_j with an uninterpreted function symbol on the top of it, such that:

1. $x_i \in Vars(t_j)$;

2. $|Top(t_j \; ; \; \{t_1, t_2, \cdots, t_m\})|$ is odd; and

3. x_i occurs in every term in $Top(t_j\ ;\ \{t_1, t_2, \cdots, t_m\})$.

Example 3.25 Solve the equation:

$$x \oplus y \oplus f(x,y) \oplus f(x,g(y)) \oplus f(y,f(x)) \overset{?}{=} 0.$$

Variables x and y are pure. Occur Check results in Fail.

Occur Check-Annul

$$\frac{(\Gamma \cup \{x \oplus \mathbf{S} \oplus h(t) \overset{?}{=} 0\})\|\Delta\|\Lambda}{(\Gamma \cup \{t \overset{?}{=} 0\} \cup \{\mathbf{S} \overset{?}{=} 0)\|\Delta\|\Lambda\}}.$$

if no pure variable in $\mathbf{S}$ and x occurs in t.

Example 3.26 Solve the unification problem:

$$\{x \oplus h(x) \overset{?}{=} 0, y \oplus h(y) \overset{?}{=} 0,$$
$$z \oplus h(z) \overset{?}{=} 0, f(x) \oplus f(y) \oplus f(z) \oplus f(w) \overset{?}{=} 0\}.$$

From Occur Check-Annul, $x \leftarrow 0, y \leftarrow 0, z \leftarrow 0$, and we can easily get $w \leftarrow 0$. If we do not have Occur Check-Annul, we will use N-Decomposition, which is much less efficient.

Decomposition

If all the pure variables x_i in $\mathbf{S}$ occur in some term s_j (or t_j) and no top symbol of a pure term of $\mathbf{S}$ is uninterpreted function symbol f, then:

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

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

Example 3.27 Solve the unification problem:

$$\{f(x) \oplus f(y) \oplus x \overset{?}{=} 0, f(x) \oplus f(y) \oplus f(z) \oplus f(w) \overset{?}{=} 0\}.$$

From Decomposition on the first equation followed by Variable Substitution, $x \leftarrow 0, y \leftarrow 0$, then applying Decomposition on the second one, we get $w \leftarrow z$. If we did not have N-Decomposition, we would use N-Decomposition, which is less efficient.

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

Lemma 3.28 *Suppose* $\mathbf{S}$ *has the reduced form* $t_1 \oplus \cdots \oplus t_n$, *where* t_i *is not a variable, and no* f *occurs as top symbol in* t_i.

$$\mathbf{S} \oplus f(s_{11}, s_{12}, \cdots, s_{1n}) \oplus f(s_{21}, s_{22}, \cdots s_{2n}) \oplus \cdots \oplus f(s_{m1}, s_{m2}, \cdots s_{mn}) \overset{?}{=} 0$$

has no solution if m *is odd.*

Proof. If there is a ground solution σ for this equation, after applying the substitution, we get

$$\mathbf{S}' \oplus f(s'_{11}, s'_{12}, \ldots, s'_{1n}) \oplus f(s'_{21}, s'_{22}, \cdots s'_{2n}) \oplus \cdots \oplus f(s'_{m1}, s'_{m2}, \cdots s'_{mn}) \overset{?}{=} 0$$

According to our definition of substitution, no f occurs as top symbol in $\mathbf{S}'$, for we have no pure variable occurring in $\mathbf{S}$.

Next, we use induction on m to prove the lemma.

Suppose $m = 1$, which means there is only one f occurring in the original equation as top function symbol. So σ satisfies $(S\sigma \oplus f(s_1, s_2, \cdots s_n)\sigma) \downarrow = 0$. But from our rewriting rules, the only way to cancel an uninterpreted function symbol is to use an identical term. In $\mathbf{S}$, no other f occurs on the top, so it is impossible that $(S\sigma \oplus f(s_1, s_2, \cdots s_n)\sigma) \downarrow = 0$.

Suppose for any odd number of f, which is less than $2n+1$, the lemma is true. If the number of f is $2n+1$, then suppose we can cancel one of the terms which has f as top symbol by another term with top symbol f in the equation. Then our equation contains $2n-1$ fs. From the assumption, the lemma is true. If there is no term with top symbol of f being canceled, obviously, this lemma is true.

So σ is not a solution of the original equation. □

Theorem 3.29 *All the auxiliary rules are (P, E, μ)-Simplifiers, where, P is in* PURE SUM *form, E is Exclusive OR and μ is $\mathbb{M}_{XOR}(\Gamma, \Delta, \Lambda)$.*

Proof. From the rules, we know there are no new variables generating during the procedure, so for proving they are E-equation preservative, it is enough 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. And from Lemma 3.28 we know that Clash is E-equation preservative.

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

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

Without loss of generality, we suppose $|Sym(s_1)| \geq |Sym(s_2)| \geq \cdots \geq |Sym(s_n)|$, $x_1 \in t_1$, $Top(t_1) = f$, 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 l is an odd number from the condition of Occur Check. Because l is an odd number, from the proof of Lemma 3.28, 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 symbols can be canceled after applying the substitution, which means $x_i\theta = s_i = t_i[x_1]\theta \oplus S\theta = t_i[s_1]\theta \oplus S\theta$, where S is a term. This means $Sym(S_1) < Sym(s_i)$ which is a contradiction. So in this case, there is no solution. This means Occur Check is E-equation preservative.

For Occur Check-Annul, if there is a solution θ, then $x\theta \oplus \mathbf{S}\theta \oplus h(t\theta) = 0$. Because $x \oplus \mathbf{S} \oplus h(t\theta) \overset{?}{=} 0$ 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 $x\theta = h(t\theta) \neq 0$ and the other one is $t\theta = 0$. For the first case, because x occurs in t, then $Sym(x\theta) = Sym(h(t\theta)) > Sym(x\theta)$, which is a contradiction. So the only possible case is the second case, which means θ is also a solution of the conclusion. The other direction is trivial.

For Decomposition, if $f(s_1, s_2, \cdots s_m) \oplus f(t_1, t_2, \cdots t_m) \neq 0$, 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 which are variables or have top symbol f. So if there is a solution δ for the premise, then $f(s_1, s_2, \cdots s_m)\delta \oplus f(t_1, t_2, \cdots t_m)\delta = 0$, which means δ is also a solution of the conclusion. The other direction is trivial.

For proving the rules are P-preservative, where P is in PURE SUM form, we only need to prove the case of the Decomposition rule, and since σ is a result of syntactic unification, there is no $\oplus$ in it. So the conclusion is still in PURE SUM form.

For μ-reduced, Trivial, Clash, Clash-Annul, Occur Check and Occur Check-Annul are trivially true. For Decomposition, either some variables are solved or two fs are removed, and $\mathbb{M}_{XOR}(\Gamma, \Delta, \Lambda)$ decreases. So the conclusion is μ-reduced.

Hence all of the above rules are simplifiers. □

3.11 Implementation

Due to the complication of the combination algorithm [54], as far as we know, there is no public implementation for the general ACUNh-unification algorithm yet.

Thanks to Maude [1], we have implemented our algorithm in an intuitive way. The implemented program has resolved many ACUNh-unification problems. The running environment is on Windows 7, Intel Core2 Duo 2.26GHz, and RAM 5 GB using Maude 2.6. Table 3.1 shows some of our results.

In Table 3.1, "YES" means the algorithm outputs the minimal complete set of unifiers for the unification problem. As we can see in the table, in all of the cases, we got the minimal complete set of unifiers. Except for the second and third one, we got the final result in less than 0.1 seconds. 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.

Problems	Real Time	# Sol.	Minimal?
$f(x) \oplus f(y) \overset{?}{=} f(a) \oplus f(b)$	57ms	2	YES
$f(x) \oplus f(y) \oplus f(z) \overset{?}{=} f(a) \oplus f(b) \oplus f(c)$	1493ms	6	YES
$f(x) \oplus f(y) \oplus f(z) \oplus f(w) \overset{?}{=}$ $f(a) \oplus f(b) \oplus f(c) \oplus f(d)$	29.9s	24	YES
$x \overset{?}{=} f(x \oplus y)$	3 ms	1	YES
$x \overset{?}{=} f(x \oplus y \oplus f(y))$	7ms	1	YES
$x \overset{?}{=} h(x \oplus y \oplus h(y))$	9ms	1	YES
$f(b) \oplus f(0) \overset{?}{=} f(x) \oplus f(y)$	62ms	2	YES
$x \oplus y \overset{?}{=}$ $h(x \oplus y) \oplus f(h(x \oplus y \oplus z), g(a))$	11ms	0	YES
$f(x, g(y, w)) \oplus f(c, g(a, z))$ $\overset{?}{=} w \oplus g(a, c)$	7ms	1	YES
$f(x, g(y, b)) + f(c, g(a, z)) \overset{?}{=} w$ $f(w) \oplus f(d) \overset{?}{=} 0$	3ms	0	YES
$h(x \oplus y) \overset{?}{=} 0$	4 ms	1	YES
$x \oplus f(y) \oplus f(x_1) \overset{?}{=} 0$ $y \oplus f(z) \oplus f(x_2) \overset{?}{=} 0$ $z \oplus f(x) \oplus f(x_3) \overset{?}{=} 0$	54ms	3	YES
$x \oplus h(y) \oplus h(z) \overset{?}{=} 0$ $y \oplus h(x) \oplus h(z) \overset{?}{=} 0$ $z \oplus h(x) \oplus h(y) \overset{?}{=} 0$	34ms	1	YES

Table 3.1: Experiment Result for ACUNh Unification Algorithm

3.12 Conclusion and Future Work

We introduced inference rules for general E-unification problems modulo XOR with homomorphism. We proved that these inference rules are 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 XOR without homomorphism. In this case, the Variable Substitution rule becomes simpler, because the conditions involving the h symbols are trivially true, N-Decomposition remains the same, and Annulization never applies. The auxiliary rules are the same, but Clash-Annul and Occur-Check-Annul no longer apply.

XOR is an important theory in cryptographic protocols, and that is the focus of our research. The algorithms are simple to implement, and have already been implemented in Maude. 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.

The theory of Abelian groups, especially with homomorphism, is also an important theory in cryptographic protocols. That is our next extension of this theory. For future work, we will also combine it with convergent rewrite theories like cancellation.

Chapter 4

Efficient General AGh-Unification

4.1 Introduction

As we said in Section 3.1, the techniques for solving general unification problems so far are highly nondeterministic and generate a highly redundant complete set of unifiers. Also, due to the complexity of the algorithms, they are not easy to implement. In this chapter, 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 AG-unification, along with uninterpreted function symbols. We have implemented our inference rules in Maude [13].

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 Chapter 3. 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.

Here we give the outline of this chapter. In Section 2, we give preliminary knowledge about 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 is given in Section 4; 2. the inference rules which are used to solve the unification problem are given in Section 6 and 7, where section 6 presents the necessary rules and Section 7 presents a kind of 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.

4.2 Preliminary

In this section, we introduce the basic notation related to the theory of a homomorphism over an Abelian group. We simplify 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 4.1 In the following set of terms:

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

only $-x, g(f(a, b))$, $h(f(a, b))$ and $f(g(a), h(b))$ are pure terms; $h(a + y)$ and $h(f(a, b))$ are h-terms but $h(a + b)$ is not a pure term; $a + b, -h(x)$ 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 4.2 *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 $\mathbf{S} \stackrel{?}{=} 0$ 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 $x + S \overset{?}{=} 0$. 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, $s \overset{?}{=}_{\mathbf{AGh}} t \iff s + (-t) \overset{?}{=}_{\mathbf{AGh}} 0$, we only consider equations of the form $\mathbf{S} =^{?} 0$. For convenience, we write an **AGh**-unification problem as $\{s_1 \overset{?}{=} 0, s_2 \overset{?}{=} 0, \cdots, s_n \overset{?}{=} 0\}$, where each s_i is a term.

4.3 Rewriting System $\mathfrak{R}_{AGh}$

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

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

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 [19]. 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 [19] 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:

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

where $c + d$ 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.*

4.4 Initialization

Our inference rules will have some requirement on the form of the problem. We require all the equations in the problem true 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.

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

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:

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

and

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

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 $\{x + h(x) \overset{?}{=} 0, y + h(x) \overset{?}{=} 0\}$, $h(x)$ occurs twice, but we want it to only occur once. So we replace $y + h(x) \overset{?}{=} 0$ by $y + x \overset{?}{=} 0$.

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.

The purification procedure will finally terminate.

For purification, we also have the following lemma:

Lemma 4.3 *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 4.4 Purify the equation:

$$5f(x_1, h(x_2)) + h(x_2) + 2g(x_3 + f(x_2, x_1)) \stackrel{?}{=} 0$$

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:

$$\begin{aligned}\{&5f(x_1, v_1) + h(x_2) + 2g(x_3 + f(x_2, x_1)) \stackrel{?}{=} 0\\ &v_1 + h(-x_2) \stackrel{?}{=} 0\}\end{aligned}$$

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:

$$\begin{aligned}
&\{5f(x_1, v_1) + h(x_2) + 2g(v_2) \overset{?}{=} 0 \\
&v_1 + h(-x_2) \overset{?}{=} 0 \\
&v_2 + (-x_3) + (-f(x_2, x_1)) \overset{?}{=} 0\}
\end{aligned}$$

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.

$$\begin{aligned}
&\{5f(x_1, v_1) + h(x_2) + 2g(v_2) \overset{?}{=} 0 \\
&v_1 + 5f(x_1, v_1) + 2g(v_2) \overset{?}{=} 0 \\
&v_2 + (-x_3) + (-f(x_2, x_1)) \overset{?}{=} 0\}
\end{aligned}$$

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 $\mathbf{S} \overset{?}{=} 0$, 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.

4.5 UnconstrainedReduce Algorithm

In [36], the authors present the relation between unification modulo Abelian Groups and solving linear equation systems. And in [3] and [48], 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:

- $x + 2 = 0$, the solution is $x = -2$.
- $2x + 1 = 0$, no integer solutions.
- $4x + 5 = 0$, 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 [34], 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 details, please refer to [34]):

Example 4.5

$$2x + 6 = 0$$

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 4.6

$$3x + 2y = 0$$

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 applying this method to the equations to the equations which has no solution:

Example 4.7

$$4x + 5 = 0$$

.

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 finally, there is only one variable in the equation and this variable's coefficient is greater than the constant.

If an equation has no solution, finally 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:

- $x + 2a \overset{?}{=} 0$;
- $x + a \overset{?}{=} 0$;
- $4x + 5a \overset{?}{=} 0$;
- $2x + 6a \overset{?}{=} 0$;
- $3x + 2y \overset{?}{=} 0$.

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 $x + f(x) \overset{?}{=} 0$, we cannot let $x \leftarrow x' + f(x)$, which will cause a loop.
- Homomorphism plays an important role in our unification problem. For example, for $2x + h(y) \overset{?}{=} 0$, $x \leftarrow h(z), y \leftarrow -2z$ 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 [34] 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 $2x + h(z) \stackrel{?}{=} 0$. We can delete other occurrences of this unconstrained variable in other equations. For example, if we also have $3x + 4z \stackrel{?}{=} 0$, we can replace it by $8z - h(3z) \stackrel{?}{=} 0$, since $2x + h(z) \stackrel{?}{=} 0 \iff 6x + h(3z) \stackrel{?}{=} 0$ and $3x + 4z \stackrel{?}{=} 0 \iff 6x + 8z \stackrel{?}{=} 0$. 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 $\mathbf{Maxi}(Q, V)$. 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 $\mathbf{Maxi}(Q, V)$. $\tilde{\Lambda}$ is used to store the substitutions which are generated during $\mathbf{Maxi}(Q, V)$.

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 cancelled 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 $4x + 3y + 5z \overset{?}{=} 0$, then different Vs will give different results. If $V = \{z\}$, then 5 is the biggest coefficient already. We do not need to do anything. If $V = \{x\}$, then we will chose x and let $x \leftarrow x' - z$ and get $4x' + 3y + z \overset{?}{=} 0$. 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 $z \leftarrow -4x' - 3y$ 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 unconstrained 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**.

Maxi(Q, V)

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.

Procedure:

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.
   - Suppose $|C_Q| = 1$,
     - if $|C'_Q| = 0$, add $x_1 \overset{?}{=} 0$ into Λ' and let $\tilde{\Lambda} = \Lambda'$ and let Ω be the empty set, then **return** $(\Omega, \tilde{\Lambda})$.
     - if $|C'_Q| \neq 0$ and $|c_1|$ is bigger than all the elements in C'_Q, then let $\tilde{\Lambda} = \Lambda'$, $\tilde{Q} = Q'$ and $\Omega = \{\tilde{Q}\}$, then **return** $(\Omega, \tilde{\Lambda})$.
   - Otherwise let $min = j$ if $|c_j| \leq |c_i|$ for all $0 < i < n$. If j is not unique, choose any one of them.
3.
   - If $c_{min} = 1$, add equation $x_{min} \overset{?}{=} -c_1x_1 + \cdots + (-c_{min-1}x_{min-1}) + (-c_{min+1}x_{min+1}) + \cdots + (-c_nx_n) + (-c'_1t_1) + \cdots + (-c'_mt_m)$ into Λ', let $\tilde{\Lambda} = \Lambda'$ and let Ω be the empty set. **Return** $(\Omega, \tilde{\Lambda})$.
   - If $c_{min} = -1$, add equation $x_{min} \overset{?}{=} c_1x_1 + \cdots + c_{min-1}x_{min-1} + c_{min+1}x_{min+1} + \cdots + c_nx_n + c'_1t_1 + \cdots c'_mt_m$ into Λ' and let $\tilde{\Lambda} = \Lambda'$ and let Ω be the empty set, **Return** $(\Omega, \tilde{\Lambda})$.

- If $|c_{min}| > 1$, then introduce a new variable x', and
  (a) add an equation P

$$x_{min} \stackrel{?}{=} x' + (-\lfloor \frac{c_1}{c_{min}} \rfloor x_1) + \cdots + (-\lfloor \frac{c_{min-1}}{c_{min}} \rfloor x_{min-1}) + (-\lfloor \frac{c_{min+1}}{c_{min}} \rfloor x_{min+1}) + \cdots + (-\lfloor \frac{c_n}{c_{min}} \rfloor x_n) + (-\lfloor \frac{c'_1}{c_{min}} \rfloor t_1) + \cdots + (-\lfloor \frac{c'_m}{c_{min}} \rfloor t_m)$$

  into $\tilde{\Lambda}$;

  (b) apply $\lambda_{\{P\}}$ to Q' and Λ';

  (c) remove the variables which are not in Q' from V' and add x' to V'; 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';

  (d) go back to step 2.

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).

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**$(Q, V))$.

Now we present the algorithm **Uniq**.

Uniq$(\Gamma, (\Omega, \tilde{\Lambda}))$

Input:

- An equation set Γ.
- A set Ω, which is the empty set or $\{cx + \mathbf{T} \overset{?}{=} 0\}$, where $|c|$ is larger than all the absolute values of the coefficients of the monic terms in T.
- An equation set in solved form $\tilde{\Lambda}$

Procedure:

```
1. Let $\Gamma' = \Gamma\lambda_{\tilde{\Lambda}}$ and $\Lambda' = \tilde{\Lambda}$
2. Look at $\Omega$:
    • if $\Omega$ is the empty set, then let $\tilde{\Gamma} = \Gamma'$, $\tilde{\Lambda'} = \Lambda'$ and **return** $(\tilde{\Gamma}, \tilde{\Lambda'})$.
    • if $\Omega$ is not the empty set, then for every equation in $\Gamma'$, which has the form $c'x + \mathbf{S} \overset{?}{=} 0$, do:
        – if $\mathbf{S}$ is zero, add equation $x \overset{?}{=} 0$ into $\Lambda'$, and let $\tilde{\Gamma} = (\Gamma' \cup \{T \overset{?}{=} 0\})[x \leftarrow 0]$ and $\tilde{\Lambda'} = \Lambda'$. **Return**$(\tilde{\Gamma}, \tilde{\Lambda'})$.
        – if $\mathbf{S}$ is not zero, remove $c'x + \mathbf{S} \overset{?}{=} 0$ from $\Gamma'$ and add $-c'\mathbf{T} + c\mathbf{S} \overset{?}{=} 0$ into $\Gamma'$.
3. Let $\tilde{\Gamma} = \Gamma' \cup \{cx + \mathbf{T} \overset{?}{=} 0\}$ and $\tilde{\Lambda'} = \Lambda'$, **Return** $(\tilde{\Gamma}, \tilde{\Lambda'})$
```

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

Example 4.8 Run **Maxi**$(5x + 13y + 7z + 4f(z) \overset{?}{=} 0, \{x, y\})$. First x has the smallest coefficient among x and y, we set $x \leftarrow v_1 - 2y - z$ and get

$$5v_1 + 3y + 2z + 4f(z) \overset{?}{=} 0$$

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

$$2v_1 + 3v_2 + 2z + f(z) \overset{?}{=} 0$$

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

$$2v_3 + v_2 + f(z) \overset{?}{=} 0$$

and the new variable set $\{v_3, v_2\}$. Here v_2's coefficient is 1, we can solve this equation by setting $v_2 \leftarrow (-2v_3) + (-f(z))$.

Finally, we get a solved equation set $\tilde{\Lambda} = \{x \leftarrow -5v_3 + (-3f(z)) + z, y \leftarrow 3v_3 + fz + (-z)\}$

If we run this algorithm for this equation on the variable set $\{x\}$, then since 5 is less then 13, we will set $x \leftarrow v_1 - 2y - z$ and get $5v_1 + 3y + 2z + 4f(z) \overset{?}{=} 0$ 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 $\{x \overset{?}{=} v_1 + (-2y) + (-z)\}$.

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

Example 4.9 Let

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

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

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

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

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

then we get

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

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, $t \stackrel{?}{=} 0 \iff ct \stackrel{?}{=} 0$, where t is a term and c is a coefficient. This fact can be proven by narrowing modulo $\mathcal{AC}$.

Lemma 4.10 *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 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 $N \mod n$. 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 $\{Q''\} = \{c_1 x_1 \overset{?}{=} 0\}$, $\{Q'\} = \emptyset$ and $\Lambda' = \Lambda' \cup \{x \overset{?}{=} 0\}$. Hence $\{Q'\} \cup \Lambda'$ is a conservative extension of $\{Q''\} \cup \Lambda''$.

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

$$\{Q''\} = \{c_1 x_1 + \cdots + x_{min} + \cdots + c_n x_n + c'_1 t_1 + \cdots + c'_m t_m =^? 0\}$$

$$\{Q'\} = \emptyset$$

$$\begin{aligned}\Lambda' = \Lambda'' \cup \{ & x_{min} = -c_1 x_1 + \cdots + (-c_{min-1} x_{min-1}) + \\ & (-c_{min+1} x_{min+1}) + \cdots + c_n x_n + c'_1 t_1 + \cdots + c'_m t_m =^? 0\}\end{aligned}$$

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

$\{Q''\} \cup \Lambda''$.

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

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

$$\{Q''\} = \{c_1 x_1 + \cdots + c_n x_n + c_1' t_1 + \cdots + c_m' t_m =^? 0\}$$

$$\begin{aligned}\{Q'\} = \ &\{(c_1 \mod c_{min})x_1 + \cdots + (c_{min-1} \mod c_{min})x_{min-1}\\ &c_{min}x' + (c_{min+1} \mod c_{min+1})x_{min+1} + \cdots (c_n \mod c_{min})x_n\\ &(c_1' \mod c_{min})t_1 + \cdots + (c_m' \mod c_{min})t_m \overset{?}{=} 0\}\end{aligned}$$

and

$$\begin{aligned}\Lambda' = \Lambda'' \cup \{x_{min} \overset{?}{=} x' + (-\lfloor \frac{c_1}{c_{min}} \rfloor x_1) + \cdots + (-\lfloor \frac{c_{min-1}}{c_{min}} \rfloor x_{min-1})\\ + (-\lfloor \frac{c_{min+1}}{c_{min}} \rfloor x_{min+1}) + \cdots + (-\lfloor \frac{c_n}{c_{min}} \rfloor x_n) +\\ (-\lfloor \frac{c_1'}{c_{min}} \rfloor t_1) + \cdots + (-\lfloor \frac{c_m'}{c_{min}} \rfloor t_m)\}\end{aligned}$$

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

$$(c_1 x_1 + \cdots + c_n x_n + c_1' t_1 + \cdots + c_m' t_m)\theta = 0$$

then

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

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

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

which satisfies Λ'

Then

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

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

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

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

then

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

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

Lemma 4.11 *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 $c'x + \mathbf{S} \overset{?}{=} 0$ we will do several things:

- If $\mathbf{S} = 0$, then the algorithm will stop after this step.
- If $\mathbf{S} \neq 0$, $c'x + \mathbf{S} \overset{?}{=} 0$ will be removed from Γ' and add $c'(-T) + cS \overset{?}{=} 0$ 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 $c'x \overset{?}{=} 0$, 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 $\Gamma' \cup \Lambda' \cup \{T \overset{?}{=} 0\}$ in this step.

If Ω is not empty and no equation in Γ' has the form $c'x \overset{?}{=} 0$, then if we have $c'x + \mathbf{S} \overset{?}{=} 0$ in it, the second part of the second part in step 2 will remove it and add a new $-c'T + cS \overset{?}{=} 0$. As we proved above, every time we run this step, N_x will decrease until no $c'x + \mathbf{S} \overset{?}{=} 0$ 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

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

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 $\theta \vDash c'(-\mathbf{T}) + c\mathbf{S} \overset{?}{=} 0$.

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 4.12 *Let Γ be a unification problem, Q be an equation and $V = \{x \in PV(Q) \mid x$ 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 4.10 and Lemma 4.11 by restricting V to the unconstrained variables in Γ. □

If **Uncons($\boldsymbol{\Gamma}$)** is used to denote the set: $\{x \in Vars(\Gamma) \mid x$ 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.

4.6 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**.

4.6.1 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 $\{\mathbf{S_1} \overset{?}{=} 0, \mathbf{S_2} \overset{?}{=} 0, \cdots, \mathbf{S_n} \overset{?}{=} 0\}$, 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 $x \overset{?}{=} S$ will be added to Λ if we apply the substitution $x \leftarrow S$ to Γ. All the equations in Λ have the form $x \overset{?}{=} S$, 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 $x \overset{?}{=} S$ is put into Λ. Initially, this set is empty.

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

$$3x + h(y) \overset{?}{=} 0$$

There is a solution: $[x \leftarrow h(z), y \leftarrow -3z]$. Since the unconstrained variable's (x) coefficient is greater than $h(y)$, 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, $y \leftarrow 6b+5a$, no matter what x's substitution is, is not a solution. If $y \leftarrow 3a$, we can let $x \leftarrow h(-a)$ to get the solution. So we can guess the solution of this algorithm has a solution of the form $y \leftarrow 3z$.

- This is not enough, since it is possible the equation contains other terms. For example $3x + 2z + h(y) \overset{?}{=} 0$(if z occurs in other equations), $y \leftarrow 3w + 2w', x \leftarrow h(-w), z \leftarrow h(-w')$ 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 $y \leftarrow 3w+w_1$, then guess $w_1 \leftarrow 2w'$.
- Generally, from the Division Algorithm, if we guess $y \leftarrow kw + w'$, then next, if we need to guess $w' \leftarrow k'w'' + t$, 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:

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

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_Γ^{ini}. 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 $h(cs) + h(dt) + S \overset{?}{=} 0$ in Γ. After being rewritten and **Purify**, this equation is replaced by $h(x') + S \overset{?}{=} 0$ and $x' + (-cs) + (-dt) \overset{?}{=} 0$, 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.

4.6.2 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, and N-Decomposition is nondeterministic. 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**-unifiers of Γ, or **Fail** if Γ is not unifiable.

Now we present the inference rules.

Trivial

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

Trivial

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

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 $3x + a \overset{?}{=} 0$) or we need to guess a solution(like $3x + h(z) \overset{?}{=} 0$), which needs our other inference rules.

Formally, the inference rule is as follows:

Variable Substitution

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

if each x_i is an unconstrained variable in $\Gamma \cup \{c_1x_1 + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m \overset{?}{=} 0\}$ 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$(\Gamma, c_1x_1+\cdots+c_nx_n+c'_1t_1+\cdots+c'_mt_m \overset{?}{=} 0, \{x_1, x_2, \cdots, x_n\})$ 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 equations 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 $x' - s - t \overset{?}{=} 0$ into Γ'. The change from Ψ to Ψ' will be according the following rule:

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

Here s and t are monic terms and x is a fresh variable. If s (or t) is not a variable, then N_s (or N_t) is 1.

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

Example 4.13 In the equation set:

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

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:

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

.

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

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

$$\{2v_1 + z \stackrel{?}{=} 0\}$$

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

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

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

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

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

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

where v_1 is generated by purification.

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

$$\{v_1 + y + h(v_1) \overset{?}{=} 0, y + h(v_2) \overset{?}{=} 0, v_2 + v_1 + y \overset{?}{=} 0\}$$

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 $x + y + z \overset{?}{=} 0$ and solve x first, we get $x \leftarrow -y + (-z)$ and a new equation set:

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

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 4.17 .

The reason we only apply our algorithm on unconstrained variables is to control looping. For example, suppose we have $\{x + f(z) \overset{?}{=} 0, z + x + y \overset{?}{=} 0\}$. If we choose z to apply our algorithm, we get $z \leftarrow -x + (-y)$, then $x + f(z) \overset{?}{=} 0$ becomes $x + f(-x + (-y))$. After purification, we will get $\{x + f(v_1) \overset{?}{=} 0, v_1 + x + y \overset{?}{=} 0\}$, 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.

N-Decomposition

The next inference rule N-Decomposition is nondeterministic. This is to solve a case, like $cf(x) + df(y) + ef(z) \overset{?}{=} 0$. In this case there are several possible guesses: $\{f(x) = f(y) \neq f(z)\}$, $\{f(z) = f(y) \neq f(x)\}$, $\{f(x) = f(z) \neq f(y)\}$, $\{f(x) \neq f(z), f(x) \neq f(y), f(y) \neq f(z)\}$, and $\{f(x) = f(y) = f(z)\}$. Since our problems is in PURE SUM form, these guesses are several syntactical unification problem 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

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

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

$$\sigma = mgu(s_1 \stackrel{?}{=} t_1, s_2 \stackrel{?}{=} t_2, \cdots, s_m \stackrel{?}{=} t_m)$$

$$\Gamma_1' = \Gamma \cup \{\mathbf{S} + cf(s_1, s_2, \cdots, s_m) + (-df(t_1, t_2, \cdots, t_m)) \stackrel{?}{=} 0\}$$

$$\Delta_1' = \Delta \cup \{f(s_1, s_2, \cdots, s_m) + (-f(t_1, t_2, \cdots, t_m)) \neq^? 0\}$$

$$Q = \mathbf{S} + (c - d)f(s_1, s_2, \cdots, s_m) \stackrel{?}{=} 0$$

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 4.15 For the equation set:

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

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 $\Psi = \emptyset$

$$\{x + 2f(y) + (-2f(z)) \overset{?}{=} 0, y + 3f(x) + (-3f(w)) \overset{?}{=} 0\} \| \emptyset \| \emptyset \| \emptyset$$

$$\Longrightarrow \text{(N-Decomposition)}$$

$$\{x \overset{?}{=} 0, z + 3f(x) + (-3f(w)) \overset{?}{=} 0\} \| \emptyset \| \{y \overset{?}{=} z\} \| \emptyset$$

$$\bigvee \Gamma_1 \| \Delta_1 \| \emptyset \| \emptyset$$

$$\overset{*}{\Rightarrow} \text{(several steps by Variable Substitution)}$$

$$\emptyset \| \emptyset \| \{x \overset{?}{=} 0, y \overset{?}{=} 3f(w) + (-3f(0)), z \overset{?}{=} 3f(w) + (-3f(0))\} \| \emptyset \bigvee \Gamma_1 \| \Delta_1 \| \emptyset \| \emptyset$$

where

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

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

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

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

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.

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 $z \leftarrow 3z'$, since $[x \leftarrow z', z \leftarrow 3z']$ is a solution of $3x + h(z) = 0$.

Factorization makes 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

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

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

Example 4.16 For the equation set

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

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

$\{2y + x_2 + h(x_1) \overset{?}{=} 0, 3z + x_1 + h(x_2) \overset{?}{=} 0\} \| \emptyset \| \emptyset \| \{(\infty, x_1), (\infty, x_2)\}$

$\Longrightarrow$ (Factorization)

$\{2v_3 + x_2 + h(v_2) \overset{?}{=} 0, 3z + 2v_1 + v_2 + h(x_2) \overset{?}{=} 0\} \| \emptyset \|$

$\{x_1 \overset{?}{=} 2v_1 + v_2, y \overset{?}{=} v_3 + h(-v_1)\} \| \{(2, v_2), (\infty, x_2)\}$

$\Longrightarrow$ (Variable Substitution on the second equation)

$\{2v_3 + x_2 + h(v_2) \overset{?}{=} 0\} \| \emptyset \| \{x_1 \overset{?}{=} 6v_4 + 3v_2 + h(2x_2)$

$y \overset{?}{=} v_3 + h(-3v_4 + (-v_2) + h(-x_2)), v_1 \overset{?}{=} 3v_4 + v_2 + h(x_2),$

$z \overset{?}{=} -2v_4 + (-v_2) + h(-x_2)\} \| \{(2, v_2)\}$

$\Longrightarrow$ (Variable Substitution)

$\emptyset \| \emptyset \| \{x_1 \overset{?}{=} 6v_4 + 3v_2 + h(-4v_3 + h(-2v_2)), y \overset{?}{=} v_3 + h(-3v_4 + (-v_2) + h(2v_3 + h(v_2))),$

$v_1 \overset{?}{=} 3v_4 + v_2 + h(-2v_3 + h(-v_2)), x_2 \overset{?}{=} -2v_3 + h(-v_2),$

$z \overset{?}{=} -2v_4 + (-v_2) + h(2v_3 + h(v_2))\} \| \emptyset$

Then we get the solution set

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

The purpose of $n > |c|$ is to avoid loops, we will see this case in Example 4.18 which is after the next rule. In the next section, we will prove this constraint will not lose any solutions.

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

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

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 4.14.

Example 4.17 For convenience, we still do not include Δ, Λ and Ψ here.

For equation set

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

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

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

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

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

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

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

Using Trivial, we get the empty set and the solution

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

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

Example 4.18 For equation

$$3y + 2x + h(x) \overset{?}{=} 0$$

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

$\{3y + 2x + h(x) \overset{?}{=} 0\}\|\emptyset\|\emptyset\|\{(\infty, x)\}$

$\Longrightarrow$ (Factorization)

$\{3v_3 + 2v_2 + h(v_2) \overset{?}{=} 0\}\|\emptyset\|\{x \leftarrow 3v_1 + v_2, y \leftarrow v_3 + (-2v_1) + h(-v_1)\}\|\{(3, v_2)\}$

$\Longrightarrow$ (Annulization)

$\{3v_3 \overset{?}{=} 0\}\|\emptyset\|\{x \leftarrow 3v_1, y \leftarrow v_3 + (-2v_1) + h(-v_1), v_2 \leftarrow 0\}\|\emptyset$

$\Longrightarrow$ (Annulization)

$\emptyset\|\{x \leftarrow 3v_1, y \leftarrow -2v_1 + h(-v_1), v_2 \leftarrow 0, v_3 \leftarrow 0\}\|\emptyset$

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

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

4.7 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 4.19 (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 used 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 4.20 (Preservative 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 preservative**. If for some property P, $P(\Gamma')$ is true whenever $P(\Gamma)$ is true, we say I is P-**preservative**. 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 Preservative, P-preservative 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**.*

4.8 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 $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$.

- $\Gamma\|\Delta\|\Lambda\|\Psi \overset{+}{\Rrightarrow}_{\Im_{AGh}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, 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 $\Im_{AGh}$ to $\Gamma\|\Delta\|\Lambda\|\Psi$.

- $\Gamma\|\Delta\|\Lambda\|\Psi \overset{*}{\Rrightarrow}_{\Im_{AGh}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, 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 $\Im_{AGh}$ to $\Gamma\|\Delta\|\Lambda\|\Psi$.

Next, we give seven measures for proving termination:

- Let $\mathbf{H}(\Gamma)$ be the number of h-terms in Γ. Since $\mathbf{H}(\Gamma)$ is a natural number, $\mathbf{H}(\Gamma)$ with standard $\leq$ on natural numbers is a well-founded ordering.

- Let $\mathbf{Coe}(\Psi)$ be a multiset, $\{n_i \mid (n_i, t_i) \in \Psi\}$. Since every n_i is a natural number, the multiset order for $Coe(\Psi)$ is a well-founded ordering.

- Let $\mathbf{Vars}(\Gamma) = \{x \mid x \text{ occurs in some equation in } \Gamma\}$. Since $|Vars(\Gamma)|$ can only be natural number, $|Vars(\Gamma)|$ with standard less than or equal on natural numbers is a well-founded ordering.

- $\mathbf{Cons(\Gamma)}$ 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 $\mathbf{Reduce(\Gamma)}$ to denote the set

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

  Let $\mathbf{RUncons(\Gamma)}$ be $Uncons(\Gamma)\backslash Reduce(\Gamma)$. $|\mathbf{RUncons(\Gamma)}|$ 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 sets.

- 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 $\mathbf{Par}(\Delta)$ be $\{(t, s) \mid t + (-s) \neq^? 0 \in \Delta\}$ and $Par(\Gamma)$ be $\{(t, s) \mid t, s$ occurs in Γ and t has the same top function symbol as $s\}$. We use $\mathbf{Par}(\Gamma \setminus \Delta)$ 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:

$$\begin{aligned}\mathbb{M}_{AGh}(\Gamma, \Delta, \Lambda, \Psi) = (&H(\Gamma), Coe(\Psi),\\ &|Vars(\Gamma)|, |Cons(\Gamma)|, |RUncons(\Gamma)|, |Sym(E)|, |Par(\Gamma \setminus \Delta)|).\end{aligned}$$

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 4.21 *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*(Γ). 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 $\mathbf{S}' + x + h(t) \stackrel{?}{=} 0$, 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. $-\mathbf{dS}' + h(-dt) + h(t') + \mathbf{S}'' \stackrel{?}{=} 0$ will be rewritten to $h(-dt + t') + -\mathbf{dS}' + \mathbf{S}'' \stackrel{?}{=} 0$ and then be purified to $\{h(x') + -\mathbf{dS}' + \mathbf{S}'', x' + dt + (-t') \stackrel{?}{=} 0\}$, 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 $\mathbf{S}' + (-x) + h(t) \stackrel{?}{=} 0$, 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 $c_1x_1 + \cdots + c_nx_n + h(t) + c_1't_1 + \cdots + c_m't_m =^? 0$, 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:

$$\begin{aligned} x' \overset{?}{=} \quad & \lfloor\frac{c_1}{c_i}\rfloor x_1 + \cdots \lfloor\frac{c_{i-1}}{c_i}\rfloor x_{i-1} + x_i + \lfloor\frac{c_{i+1}}{c_i}\rfloor x_{i+1} + \\ & \cdots + \lfloor\frac{c_n}{c_i}\rfloor x_n + \lfloor\frac{1}{c_i}\rfloor h(t) + \lfloor\frac{c_1'}{c_i}\rfloor t_1 + \cdots + \lfloor\frac{c_m'}{c_i}\rfloor t_m \\ = \quad & \lfloor\frac{c_1}{c_i}\rfloor x_1 + \cdots \lfloor\frac{c_{i-1}}{c_i}\rfloor x_{i-1} + x_i + \lfloor\frac{c_{i+1}}{c_i}\rfloor x_{i+1} + \\ & \cdots + \lfloor\frac{c_n}{c_i}\rfloor x_n + \lfloor\frac{c_1'}{c_i}\rfloor t_1 + \cdots + \lfloor\frac{c_m'}{c_i}\rfloor t_m \end{aligned}$$

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 $mgu(s_1 \overset{?}{=} t_1, s_2 \overset{?}{=} t_2, \cdots, s_n \overset{?}{=} t_n)$, 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 4.22 *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 4.21 by induction. □

4.9 Soundness

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

Lemma 4.23 *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 4.12, we know $\tilde{\Gamma} \cup \tilde{\Lambda}$ is a conservative extension of

$$\Gamma \cup \{c_1 x_1 + \cdots + c_n x_x + c_1' t_1 + \cdots + c_m' t_m \overset{?}{=} 0\} \cup \Lambda.$$

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

$$\Gamma \cup \{c_1 x_1 + \cdots + c_n x_x + c_1' t_1 + \cdots + c_m' t_m \overset{?}{=} 0\} \cup \Lambda.$$

So the statement is true.

N-Decomposition:

First Choice:

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

where $\Gamma' = \Gamma \cup \{\mathbf{S} + (c-d)f(s_1, s_2, \cdots, s_m) \overset{?}{=} 0\}$, and $\sigma = mgu(s_1 \overset{?}{=} t_1, s_2 \overset{?}{=} t_2, \cdots, s_m \overset{?}{=} t_m)$.

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

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

So this statement is true.

The second choice is obvious.

Factorization: It is enough to show if

$$
\begin{aligned}
\theta \vDash \{&cy' + \mathbf{S} + dx'' + h(x'') \overset{?}{=} 0, \\
&x \overset{?}{=} cx' + x'', y \overset{?}{=} y' + (-dx') + (h(-x'))\}
\end{aligned}
$$

then $\theta \vDash \{cy + \mathbf{S} + dx + h(x) \overset{?}{=} 0\}$.

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

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

So the statement is true for Factorization.

Annulization: We only need to show $\theta \vDash t \overset{?}{=} 0$ implies $\theta \vDash h(t) \overset{?}{=} 0$. This is true from our rewriting rules.

From the definition of Simplifier, we know the statement is true for Simplifiers. $\square$

Theorem 4.24 *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 4.23. □

From above theorem we have the following corollary:

Corollary 4.25 (Soundness) Let Γ be an AGh-unification problem. Suppose after applying the inferences rules from $\mathfrak{I}_{AGh}$ to $\Gamma\|\emptyset\|\emptyset\|\Psi^{ini}_{\Gamma}$ exhaustively, we get $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, i.e.

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

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.

4.10 Completeness

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

Lemma 4.26 *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 4.23, we know $\Gamma'\sigma \cup \Lambda\sigma$ is a conservative extension of

$$\Gamma \cup \{c_1x_1 + \cdots + c_nx_x + c_1't_1 + \cdots + c_m't_m \stackrel{?}{=} 0\} \cup \Lambda.$$

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 $\mathbf{S} + ch(t) + dh(s) \stackrel{?}{=} 0$, 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 $t\theta' \downarrow = \Sigma_{i=1}^{k} c_i t_i$ and $s\theta' \downarrow = \Sigma_{i=1}^{l} d_i s_i$, 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 $\theta'' = \theta' \cup \{x' \leftarrow c(-t) + d(-s)\}$, then $x'\theta'' = c(-t)\theta'' + d(-s)\theta'' = c\Sigma_{i=1}^{k} c_i t_i + d\Sigma_{i=1}^{l} d_i s_i$. So for every i, we have $|cc_i| < |cN_1|$ and $|dd_i| < |dN_2|$. So for $x'\theta''\Sigma_{i=1}^{l} d'r_i$, the largest absolute value of the coefficient is at most $cN_1 + dN_2$, which means θ'' satisfies Ψ.

And because if $\theta' \vDash \mathbf{S} + ch(t) + dh(s) \stackrel{?}{=} 0$, then $\theta'' \vDash \{\mathbf{S} + h(x') \stackrel{?}{=} 0, x' + (-ct) + (-ds) \stackrel{?}{=}$

$0\}$, 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 $s\theta = \Sigma_{i=i}^{k} c_i s_i$, $c_i < N$ 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. $\square$

Lemma 4.27 *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

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

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

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

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

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

and $\theta\delta$ satisfies Ψ'.

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

Then

$(cy' + \mathbf{S} + dx'' + h(x''))\theta\delta$

$= c(y + dx' + h(x'))\theta\delta + \mathbf{S}\theta\delta + d(x + (-cx'))\theta\delta + h(x + (-cx'))\theta\delta$

$= cy\theta\delta + cdx'\theta\delta + ch(x')\theta\delta + \mathbf{S}\theta\delta + dx\theta\delta + (-cdx')\theta\delta + h(x)\theta\delta + h(-cx')\theta\delta$

$= cy\theta\delta + \mathbf{S}\theta\delta + h(x)\theta\delta$

$= 0$

$(cx' + x'')\theta\delta$

$= cx'\theta\delta + x''\theta\delta$

$= cx'\theta\delta + x\theta\delta + (-cx')\theta\delta$

$= x\theta\delta$

$= x\theta$

$(y' + (-dx') + h(-x'))\theta\delta$

$= y'\theta\delta + (-dx')\theta\delta + h(-x')\theta\delta$

$= y\theta\delta + dx'\theta\delta + h(x')\theta\delta + (-dx')\theta\delta + h(-x')\theta\delta$

$= y\theta\delta$

$= y\theta$

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

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

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

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

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 4.28 *If t is reduced with respect to $\mathfrak{R}_{AGh}$, then* $\mathbf{Lay}(t)$ *has the following value:*

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

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

Lemma 4.29 *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 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 $c_{y_i}y_i + c_{i1}x_{i1} + c_{i2}x_{i2} + \cdots + c_{im}x_{im} + h(t) \overset{?}{=} 0$, where x_{ij} is not unconstrained, $(c, t) \in \Psi$ and $n \leq c$.

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 $h(t)$, $(h(t)\theta)\downarrow \neq 0$.

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

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

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 $cy_i+c_{i1}x_{i1}+c_{i2}x_{i2}+\cdots+c_{im}x_{im}+h(t) \overset{?}{=} 0$, where $(h(t)\theta)\downarrow\neq 0$, we have:

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

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

So $Lay(T_1)$ is not zero. Since x_1 occurs in t in the equation $cy_i + c_{i1}x_{i1} + c_{i2}x_{i2} + \cdots + c_{im}x_{im} + h(t) \overset{?}{=} 0$, 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 $Lay(T_l) = Lay(kh(t)\theta + T) \geq Lay(kh(t)\theta) \geq Lay(h(t)\theta) > Lay(x_1\theta) = Lay(T_1)$, which is a contradiction. If $y_i\theta$ can cancel $h(t)\theta$, then the case is as same as above.

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

Lemma 4.30 *Let $\Gamma\|\Delta\|\Lambda\|\Psi$ be a set quadruple. If there exists another set quadru-*

ple $\Gamma'\|\Delta'\|\Lambda'\|\Psi'$*, 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:

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

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 $s_1 \stackrel{?}{=} t_1, \cdots, s_m \stackrel{?}{=} t_m$. Since $f(s_1, s_2, \cdots, s_m)\theta = f(t_1, t_2, \cdots, t_m)\theta$, θ is a solution of $s_1 \stackrel{?}{=} t_1, \cdots, s_m \stackrel{?}{=} t_m$. 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 $\sigma \vDash \mathbf{S} + cf(s_1, s_2, \cdots, s_m) + (-df(t_1, t_2, \cdots, t_m)) \stackrel{?}{=} 0$ and $\sigma \vDash f(s_1, s_2, \cdots, s_m) = f(t_1, t_2, \cdots, t_m)$, we can get $\sigma \vDash \mathbf{S} + (c-d)f(s_1, s_2, \cdots, s_m) \stackrel{?}{=} 0$. So $\sigma \vDash (\mathbf{S} + (c-d)f(s_1, s_2, \cdots, s_m) \stackrel{?}{=} 0)\sigma$. 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 4.26. 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

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

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

We apply N-Decomposition(the second choice) and add $f(s_1, s_2, \cdots, s_m) +$

$(-f(t_1, t_2, \cdots, t_m)) \neq^? 0$ into the disequation set. It is trivially true that

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

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 4.31 *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:

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

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 $\{x_1 \leftarrow T_1, x_2 \leftarrow T_2, \cdots x_n \leftarrow T_n, y_1 \leftarrow S_1, y_2 \leftarrow S_2, \cdots, y_l \leftarrow S_l\}$, 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

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

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 $\mathbf{S} + ct \overset{?}{=} 0$ 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 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 $x\theta = \Sigma_{k=1}^{m} d_k s_k$ 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. $\square$

Then by combining Lemma 4.26, Lemma 4.27, Lemma 4.29, Lemma 4.30 and Lemma 4.31 we get:

Lemma 4.32 *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* $\Gamma\|\Delta\|\Lambda\|\Psi \Rightarrow_{\Im_{AGh}} \bigvee_i(\Gamma'_i\|\Delta'_i\|\Lambda'_i\|\Psi'_i)$, *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 4.32, we get:

Theorem 4.33 *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* $\Gamma\|\Delta\|\Lambda\|\Psi \overset{+}{\Rightarrow}_{\Im_{AGh}} \bigvee_i(\Gamma'_i\|\Delta'_i\|\Lambda'_i\|\Psi'_i)$, *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 4.34 (Completeness) Let Γ be an AGh-unification problem. Suppose after applying the inferences rules from $\Im_{AGh}$ to $\Gamma\|\emptyset\|\emptyset\|\Psi^{ini}_{\Gamma}$ exhaustively, we get $\bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$, i.e.

$$\Gamma\|\emptyset\|\emptyset\|\Psi^{ini}_{\Gamma} \overset{*}{\Rightarrow}_{\Im_{AGh}} \bigvee_i(\Gamma_i\|\Delta_i\|\Lambda_i\|\Psi_i)$$

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 $\sigma \leq_{AGh} \delta|_{Vars(\Gamma)}$.

4.11 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

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

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 4.35 The equation:

$$2f(x) + 3f(y) + (-3f(a)) \stackrel{?}{=} 0$$

has no solution.

Clash-Annul

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

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

Example 4.36 For the equation:

$$2f(x) + (-2f(y)) + (-2f(0)) + h(x) \stackrel{?}{=} 0$$

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

Occur Check

$$\frac{(\Gamma \cup \{c_1x_1 + c_2x_2 + \cdots + c_nx_n + c'_1t_1 + c'_2t_2 + \cdots + c'_mt_m \overset{?}{=} 0\})\|\Delta\|\Lambda\|\Psi}{\textbf{Fail}}.$$

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 4.37 The equation $x + 2f(x) \overset{?}{=} 0$ has no solution.

Occur Check-Annul

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

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

Example 4.38 We have equation set $\{x + h(x) \overset{?}{=} 0, f(x) + (-f(0)) + f(y) + (-f(z)) \overset{?}{=} 0\}$. We use Occur Check-Annul to get $x \leftarrow 0$, and then get the solution $[x \leftarrow 0, y \leftarrow z]$. This rule can avoid applying N-Decomposition too early.

Decomposition -II

If all the pure variables x_i in $\mathbf{S}$ 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 $\mathbf{S}$ is f, then:

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

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

Decomposition -III

If all the pure variables x_i in $\mathbf{S}$ 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 $\mathbf{S}$ is f, then:

$$\frac{(\Gamma \cup \{\mathbf{S} + c_1f(s_1, \cdots, s_m) + c_2f(t_1, \cdots, t_m) + c_3f(r_1, \cdots, r_m) \overset{?}{=} 0\})\|\Delta\|\Lambda}{(\Gamma_1\sigma)\|(\Delta\sigma)\|(\Lambda\sigma \cup \{[\sigma]\})}$$

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

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

Lemma 4.39 *Suppose* $\mathbf{S}$ *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*

$$\mathbf{S} + c_1f(s_{11}, s_{12}, \cdots, s_{1n}) + c_2f(s_{21}, s_{22}, \cdots s_{2n}) + \cdots + c_mf(s_{m1}, s_{m2}, \cdots s_{mn}) \overset{?}{=} 0$$

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

Proof. If there is a ground solution σ for this equation, after applying the substitution, we get

$$\mathbf{S}' + c_1f(s'_{11}, s'_{12}, \cdots, s'_{1n}) +$$
$$c_2f(s'_{21}, s'_{22}, \cdots s'_{2n}) + \cdots + c_mf(s'_{m1}, s'_{m2}, \cdots s'_{mn}) \overset{?}{=} 0$$

Since no pure variable occurs in $\mathbf{S}$, no f occurs as top symbol in $\mathbf{S}'$.

Next, we use induction on m to prove the lemma.

Suppose $m = 1$ and $c_1 \neq 0$. So σ satisfies $(S\sigma + c_1 f(s_1, s_2, \cdots s_n)\sigma) \downarrow = 0$. But from our rewriting rules, the only way to cancel an uninterpreted function symbol is to use an inverse term, which is $-c_1 f(s_1, s_2, \cdots s_n)\sigma$. In $\mathbf{S}$, no other f occurs on the top, so it is impossible that $(S\sigma + c_1 f(s_1, s_2, \cdots s_n)\sigma) \downarrow = 0$.

Suppose for any k, $k < m$, and $\Sigma_{i=1}^{k} c_i \neq 0$, the lemma is true. If $k = m$, then as we analyzed above, we need another term with f occurring on the top. These terms can only have the form

$$c'_1 f(s'_{11}, s'_{12}, \cdots, s'_{1n}) + c'_2 f(s'_{21}, s'_{22}, \cdots s'_{2n})$$
$$+ \cdots + c'_{m-1} f(s'_{(m-1),1}, s'_{(m-1)2}, \cdots s'_{(m-1)n})$$

And we should have $\Sigma_{i=1}^{m-1} c'_i = -c_m$. So our problem becomes whether the leftover terms

$$(c_1 - c'_1) f(s'_{11}, s'_{12}, \cdots, s'_{1n}) + (c_2 - c'_2) f(s'_{21}, s'_{22}, \cdots s'_{2n})$$
$$+ \cdots + (c_{m-1} - c'_{m-1}) f(s'_{(m-1),1}, s'_{(m-1)2}, \cdots s'_{(m-1)n})$$

can cancel each other. But $\Sigma_{i=1}^{m-1}(c_i - c'_i) \neq 0$ because $\Sigma_{i=1}^{m} c_i \neq 0$ and $\Sigma_{i=1}^{m-1} c'_i = -c_m$. Hence no solution exists. □

Theorem 4.40 *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 4.26, we proved that the purification procedure will preserve Ψ. So for proving the auxiliary rules are E-equation preservative, 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 4.39 we know that Clash is E-equation preservative.

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

$$\theta = [x_1 \leftarrow s_1, x_2 \leftarrow s_2, \cdots x_n \leftarrow s_n, y_1 \leftarrow s'_1, \cdots, y_k \leftarrow s'_k]$$

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 $\Sigma_{i=1}^{l} c'_i \neq 0$ from the condition of Occur Check. Because $\Sigma_{i=1}^{l} c'_i \neq 0$, from the proof of Lemma 4.39, 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 symbols 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 preservative.

For Occur Check-Annul, if there is a solution θ, then $cx\theta + \mathbf{S}\theta + h(t\theta) = 0$. Because $cx + \mathbf{S} + h(t\theta) \overset{?}{=} 0$ 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, $t \overset{?}{=} 0$ will fail from Lemma 4.39. 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 cancelled. 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. □

4.12 Implementation

Due to the complication of the combination algorithm [54], as far as we know, there is no public implementation for the general AGh-unification algorithm yet.

Thanks to Maude [1], we have implemented our algorithm in an intuitive way. The implemented program has resolved many AGh-unification problems. The running environment is on Windows 7, Intel Core2 Duo 2.26GHz, and RAM 5 GB using Maude 2.6. Table 4.1 shows some of our results.

In Table 4.1, “YES” means the algorithm outputs the minimal complete set of unifiers for the unification problem. As we can see in the table, in all of the cases, we got the minimal complete set of unifiers. 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 results are very satisfactory.

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

Table 4.1: Experiment Result for AGh Unification Algorithm

4.13 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 homomorphism as given in Chapter 3. 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. 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.

Chapter 5

General Asymmetric XOR-Unification Algorithm

5.1 Introduction

The Maude-NPA [24] is a cryptographic protocols analysis tool which was designed and implemented by Meadows [41]. It is one of the protocol verification tools using formal methods. It uses strand spaces [60, 61] to express the runs of protocols and the backward search strategy to search for attacks. Maude-NPA can also handle the algebraic properties of cryptographic primitives. One of the ways for handling the algebraic properties is calling the corresponding E-unification algorithm for a theory E if the algorithm exists.

For improving the search efficiency, the authors adopt many techniques for automatically identifying unreachable and redundant states such that it can prune useless branches as early as possible. This can improve the efficiency very much. These techniques includes the generating of formal grammars [24, 42, 23], early detection of inconsistent states etc. Those techniques are very useful in reducing the search space in Maude-NPA. Some of them have common syntax and semantics such that they can be adapted into other tools [25, 56, 16, 7].

But these techniques may be incompatible with the unification methods. Here we want to talk about the incompatibility between early detection of inconsistent states and the E-unification algorithms.

Inconsistent states are the states which are not supposed to be there in real runs of the protocol. For example [22], in the following running protocol, Bob wants to talk to Alice by sending his own nonce N_B encrypted by Alice's public key and Alice sends her own nonce N_A encrypted by Bob's public key to Bob upon receiving his request. Finally, Bob will send Alice back $N_A \oplus N_B$ to prove he is Bob by showing he knows N_A already. Here they use the property of exclusive OR $N_A \oplus N_B \oplus N_A \approx N_B$.

Example 5.1

1. $B \to A$: $\{N_B\}^p_{K_A}$
2. $A \to B$: $\{N_A\}^p_{K_B}$
3. $B \to A$: $N_A \oplus N_B$

So from Alice's perspective, informally, her strand space is like:

$$\{-\{X\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus X)\}$$

If we find an instance of the protocol by applying the substitution $X \leftarrow N_A \oplus Y$, we will get

$$\{-\{N_A \oplus Y\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus N_A \oplus Y)\}$$

Here we can see N_A is supposed to be generated when A sends out $\{N_A\}^p_{K_B}$, But N_A appears before A sends out $\{N_A\}^p_{K_B}$. So if any state containing above strand is an inconsistent state. Maude-NPA will discard state to avoid redundant search.

However, if we further instantiate Y by $Y \leftarrow N_A \oplus N_B$, then we can get:

$$\{-\{N_A \oplus N_A \oplus N_B\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus N_A \oplus N_A \oplus B)\}$$

which is:

$$\{-\{N_B\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus B)\}$$

We can see any state containing this strand is a legal execution of the protocol. This means discarding inconsistent states may prune a legal branch, which makes the search algorithm incomplete.

Hence, though standard exclusive OR unification algorithm (here we call the traditional unification algorithm like we introduced in Chapters 3 and 4 a *standard one*) gives a small complete set of unifiers, it hinders the optimization of detecting inconsistent states.

Thanks to the finite variant properties of exclusive OR and Abelian groups [14], Maude-NPA can overcome this problem in the following way(let us take exclusive OR as an example):

The key idea [22] of removing the inconsistent state without losing the completeness of the search algorithm is to divide $N_A \oplus X$ into two possible forms $N_A \oplus X$ and Z (it has four forms according to the formal definition, here, for simplicity, we use two of them to explain the idea). Then we can assume neither of them are reducible. The state becomes two independent states:

one contains the following strand:

$$\{-\{X\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus X)\}$$

with the substitution *id* and the other one contains the following strand:

$$\{-\{N_A \oplus Y\}^p_{K_A}, +\{N_A\}^p_{K_B}, -(N_A \oplus N_A \oplus Y)\}$$

with the substitution $X \leftarrow N_A \oplus Y$.

The second state is an inconsistent state, and Maude-NPA can safely discard it, because in the future, we do not allow any substitution to make Y contains N_A which will cancel N_A in $N_A \oplus Y$. So Maude-NPA can only focus on the first state to continue the search.

As we discussed in Chapter 1, we need a unification algorithm to unify the states and transition rules. But here we need the states to be irreducible for any substitution. So the problem becomes finding a complete set of unifiers U for E-unification problem $t \overset{?}{=}_E t'$, where t is a term from some transition rule and t' is a term in some state, such that for every $\sigma \in U$, $t\sigma =_E' t'\sigma$ and $t'\sigma$ is irreducible by R modulo theory E', where (R, E') is the decomposition of E.

Without using the standard exclusive OR unification algorithm, Maude-NPA has its own way to overcome this difficulty, which is called *folding variant narrowing* [27].

Though this technique can overcome the problem of detecting inconsistent states, it still has some disadvantage:

- Computing E-variants [14] is not very efficient. This is totally based on narrowing [6], which is highly non-deterministic.

- The complete set of unifiers directly got by this method is normally not a small set. There are two reasons: the first one is the complete set of variants generated by narrowing may be much larger than the minimal complete set of variants of the term in the right side [27]; the second one is $\mathcal{AC}$ unification problems may have a doubly exponential unifiers [31].

Though we have to compute the minimal complete set of E-variants for the terms to distinguish the inconsistent states, can we avoid computing the variants of the terms from transitions rules? Also in Chapters 3 and 4, we already have two efficient algorithms for standard unification algorithm for exclusive OR (with homomorphism) and Abelian groups (with homomorphism), can we adapt these algorithms to find asymmetric unifiers?

In the following two chapters, 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.

In this chapter, we have developed a sound, complete and terminating set of inference rules for asymmetric exclusive OR unification problems along with uninterpreted function symbols. We have implemented our inference rules in Maude [13] and they are being incorporated into the NRL protocol analyzer [24] with encouraging preliminary result. We have designed them in such a way that they can be extended to Abelian groups.

Generally, we start from a standard unifier of the unification problem (we can get this by using the algorithm presented in Chapter 3), and search for corresponding asymmetric unifiers fulfilling the constraints from the asymmetric unification problem.

For example, suppose we have the problem $\{a \oplus y \oplus z =_{\downarrow} x \oplus a, a \oplus z =_{\downarrow} z \oplus a\}$ and we got a standard unifier $\sigma = [x \leftarrow a \oplus b \oplus y \oplus z]$. In this case, since $x \leftarrow a \oplus b \oplus y \oplus z$ will make $x \oplus a$ reducible, we need another unifier to fit this constraint. The idea is to non-deterministically try $y \leftarrow a \oplus v$ and $z \leftarrow a \oplus v'$, where v and v' are fresh variables. We can see here $\sigma_1 = [y \leftarrow a \oplus v, x \leftarrow v \oplus b \oplus z]$ is an asymmetric unifier but $\sigma_2 = [z \leftarrow a \oplus v', x \leftarrow v' \oplus b \oplus y]$ is not an asymmetric unifier because it does not fulfill the constraint which is that $z \oplus a$ should be irreducible. Since σ_1, σ_2 are both equivalent to σ modulo exclusive OR, we can just chose σ_1 as the final asymmetric unifier.

If there is no equivalent asymmetric unifier, we use inference rules *Useless Branching, Decomposition Instantiation* and *Elimination Instantiation* to help us find all the possible instances. Formally, we give six rules which are easy to implement and efficient. For generating a smaller complete set of unifiers and avoiding unnecessary non-deterministic steps, we also give several auxiliary rules which can be used to increase the efficiency without losing soundness, completeness and termination.

Here we give the outline of this chapter. In Section 2, we give preliminary knowledge about the exclusive OR theory related to asymmetric unification problems. In Section 3, we present the problem format and the meaning of each part in the problem. Next, we present our inference system in three parts: 1.a preprocessing step (Splitting) that transforms a standard unifier into an equivalent unifier which only contains support variables as pure terms in the range is given in Section 4.1; 2. the inference rules which are used to search for equivalent asymmetric unifiers are given in Section 4.2; 3. Section 4.3 presents the inference rules which collect all the possible instances. The algorithm which finds the asymmetric unifiers using the inference rules is given in Section 5. The proofs of termination, soundness and completeness of our inference system are given in Sections 6 and 7 respectively. Section 8 gives some auxiliary rules. Section 9 shows some of our implementation results. The conclusion is given in Section 10.

5.2 Preliminaries

The exclusive OR theory(XOR) has the property of $\mathcal{ACUN}$, namely:

- $x \oplus y \approx y \oplus x$ [Commutativity];
- $x \oplus (y \oplus z) \approx (x \oplus y) \oplus z$ [Associativity];
- $x \oplus 0 \approx x$ [Unity];
- $x \oplus x \approx 0$ [Nilpotent].

We divide XOR to E and R, where E is $\mathcal{AC}$, namely:

- $x \oplus y \approx y \oplus x$ [commutativity];
- $x \oplus (y \oplus z) \approx (x \oplus y) \oplus z$ [associativity].

and R a convergent rewriting system of $\mathcal{UN}$, namely

- $x \oplus 0 \to x$;
- $x \oplus x \to 0$;
- $x \oplus y \oplus x \to y$.

For a unifier σ of Γ, we have the following notation:

- We call the variables which are in the unification problem Γ *original variables* and the variables, which are not in the unification problems but in the range of σ, *support variables*.
- We call a term S a *sum* term if its normal form has the form $s_1 \oplus \cdots \oplus s_n$, where $n > 1$. Otherwise, we call it a *simple* term. If a simple term occurs as a non-sub-term in an assignment, substitution or unification problem, we call it is a *top* term in that assignment, substitution or unification problem.
- $\mathit{Simple}(\sigma) = \{t \mid x \leftarrow t \oplus T \in \sigma \text{ and } t \neq 0\}$.

Because for a substitution σ, the assignments of support variables in σ will not affect the result of $\Gamma\sigma$. So for convenience, in our examples in the following sections, we will omit the assignments for support variables in σ unless we want to emphasize the assignments for support variables.

Example 5.2 If we have :

$$\Gamma = \{ \ x \oplus f(x \oplus y \oplus z) =_{\downarrow} 0 \ \}$$
$$\delta = [\ x \leftarrow f(v), y \leftarrow z \oplus v \oplus f(v) \ ]$$
$$\sigma = [\ z \leftarrow a, v \leftarrow b \ ]$$

Then

$$\delta\sigma = \{x \leftarrow f(b), y \leftarrow a \oplus b \oplus f(b), z \leftarrow a\}$$

and

$$\delta \circ \sigma = \{x \leftarrow f(b), y \leftarrow a \oplus b \oplus f(b), z \leftarrow a, \mathbf{v} \leftarrow \mathbf{b}\}$$

We will rewrite every term to its normal form if we do not indicate. In the following, when we compare two terms, we will compare them modulo $\mathcal{AC}$ if we do not indicate. For convenience, we will omit XOR and Γ if they are clear. Normally, we will use lower case letters s, t to denote simple terms, and capital letters S, T to denote terms which may be a simple term, sum term or identity.

5.3 Problem Format

We introduce a set of inference rules to transform members of a complete set of standard XOR unifiers to a complete set of asymmetric XOR unifiers. Our inference rules involves triples and have the following form:

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma' \| \Upsilon' \| \Delta'}$$

Here σ is a standard XOR unifier of Γ. Υ is called a *constraint set*, which is a set of pairs in which each member has the form (v, s), where v is a variable and s is a simple term or a variable. We also call (v, s) a constraint pair. Δ is called a *disequation* set, in which every member has the form $s \oplus t \neq^? 0$, where s and t have the same uninterpreted function symbol at the top.

If $\Upsilon = \{(v_1, s_1), \cdots, (v_n, s_n)\}$, then the result of applying a substitution θ to Υ is $\Upsilon\theta = \{(v_i, s_i\theta\downarrow) \mid (v_i, s_i) \in \Upsilon\}$. If $s_i\theta\downarrow$ is not simple but of the form $t_1 \oplus \cdots \oplus t_n$, we will rewrite $(v_i, t_1 \oplus \cdots \oplus t_n)$ to $(v_i, t_1), \cdots, (v_i, t_n)$.

For convenience, we have the following notation:

- $\Upsilon \cup \{(v, s)\}$ is abbreviated by $\Upsilon \cup (v, s)$
- $\Upsilon[v'/v]$ means the set $\{(y, t) \in \Upsilon \mid y \neq v\} \cup \{(v', t) \mid (v, t) \in \Upsilon\}$
- $\Upsilon[v/(v_1, v_2)]$ means the set $\{(y, t) \in \Upsilon \mid y \neq v_1, y \neq v_2\} \cup \{(v, t) \mid (v_1, t) \in \Upsilon\} \cup \{(v, t) \mid (v_2, t) \in \Upsilon\}$.

Definition 5.3 (Violation, Satisfaction) *Let δ be a substitution and (v, s) a constraint pair. If $v\delta\downarrow \oplus s\delta\downarrow$ is irreducible, then we say δ* ***violates*** *the pair (v, s). Otherwise, we say δ* ***satisfies*** *Υ. If δ violates at least one pair in Υ, we say δ* ***violates*** *Υ. Otherwise, we say δ* ***satisfies*** *Υ.*

Example 5.4 Let

$$\begin{aligned} \delta_1 &= [x \leftarrow v \oplus a \oplus b, y \leftarrow v \oplus c] \\ \delta_2 &= [x \leftarrow c \oplus b, y \leftarrow a] \\ \Upsilon &= \{(x, a), (x, y), (y, b)\}. \end{aligned}$$

Here we can see δ_1 violates (x, a) and (x, y), and satisfies (y, b). δ_2 satisfies each pair in Υ. So δ_1 violates Υ but δ_2 satisfies Υ.

Similarly, we have

Definition 5.5 (Violation, Satisfaction) *Let δ be a substitution and $s \oplus t \neq^? 0$ a disequation. If $(s\delta \oplus t\delta) \downarrow = 0$, then we say δ* ***violates*** *the disequation $s \oplus t \neq^? 0$. Otherwise, we say δ* ***satisfies*** *Δ. If δ violates at least one disequation in Δ, we say δ* ***violates*** *Δ. Otherwise, we say δ* ***satisfies*** *Δ.*

Example 5.6 Let

$$\begin{aligned}
\delta_1 &= [x \leftarrow v \oplus a \oplus b, y \leftarrow v \oplus c] \\
\delta_2 &= [x \leftarrow c \oplus b, y \leftarrow a] \\
\Delta &= \{f(x) \oplus f(a) \neq^? 0, f(y \oplus a) \oplus f(0) \neq^? 0\}.
\end{aligned}$$

We can see δ_1 satisfies every disequation in Δ. δ_2 satisfies $f(x) \oplus f(a) \neq^? 0$, but violates $f(y \oplus a) \oplus f(0) \neq^? 0$, because $f(y \oplus a)\delta_2 = f(0)$. So δ_1 satisfies Δ, but δ_2 violates Δ.

The triple $\sigma \| \Upsilon \| \Delta$ denote the following set of substitutions:

Definition 5.7 (Instance of $(\Gamma, \sigma, \Upsilon, \Delta)$)

$$\begin{aligned}
\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \quad & \{\delta \mid \delta \textit{ is an asymmetric XOR unifier of } \Gamma; \\
& \textit{and there exists } \theta \textit{ such that } \sigma\theta = \delta; \\
& \textit{and } \sigma \circ \theta \textit{ satisfies } \Upsilon\theta; \\
& \textit{and } \sigma \circ \theta \textit{ satisifes } \Delta\theta.\}
\end{aligned}$$

We will use **Failure** as a special triple, such that $\mathfrak{Inst}(\textbf{Failure}) = \emptyset$.

Example 5.8 Let

$$\begin{aligned}
\Gamma &= \{a \oplus b \oplus c \oplus z =_{\downarrow} x \oplus y\},\\
\sigma &= [x \leftarrow v \oplus a \oplus b, y \leftarrow v \oplus c, z \leftarrow 0],\\
\Upsilon &= \{(v, a)\},\\
\Delta &= \{f(v \oplus c) \oplus f(0) \neq^{?} 0, f(v \oplus a) \oplus f(0) \neq^{?} 0\},\\
\delta_1 &= [x \leftarrow b \oplus c, y \leftarrow a, z \leftarrow 0],\\
\delta_2 &= [x \leftarrow b, y \leftarrow a \oplus c, z \leftarrow 0],\\
\delta_3 &= [x \leftarrow a \oplus c, y \leftarrow b, z \leftarrow 0].
\end{aligned}$$

For the triple $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, we see that

- $\sigma \notin \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$ because σ is not an asymmetric unifier of Γ.
- $\delta_1 \notin \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$ because $\delta_1 = \sigma[v \leftarrow a \oplus c]$ and $\sigma \circ [v \leftarrow a \oplus c]$ violates (v, a).
- $\delta_2 \notin \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$ because $\delta_2 = \sigma[v \leftarrow a]$ and $\sigma \circ [v \leftarrow a]$ will make $f(v \oplus b) \oplus f(0) = 0$.
- $\delta_3 \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$ because δ_3 is an asymmetric unifier of Γ, $\delta_3 = \sigma[v \leftarrow b \oplus c]$, $\sigma \circ [v \leftarrow b \oplus c]$ satisfies (v, a), $(f(v \oplus c) \oplus f(0))[v \leftarrow b \oplus c] = f(b) \oplus f(0) \neq^{?} 0$, and $f(v \oplus a) \oplus f(0)[v \leftarrow b \oplus c] = f(a \oplus b \oplus c) \oplus f(0) \neq^{?} 0$.

In the following, for a unification problem Γ and an XOR unifier σ, we say in the assignment $x \leftarrow t \oplus T \in \sigma$, some original variable x has a *conflict* at some simple term t, if:

- there exists $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus s] \in \Gamma$ and
- there exists T' such that $s\sigma = t \oplus T'$.

where s and t are simple terms and T' might be empty.

Example 5.9 $\sigma = [x \leftarrow f(a) \oplus y, z \leftarrow a]$, and $\Gamma = \{y =_\downarrow x \oplus f(z), z \oplus a =_\downarrow 0\}$.

x has a conflict at $f(a)$ since $x\sigma = f(a) \oplus y$ and $f(z)\sigma = f(a)$.

Example 5.10 $\sigma = [x \leftarrow v \oplus a, y \leftarrow v \oplus b]$, and $\Gamma = \{a \oplus b =_\downarrow x \oplus y\}$.

x has a conflict at v, since $x\sigma = v \oplus a$, and $v\sigma = v \oplus b$.

5.4 Inference System $\mathfrak{I}_{AXO}$

All of our inference rules are don't care nondeterministic rules and we divide our rules into three parts: **Splitting**, **Branching** and **Instantiation**. Some of the rules have the following form:

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma' \| \Upsilon' \| \Delta' \bigvee \sigma'' \| \Upsilon'' \| \Delta''}$$

Here $\bigvee$ means we divide one triple $\sigma \| \Upsilon \| \Delta$ into two triples $\sigma' \| \Upsilon' \| \Delta'$ and $\sigma'' \| \Upsilon'' \| \Delta''$. $\sigma' \| \Upsilon' \| \Delta'$ and $\sigma'' \| \Upsilon'' \| \Delta''$ are two independent problems with the same Γ.

5.4.1 Part I - Splitting

There will be only one rule in this part, which is called Splitting. This rule will transform an XOR unifier σ to an equivalent XOR unifier σ' such that all the top variables in $Range(\sigma')$ are support variables.

Splitting

Splitting

$$\frac{[x \leftarrow y \oplus S \oplus T] \cup \sigma \| \Upsilon \| \Delta}{([x \leftarrow y \oplus S \oplus T] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta}$$

where $\theta = \{y \leftarrow v \oplus S\}$ and v is a fresh support variable.

Conditions:

- $x, y \in Vars(\Gamma)$;
- $y \notin Vars(S)$.

Here, S and T can be chosen in any way we want since this rule is do not care nondeterministic. However, if x has a conflict at some simple term s in $S \oplus T$, then for efficiency in our implementation, we will put s into T, unless $y \in Vars(s)$.

Example 5.11 $\Gamma = \{y \oplus f(a) =_{\downarrow} x \oplus a\}$

$$\frac{[x \leftarrow y \oplus f(a) \oplus a] || \emptyset || \emptyset}{[x \leftarrow v \oplus f(a), y \leftarrow v \oplus a] || \emptyset || \emptyset}$$

Example 5.12 $\Gamma = \{x \oplus z =_{\downarrow} y \oplus a, x \oplus a =_{\downarrow} x \oplus a, x \oplus z =_{\downarrow} x \oplus z\}$. Let $\sigma = [x \leftarrow y \oplus z \oplus a]$

For convenience, we omit Υ and Δ here.

First, we apply Splitting and get

$$\sigma_1 = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus z]$$

Second, we apply Splitting again and get

$$\sigma_2 = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$

As we will see in the next section, Splitting will be applied first. After exhaustively applying Splitting, there will be no original variables in the range of σ. So from the next section, we will assume that all the top variables which appear in the range of σ are support variables.

5.4.2 Part II - Branching

There are three rules in this part. The main idea is to try to transform a unifier into an equivalent one without conflicts.

Non-Variable Branching

The first Branching inference rule is called *Non-Variable Branching*, which solves the case that some original variable x has a conflict at some non-variable simple term s by dividing the triple to two triples: the substitutions in the first triple assume some variable v can cancel s and the substitutions in the second triple assume v cannot cancel s

For example, if we have $x \leftarrow v_1 \oplus v_2 \oplus a$ in σ, but we also have $x \oplus a =_{\downarrow} x \oplus a$ in Γ, then x has a conflict at a. So our rule tries to see if v_1 can cancel a or v_2 can cancel a in two steps.

Formally:

Non-Variable Branching

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| (\Upsilon[v'/v] \cup (v', s))\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, s)\} \| \Delta\theta}$$

where there exists an assignment $[x \leftarrow v \oplus s \oplus S] \in \sigma$ and

$$\theta = [v \leftarrow v' \oplus s]$$

Conditions:

- s is a simple term but not a variable and $v \notin Vars(s)$;
- x has a conflict at s in σ;

- $(v, s) \notin \Upsilon$.

Example 5.13 We continue Example 6.8.

$\Gamma = \{x \oplus z =_{\downarrow} y \oplus a, x \oplus a =_{\downarrow} x \oplus a, x \oplus z =_{\downarrow} x \oplus z\}$

Let $\sigma = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$.

For convenience, we omit Δ here.

We apply Non-Variable Branching to $\sigma \| \Upsilon \| \Delta$ to solve the case that x has a conflict at a and get:

Branch 1:

$$\begin{aligned} \sigma_1 &= [x \leftarrow v_3, y \leftarrow v_3 \oplus a \oplus v_2, z \leftarrow v_2] \\ \Upsilon_1 &= \{(v_3, a)\} \end{aligned}$$

where $\theta = [v_1 \leftarrow v_3 \oplus a]$; and Branch 2

$$\begin{aligned} \sigma_2 &= [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2] \\ \Upsilon_2 &= \{(v_1, a)\} \end{aligned}$$

In Branch 1, y has a conflict at a and $(v_3, a) \in \Upsilon_1$, so apply Non-Variable Branching, let $v_2 \leftarrow v_4 \oplus a$ and get:

Branch 1.1

$$\sigma_{1.1} = [x \leftarrow v_3, y \leftarrow v_3 \oplus v_4, z \leftarrow v_4 \oplus a]$$
$$\Upsilon_{1.1} = \{(v_3, a), (v_4, a)\}$$

where $\theta = [v_2 \leftarrow v_4 \oplus a]$; and Branch 1.2

$$\sigma_{1.2} = [x \leftarrow v_3, y \leftarrow v_3 \oplus a \oplus v_2, z \leftarrow v_2]$$
$$\Upsilon_{1.2} = \{(v_3, a), (v_2, a)\}$$

We can see $\sigma_{1.1}$ an asymmetric unifier of Γ.

Here $\sigma_{1.1}$ is equivalent to σ. Non-Variable Branching will not apply to Branch 2 and 1.2 because the conditions for applying Non-Variable Branching are not satisfied.

Variable Branching

The next rule is called *Variable Branching*. It will be applied to the case that some original variable x has a conflict at some support variable v.

But it will not solve the conflict directly. It is preparing for the instantiation part.

For example, suppose we have $x \leftarrow v_1 \oplus v_2 \oplus v_3, y \leftarrow v_1$ in σ and $x \oplus y =_{\downarrow} x \oplus y$ in Γ. We will try to see whether v_2 or v_3 can cancel v_1. Since applying a substitution to a variable will turn the variable into a sum term, it is also possible that sum $v_1 \oplus v_3$ cancels v_1. So here, we will let $v_2 \leftarrow v_2' \oplus v$ and $v_1 \leftarrow v_1' \oplus v$.

Formally, this rule is:

Variable Branching

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| \Upsilon'\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v_1, v_2)\} \| \Delta}$$

where

$$\theta = [v_1 \leftarrow v_{12} \oplus v_1', v_2 \leftarrow v_{12} \oplus v_2']$$

There exist an assignment $[x \leftarrow v_1 \oplus v_2 \oplus S, y \leftarrow v_1 \oplus S']$ in σ

$$\begin{aligned}\Upsilon' = \Upsilon[v_{12}/(v_1, v_2)] \cup \Upsilon[v_1'/v_1] \cup \Upsilon[v_2'/v_2] \cup \\ \{(v_{12}, v_1'), \quad (v_{12}, v_2'), \quad (v_1', v_2'), \\ (v_1', v_{12}), \quad (v_2', v_{12}), \quad (v_2', v_1')\}\end{aligned}$$

Here v_{12}, v_1' and v_2' are fresh support variables.

Conditions:

- There is an equation $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus y] \in \Gamma$, which means x has a conflict at v_1 in the premise;
- $(v_1, v_2) \notin \Upsilon$.

Note:

- Here in the first branch, we assume $v_1 \leftarrow v \oplus v_1'$ and $v_2 \leftarrow v \oplus v_2'$, which means in each instance δ of $\sigma\theta$, $v_1\delta$ and $v_2\delta$ should have the common part $v\delta$; at the same time $v_1'\delta$ and $v_2'\delta$ should not have any common part.

- In the second branch, we assume for any instance δ of σ, $v_1\delta$ and $v_2\delta$ have no common parts, so (v_1, v_2) and (v_2, v_1) are also added into Υ.
- Here we add constraints in both directions((v_1, v_2) and (v_2, v_1)). It is equivalent to only give the constrain in one direction.

Example 5.14

$$\begin{aligned}\Gamma = \{\ & x \oplus y \oplus z =_{\downarrow} a,\\ & x \oplus y =_{\downarrow} x \oplus y,\\ & z \oplus a =_{\downarrow} z \oplus a\ \}\\ \sigma = [&x \leftarrow y \oplus z \oplus a]\end{aligned}$$

After applying Splitting exhaustively, we can get

$$\sigma' = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$

After applying Variable Branching for the case that y has a conflict at v_1, by letting $v_1 \leftarrow v_1' \oplus v$ and $v_2 \leftarrow v_2' \oplus v$, we get

Branch 1:

$$\begin{aligned}\sigma_1 &= [x \leftarrow v_1' \oplus v \oplus a, y \leftarrow v_1' \oplus v_2', z \leftarrow v_2' \oplus v]\\ \Upsilon_1 &= \{(v_1', v), (v_1', v_2'), (v, v_1'), (v, v_2'), (v_2', v), (v_2', v_1')\}\end{aligned}$$

and Branch 2:

$$\sigma_2 = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$

$$\Upsilon_2 = \{(v_2, v_1), (v_1, v_2)\}$$

Note: We have not finished this problem yet. We will see the rest of this solution in the next section.

Useless Branching

The next rule is called *Useless Branching.* It will be applied to the case that some original variable x has a conflict at some support variable v. But similar to Variable Branching it will not solve the conflict directly, it is preparing for the instantiation part. The difference between Variable Branching and Useless Branching is in Variable Branching, we guess in some substitution θ there is another support variable v' such that $v'\theta \downarrow \oplus v\theta \downarrow$ is irreducible, but in Useless Branching, we guess there is another simple non-variable term t, such that $t\theta \downarrow \oplus v\theta \downarrow$ is irreducible.

For example if we have $x \leftarrow a \oplus v$ and $y \leftarrow v$ in σ and $x \oplus y =_{\downarrow} x \oplus y$ in Γ we will guess in some substitution θ, $v\theta \downarrow = a\theta \oplus S$ by letting $v \leftarrow v' \oplus a$.

Formally, the rule is:

Useless Branching

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| (\Upsilon[v'/v] \cup (v', s))\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, s)\} \| \Delta}$$

where

$$\theta = \{v \leftarrow v' \oplus s\}$$

There exist two assignments $[x \leftarrow v \oplus s \oplus S, y \leftarrow v \oplus S']$ in σ.

Conditions:

- s is a simple non-variable term;
- There is an equation $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus y] \in \Gamma$;
- $(v, s) \notin \Upsilon$.
- $v \notin Vars(s)$

Example 5.15 We continue Example 6.12

$$\begin{aligned}\Gamma = \{ & \; x \oplus y \oplus z =_{\downarrow} a, \\ & \; x \oplus y =_{\downarrow} x \oplus y, \\ & \; z \oplus a =_{\downarrow} z \oplus a \; \}\end{aligned}$$

Branch 1:

$$\sigma_1 = [x \leftarrow v_1' \oplus v \oplus a, y \leftarrow v_1' \oplus v_2', z \leftarrow v_2' \oplus v]$$

$$\Upsilon_1 = \{(v_1', v), (v_1', v_2'), (v, v_1'), (v, v_2'), (v_2', v), (v_2', v_1')\}$$

and Branch 2:

$$\sigma_2 = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$
$$\Upsilon_2 = \{(v_2, v_1), (v_1, v_2)\}$$

We can apply Useless Branching to Branch 1 because x has a conflict at v_1' and $(v_1', a) \notin \Upsilon$, after applying Useless Branching by letting $v_1' \leftarrow v_1'' \oplus a$ we get:

Branch 1.1

$$\sigma_{1.1} = [x \leftarrow v_1'' \oplus v, y \leftarrow v_1'' \oplus a \oplus v_2', z \leftarrow v_2' \oplus v]$$
$$\Upsilon_{1.1} = \{(v_1'', v), (v_1'', v_2'), (v, v_1''), (v, a), (v, v_2'), (v_2', v), (v_2', v_1''), (v_2', a), (v_1'', a)\}$$

and Branch 1.2

$$\sigma_{1.2} = [x \leftarrow v_1' \oplus v \oplus a, y \leftarrow v_1' \oplus v_2', z \leftarrow v_2' \oplus v]$$
$$\Upsilon_{1.2} = \{(v_1', v), (v_1', v_2'), (v, v_1'), (v, v_2'), (v_2', v), (v_2', v_1'), (v_1', a)\}$$

We cannot apply any Branching rules to Branch 1.1 and 1.2

Let look at Branch 2. We still can apply Useless Branching to Branch 2 since $(v_1, a) \notin \Upsilon_2$. Let $v_1 \leftarrow v_1''' \oplus a$ and get:

Branch 2.1:

$$\sigma_{2.1} = [x \leftarrow v_1''', y \leftarrow v_1''' \oplus a \oplus v_2, z \leftarrow v_2]$$

$$\Upsilon_{2.1} = \{(v_2, v_1'''), (v_2, a), (v_1''', v_2), (v_1''', a)\}$$

and Branch 2.2:

$$\sigma_{2.2} = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$

$$\Upsilon_{2.2} = \{(v_2, v_1), (v_1, v_2), (v_1, a)\}$$

For Branch 1.1, 1.2, 2.1 and 2.2 we have no rules in the Branching part applicable.

5.4.3 Part III - Instantiation

The following rules are called Instantiation Rules. They are used for solving the conflicts by instantiating some support variables.

Once we get an instance, the result may violate Υ or Δ. If this is true, we will throw out this instantiation branch.

Decomposition Instantiation

The first instantiation rule is called Decomposition Instantiation. It is used to solve the case that some original variable x has a conflict at some uninterpreted function term t.

For example if $x \leftarrow f(a) \oplus f(y) \oplus a$ is in σ and $x \oplus f(y) =_\downarrow x \oplus f(y)$ is in Γ, we will unify $f(a)$ and $f(y)$, so that the conflict disappears.

Formally, the rule is:

Decomposition Instantiation

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta_1 \| \Upsilon\theta_1 \| \Delta\theta_1 \quad \bigvee \cdots \bigvee \quad \| \sigma \circ \theta_n \| \Upsilon\theta_n \| \Delta\theta_n \quad \bigvee \sigma \| \Upsilon \| \Delta''}$$

where

there exists an assignment $[x \leftarrow s \oplus t \oplus S]$ in σ. $\{\theta_1, \cdots, \theta_n\}$ is a complete set of XOR unifiers of $s \overset{?}{=} t$ and $\Delta'' = \Delta \cup \{s \oplus t \neq^? 0\}$.

Conditions:

- s and t are not variables and have the same uninterpreted function symbol on the top;
- x has a conflict at s in the premise.

Example 5.16

$$\Gamma = \{f(a) \oplus f(b) =_\downarrow x \oplus f(y)\}$$
$$\sigma = [x \leftarrow f(y) \oplus f(a) \oplus f(b)]$$

Since x has a conflict at $f(y)$, we apply Decomposition Instantiation by unifying $f(a)$ and $f(y)$. Then get:

Branch 1:

$$\sigma_1 = [x \leftarrow f(b), y \leftarrow a]$$
$$\Delta_1 = \emptyset$$

and Branch 2:

$$\sigma_2 = [x \leftarrow f(y) \oplus f(a) \oplus f(b)]$$
$$\Delta_2 = \{f(y) \oplus f(a) \neq^? 0\}$$

σ_1 is an asymmetric unifier of Γ. We keep it and look at Branch 2.

In Branch 2, we still can apply Decomposition Instantiation since x has a conflict at $f(y)$, and $f(b) \oplus f(y) \neq^? 0 \notin \Delta$. We unify $f(y)$ and $f(b)$ and get:

Branch 2.1

$$\sigma_{2.1} = [x \leftarrow f(a), y \leftarrow b]$$
$$\Delta_{2.1} = \{f(y) \oplus f(a) \neq^? 0\}$$

and Branch 2.2

$$\sigma_{2.2} = [x \leftarrow f(y) \oplus f(a) \oplus f(b)]$$
$$\Delta_{2.2} = \{f(y) \oplus f(a) \neq^? 0, f(y) \oplus f(b) \neq^? 0\}$$

$\sigma_{2.1}$ is another asymmetric unifier. $\sigma_{2.2}$ is not an asymmetric unifier. however, we cannot apply any rules so far to Branch 2.2.

Elimination Instantiation

The second instantiation rule is called Elimination Instantiation. It is used to solve the case that some original variable x has a conflict at some support variable v.

For example if $x \leftarrow v \oplus b, y \leftarrow v \oplus d$ is in σ, $x \oplus y =_\downarrow x \oplus y$ is in Γ, and $(v, a), (v, d) \in \Upsilon$, we will let $v \leftarrow 0$, so that the conflict disappears.

formally, the rule is:

Elimination Instantiation

$$\frac{[x \leftarrow v \oplus S] \cup \sigma \| \Upsilon \| \Delta}{([x \leftarrow S] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta}$$

where

$$\theta = \{v \leftarrow 0\}$$

Conditions:

- There is an equation $\mathbf{u} =_\downarrow \mathbf{v}[x \oplus y] \in \Gamma$, such that $y\sigma = v \oplus S'$ for some S'.

Note: Since we let $v \leftarrow 0$, all the pairs of the form (v, s) in Υ will be removed from Υ.

Example 5.17 We continue Example 6.13

$$\Gamma = \{ \ x \oplus y \oplus z =_\downarrow a, \quad x \oplus y =_\downarrow x \oplus y, \quad z \oplus a =_\downarrow z \oplus a \ \}$$

We get:

Branch 1.1

$$\sigma_{1.1} = [x \leftarrow v_1'' \oplus v, y \leftarrow v_1'' \oplus a \oplus v_2', z \leftarrow v_2' \oplus v]$$

$$\Upsilon_{1.1} = \{(v_1'', v), (v_1'', v_2'), (v, v_1''), (v, a), (v, v_2'), (v_2', v), (v_2', v_1''), (v_2', a), (v_1'', a)\}$$

Branch 1.2

$$\sigma_{1.2} = [x \leftarrow v_1' \oplus v \oplus a, y \leftarrow v_1' \oplus v_2', z \leftarrow v_2' \oplus v]$$

$$\Upsilon_{1.2} = \{(v_1', v), (v_1', v_2'), (v, v_1'), (v, v_2'), (v_2', v), (v_2', v_1'), (v_1', a)\}$$

Branch 2.1:

$$\sigma_{2.1} = [x \leftarrow v_1''', y \leftarrow v_1''' \oplus a \oplus v_2, z \leftarrow v_2]$$

$$\Upsilon_{2.1} = \{(v_2, v_1'''), (v_2, a), (v_1''', v_2), (v_1''', a)\}$$

and Branch 2.2:

$$\sigma_{2.2} = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$

$$\Upsilon_{2.2} = \{(v_2, v_1), (v_1, v_2), (v_1, a)\}$$

For each branch, we have no rules except Elimination Instantiation applicable. In Branch 1.1, x has a conflict at v_1'' in $x \leftarrow v_1'' \oplus v$, so we let $v_1'' \leftarrow 0$ and get Branch 1.1.1:

$$\sigma_{1.1.1} = [x \leftarrow v, y \leftarrow a \oplus v_2', z \leftarrow v_2' \oplus v]$$
$$\Upsilon_{1.1.1} = \{(v, a), (v, v_2'), (v_2', v), (v_2', a)\}$$

Here $\sigma_{1.1.1}$ is an asymmetric unifier.

In Branch 1.2, x has a conflict at v_1' in $x \leftarrow v_1' \oplus v \oplus a$, so we let $v_1' \leftarrow 0$ and get:

$$\sigma_{1.2.1} = [x \leftarrow v \oplus a, y \leftarrow v_2', z \leftarrow v_2' \oplus v]$$
$$\Upsilon_{1.2.1} = \{(v, v_2'), (v_2', v)\}$$

Here $\sigma_{1.2.1}$ is an asymmetric unifier, which is equivalent to $\sigma_{1.1.1}$.

In Branch 2.1, x has a conflict at v_1''' in $x \leftarrow v_1'''$, so we let $v_1''' \leftarrow 0$ and get:

$$\sigma_{2.1.1} = [x \leftarrow 0, y \leftarrow a \oplus v_2, z \leftarrow v_2]$$
$$\Upsilon_{2.1.1} = \{(v_2, a)\}$$

Here $\sigma_{2.1.1}$ is not an asymmetric unifier.

In Branch 2.2, x has a conflict at v_1 in $x \leftarrow v_1 \oplus a$, so we let $v_1 \leftarrow 0$ and get:

$$\sigma_{2.2.1} = [x \leftarrow a, y \leftarrow v_2, z \leftarrow v_2]$$
$$\Upsilon_{2.2.1} = \emptyset$$

σ is an asymmetric unifier, which is an instance of $\sigma_{1.1.1}$.

5.5 Algorithm for Searching for the Complete Set of Asymmetric unifiers via $\mathfrak{I}_{AXO}$

Our inference system $\mathfrak{I}_{AXO}$ has three parts: Splitting, Branching and Instantiation.

First, we introduce three sub-algorithms, in which, the rules in our three parts will be applied separately. Then we will give an algorithm to show how to call three algorithms and output a complete set of asymmetric unifiers.

The first algorithm is called SPLITTING:

SPLITTING

`INPUT :`

- the unification problem Γ
- an XOR unifier σ;
- a constraint set Υ;
- a disequation set Δ.

`OUTPUT`

- a set of triple Σ, which contains all the results by applying rules in $\mathfrak{I}_{AXO}$.

`PROCEDURE:`

1. Check whether σ is an asymmetric unifier of Γ:
    - If it is, **return** $\{\sigma \| \Upsilon \| \Delta\}$.
    - If it is not, check whether the Splitting rule is applicable:
        - If it is, Apply Splitting to $\sigma \| \Upsilon \| \Delta$ and get $\sigma' \| \Upsilon' \| \Delta'$. **Return** SPLITTING$(\Gamma, \sigma', \Upsilon', \Delta')$.
        - If it is not, **return** BRANCHING$(\Gamma, \{\sigma \| \Upsilon \| \Delta\}, \emptyset)$

The second algorithm is called BRANCHING.

BRANCHING

`INPUT`:

- the unification problem Γ;
- a set of triples Θ which stores all the branches which are waiting for being applied Branching rules;
- a set of triples Σ which stores all the results after applying branching rules..

`OUTPUT`:

- a set of triple Σ, which represent the result of applying rules in Branching Part.

`PROCEDURE`:

1. Check whether Σ has only one member $\sigma\|\Upsilon\|\Delta$.
   - if yes and σ is an asymmetric unifier of Γ, **return** Σ.
2. Check whether Θ is empty.
   - If it is, **return** INSTANTIATION(Γ, Σ).
   - If it is not, pick up a member $\sigma\|\Upsilon\|\Delta$ from Θ.
     (a) If σ is an asymmetric unifier, return $\{\sigma\|\Upsilon\|\Delta\}$.
     (b) If Non-Variable Branching rule is applicable, apply Non-Variable Branching to $\sigma\|\Upsilon\|\Delta$ and get $\sigma'\|\Upsilon'\|\Delta'$ and $\sigma''\|\Upsilon''\|\Delta''$, **return** BRANCHING$(\Gamma, \Theta \cup \{\sigma'\|\Upsilon'\|\Delta', \sigma''\|\Upsilon''\|\Delta''\}, \Sigma)$. Else

(c) If Variable Branching rule is applicable, apply Variable Branching to $\sigma \| \Upsilon \| \Delta$ and get $\sigma' \| \Upsilon' \| \Delta'$ and $\sigma'' \| \Upsilon'' \| \Delta''$, **return** BRANCHING$(\Gamma, \Theta \cup \{\sigma' \| \Upsilon' \| \Delta', \sigma'' \| \Upsilon'' \| \Delta''\}, \Sigma)$. Else

(d) If Useless-Variable Branching rule is applicable, apply Useless-Variable Branching to $\sigma \| \Upsilon \| \Delta$ and get $\sigma' \| \Upsilon' \| \Delta'$ and $\sigma'' \| \Upsilon'' \| \Delta''$, **return** BRANCHING$(\Gamma, \Theta \cup \{\sigma' \| \Upsilon' \| \Delta', \sigma'' \| \Upsilon'' \| \Delta''\}, \Sigma)$. Else

(e) Add $\sigma \| \Upsilon \| \Delta$ to Σ, and **return** BRANCHING$(\Gamma, \Theta \setminus \{\sigma \| \Upsilon \| \Delta\}, \Sigma)$

Note: Here the reason why we call Branching as an argument to itself but we do not take the union of two Branching calls because if we find one asymmetric unifier, we will not check another branch since this one must be equivalent to the initial unifier.

The third algorithm is called INSTANTIATION:

INSTANTIATION

`INPUT`

- the unification problem Γ;
- a set of triples Σ.

`OUTPUT`:

- a set of triple Σ', which represent the result of applying rules in Instantiation Part.

`PROCEDURE`

Given an empty set of triple Σ', for each member $\sigma\|\Upsilon\|\Delta$ of Σ,

1. Check whether Decomposition Instantiation is applicable:
    - if it is, apply Decomposition Instantiation to $\sigma\|\Upsilon\|\Delta$ and get $\sigma_1\|\Upsilon_1\|\Delta_1, \cdots \sigma_n\|\Upsilon_n\|\Delta_n$. For each $\sigma_i\|\Upsilon_i\|\Delta_i$, check whether σ_i is an asymmetric unifier of Γ:
        - if it is an asymmetric unifier, add it to Σ' and go to next member.
        - if is not, union SPLITTING$(\Gamma, \sigma_i, \Upsilon_i, \Delta_i)$ with Σ'
2. Check whether Elimination Instantiation is applicable:
    - If it is applicable, apply Elimination Instantiation to $\sigma\|\Upsilon\|\Delta$ until it is not applicable and get $\sigma'\|\Upsilon'\|\Delta'$. Check whether σ' is an asymmetric unifier of Γ:
        - if it is, add the triple to Σ' and go to next member.

```
        – if it is not, union BRANCHING($\Gamma, \{\sigma' \| \Upsilon' \| \Delta'\}, \emptyset$) with $\Sigma'$.
    • If it is not, go to next member.

Return $\Sigma'$.
```

Next, we give out our algorithm *SEARCHING* which will call above three algorithms.

```
INPUT

    • An asymmetric unification problem $\Gamma$;

    • A standard unifier $\sigma$ of $\Gamma$;

PROCEDURE

    1. Let $\Sigma' = \text{SPLITTING}(\Gamma, \sigma, \emptyset, \emptyset, \emptyset)$

    2. Let $\Sigma_\sigma = \{\sigma \mid \sigma \| \Upsilon \| \Delta \in \Sigma\}$.

    3. Return $\Sigma_\sigma$.
```

5.6 Termination

We will use the following notation in the following sections.

- $\sigma\|\Upsilon\|\Delta \Rightarrow_{\mathfrak{I}_{AXO}} \sigma'\|\Upsilon'\|\Delta'$, means $\sigma'\|\Upsilon'\|\Delta'$ is deduced from $\sigma\|\Upsilon\|\Delta$ by one step.
- $\sigma\|\Upsilon\|\Delta \overset{*}{\Rightarrow}_{\mathfrak{I}_{AXO}} \sigma'\|\Upsilon'\|\Delta'$, means $\sigma'\|\Upsilon'\|\Delta'$ is deduced from $\sigma\|\Upsilon\|\Delta$ by zero or more steps.
- $\sigma\|\Upsilon\|\Delta \overset{+}{\Rightarrow}_{\mathfrak{I}_{AXO}} \sigma'\|\Upsilon'\|\Delta'$ means $\sigma'\|\Upsilon'\|\Delta'$ is deduced from $\sigma\|\Upsilon\|\Delta$ by one or more steps.

As we can see in the inference rules, some rules divide $\sigma\|\Upsilon\|\Delta$ into two (Branching rules) or even more triples (Decomposition Instantiation). So for a triple $\sigma\|\Upsilon\|\Delta$, after applying some inference rules, the result may be a disjunction of triples $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$, not just one triple. So we give the following notation:

- $\sigma\|\Upsilon\|\Delta \Rrightarrow_{\mathfrak{I}_{AXO}} \bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$, where $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$ is a disjunction of triples, means after applying some inference rule to $\sigma\|\Upsilon\|\Delta$ once, $\sigma\|\Upsilon\|\Delta$ becomes $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$.

- $\sigma\|\Upsilon\|\Delta \overset{+}{\Rrightarrow}_{\mathfrak{I}_{AXO}} \bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$, where $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$ is a disjunction of triples, means after applying some inference rules once or more than once, $\sigma\|\Upsilon\|\Delta$ becomes $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$, namely $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$ is the set of all branches we get after applying one or more than once the rules in $\mathfrak{I}_{AXO}$ to $\sigma\|\Upsilon\|\Delta$.

- $\sigma\|\Upsilon\|\Delta \overset{*}{\Rrightarrow}_{\mathfrak{I}_{AXO}} \bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$, where $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$ is a disjunction of triples, means after zero or more steps, $\sigma\|\Upsilon\|\Delta$ becomes $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$, namely $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$ is the set of all branches we get after applying zero or more times the rules in $\mathfrak{I}_{AXO}$ to $\sigma\|\Upsilon\|\Delta$.

We will give a measurement of a triple and prove it will decrease every time we apply some inference rule to the triple, such that the termination of inference system $\mathfrak{I}_{AXO}$ is proven.

Before formally giving the measurement, we need some preparation. We divide this preparation into two steps:

- Give a way of looking at the change of all the uninterpreted function terms in σ.

- Give a way of looking at the change of all the support variables in σ.

First, let use have a look at the change of all the uninterpreted function terms. Before doing this, we need some definitions and lemmas.

Definition 5.18 (Equivalent and Non-Equivalent Terms) *Let Ω be a set of identities. Let t and s be two terms. If $\Omega \models_{XOR} t = s$, we say t and s are* equivalent *in Ω and denote it by $t \simeq_\Omega s$. Otherwise, we say t and s are two* non-equivalent terms *in Ω and denote it by $t \not\simeq_\Omega s$.*

$\simeq_\Omega$ is an equivalent relation.

Example 5.19 Let $\Omega = \{f(x) \oplus f(a) = 0, v_1 \oplus x \oplus y = 0, v_2 \oplus x \oplus y = 0\}$. Then we will have the following equivalent terms:

- $f(x) \simeq_\Omega f(a)$.
- $x \simeq_\Omega a$.
- $g(x) \simeq_\Omega g(a)$, where g is some uninterpreted function symbol.
- $v_1 \simeq_\Omega x \oplus y$;
- $v_1 \simeq_\Omega v_2$;
- $f(v_1) \simeq_\Omega f(v_2)$;
- $\cdots$

For an asymmetric unification problem Γ, we use $[\![\Gamma]\!]$ to denote the following set of identities:

$$\{s =_{AC} t \mid s =_{\downarrow} t \in \Gamma\}$$

Without ambiguity, for a substitution σ we use $[\![\sigma]\!]$ to denote the following set of identities:

$$\{x =_{AC} T \mid x \leftarrow T \in \sigma\}$$

Definition 5.20 (Uninterpreted Function Terms) *We say a term t is an* uninterpreted function term *if t is not a constant or variable and t's top function symbol is an uninterpreted function symbol.*

Example 5.21 $f(x \oplus y)$ and $f(g(x,a))$ are uninterpreted function terms. x, a and $x \oplus f(a)$ are not uninterpreted function terms.

Let $Terms(\Gamma) = \{t \mid t$ occurs in Γ as a term or subterm $\}$, $Terms(\sigma) = \{t \mid t$ occurs in the range of σ as a term or subterm$\}$.

Given a set of terms Ξ and a set of identities Ω, the equivalence class of a term t in Ξ is denoted $[t]_\Omega$ and may be defined as the set:

$$[t]_\Omega^\Xi = \{s \in \Xi \mid s \simeq_\Omega t\}$$

We let $nET(\Xi, \Omega) = \{[s]_\Omega^\Xi \mid s$ is an uninterpreted function term$\}$ Since here Ξ and Ω is clear, we will simplify $[s]_\Omega^\Xi$ to $[s]$. So Here $nET(\Xi, \Omega)$ can be regarded as the set of equivalence classes of uninterpreted function terms in Ξ up to the equivalence relation Ω.

Example 5.22

$$\begin{aligned}\Gamma = \{&f(g(x, y \oplus f(a \oplus c)) \oplus y) =_\downarrow g(x,y),\\ &f(a \oplus c) \oplus f(z) =_\downarrow 0\}.\end{aligned}$$

Then

$$\begin{aligned}nET(Terms(\Gamma), [\![\Gamma]\!]) = \{&[f(g(x, y \oplus f(a \oplus c)) \oplus y)],\\ &[g(x, y \oplus f(a \oplus c))], [f(z)]\}.\end{aligned}$$

Here $f(a \oplus c) \in [f(z)]$ since $f(a \oplus c) \oplus f(z) = 0 \models f(a \oplus c) \simeq_\Omega f(z)$

Example 5.23

$$\sigma = \{x \leftarrow f(f(v_1 \oplus v_2)) \oplus f(v_1) \oplus g(v_2, a),$$
$$y \leftarrow f(v_1 \oplus v_2) \oplus f(v_2) \oplus g(b, a)\}.$$

Then

$$nET(Terms(\sigma), [\![\sigma]\!]) = \{[f(f(v_1 \oplus v_2))], [f(v_1 \oplus v_2)], [f(v_1)],$$
$$[g(v_2, a)], [f(v_2)], [g(b, a)]\}.$$

Definition 5.24 (replacement of terms) *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities. Let s and t be two uninterpreted function terms such that $[s] \in nET(\Xi_1, \Omega_1)$ and $[t] \in nET(\Xi_2, \Omega_2)$. If $\Omega_2 \models_{XOR} \Omega_1$ and $s \simeq_{\Omega_2} t$, we call $[t]$ a* replacement *of $[s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$. In this case, we write $[t] \twoheadleftarrow [s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$.*

Sometimes, we will omit "*from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$*" if it is clear. Next, we will give some properties of replacements.

Lemma 5.25 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities satisfying $\Omega_2 \models \Omega_1$. Let s and t be two uninterpreted function terms and $s \in \Xi_1$ and $t \in \Xi_2$. Let $s' \in [s]$ and $t' \in [t]$. If $s \simeq_{\Omega_2} t$ then $s' \simeq_{\Omega_2} t'$.*

This lemma shows that the property of replacements is an invariant of $\simeq_{\Omega}$.

Proof. Since for all t_j, we have $t \simeq_{\Omega_2} t'$, all we need to prove is $s' \simeq_{\Omega_2} t$.

Because $\Omega_1 \models_{XOR} s' \oplus s = 0$ and $\Omega_2 \models_{XOR} \Omega_1$, $\Omega_2 \models_{XOR} s' \oplus s = 0$. And we already have $\Omega_2 \models_{XOR} s \oplus t = 0$, so $\Omega_2 \models_{XOR} s' \oplus t = 0$. $\square$

Lemma 5.26 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities satisfying $\Omega_2 \models \Omega_1$. Let s, t_1 and t_2 be three uninterpreted function terms such that $[s] \in nET(\Xi_1, \Omega_1)$ and $[t_1], [t_2] \in nET(\Xi_2, \Omega_2)$. If $[t_1] \twoheadleftarrow [s]$ and $[t_2] \twoheadleftarrow [s]$, then $[t_1] = [t_2]$, namely $t_1 \simeq_{\Omega_2} t_2$.*

Proof. Since $[t_1] \twoheadleftarrow [s]$ and $[t_2] \twoheadleftarrow [s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$, $s \simeq_{\Omega_2} t_1$ and $s \simeq_{\Omega_2} t_2$. So $t_1 \simeq_{\Omega_2} t_2$. □

Lemma 5.27 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities satisfying $\Omega_2 \models \Omega_1$. Let $s_1, \cdots s_n$ and $t_1 \cdots t_n$ be uninterpreted function terms such that $[s_i] \in nET(\Xi_1, \Omega_1)$ and $[t_i] \in nET(\Xi_2, \Omega_2)$. If*

- *$[t_i] \twoheadleftarrow [s_i]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$;*
- *for some uninterpreted function symbol f, we have $[f(s_1, \cdots, s_n)] \in nET(\Xi_1, \Omega_1)$ and $[f(t_1, \cdots, t_n)] \in nET(\Xi_2, \Omega_2)$,*

then $[f(t_1, \cdots, t_n)]$ is a replacement of $[f(s_1, \cdots, s_n)]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$.

Proof. Since $[t_i] \twoheadleftarrow [s_i]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$, we have $t_i \simeq_{\Omega_2} s_i$. Hence $f(t_1, \cdots, t_n) \simeq_{\Omega_2} f(s_1, \cdots, s_n)$. □

Lemma 5.28 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities. If*

- *$\Omega_2 \models_{XOR} \Omega_1$; and*
- *for every $t \in nET(\Xi_2, \Omega_2)$, there exists $s \in nET(\Xi_1, \Omega_1)$, such that $[t] \twoheadleftarrow [s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$,*

then $|nET(\Xi_2, \Omega_2)| \leq |nET(\Xi_1, \Omega_1)|$.

Proof. Let $t, t' \in nET(\Xi_2, \Omega_2)$ and $s, s' \in nET(\Xi_1, \Omega_1)$, such that $[t] \twoheadleftarrow [s]$ and $[t'] \twoheadleftarrow [s']$. Then it is enough to prove if $t \not\simeq_{\Omega_2} t'$, then $s \not\simeq_{\Omega_1} s'$.

However, if $s' \simeq_{\Omega_1} s$, from Lemma 6.24, we will have $t \simeq_{\Omega_2} t'$. This is a contradiction.

□

If $nET(\Xi_1, \Omega_1)$ and $nET(\Xi_2, \Omega_2)$ satisfies the conditions in Lemma 6.27, we write $nET(\Xi_2, \Omega_2) \preceq nET(\Xi_1, \Omega_1)$.

Lemma 5.29 *If $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_2, \Omega_2)$ and $nET(\Xi_2, \Omega_2) \preceq nET(\Xi_1, \Omega_1)$, then $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_1, \Omega_1)$.*

Proof. First, because $\Omega_3 \models_{XOR} \Omega_2$ and $\Omega_2 \models_{XOR} \Omega_1$, $\Omega_3 \models_{XOR} \Omega_1$.

Second, let $[t] \in nET(\Xi_3, \Omega_3)$, then since $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_2, \Omega_2)$, there exists an equivalence class $[t']$ in $nET(\Xi_2, \Omega_2)$ such that $[t] \twoheadleftarrow [t']$ from $nET(\Xi_2, \Omega_2)$ to $nET(\Xi_3, \Omega_3)$. And we have $t' \simeq_{\Omega_3} t$

Since $nET(\Xi_2, \Omega_2) \preceq nET(\Xi_1, \Omega_1)$ and $[t'] \in nET(\Xi_2, \Omega_2)$, there exists an equivalence class $[t'']$ in $nET(\Xi_1, \Omega_1)$, such that $[t''] \twoheadleftarrow [t']$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$ and $t' \simeq_{\Omega_2} t''$. Since $\Omega_3 \models_{XOR} \Omega_2$, $t' \simeq_{\Omega_3} t''$. So $t \simeq_{\Omega_3} t''$.

Hence for every $[t] \in nET(\Xi_3, \Omega_3)$, there exists a $[t''] \in nET(\Xi_1, \Omega_1)$, such that $[t] \twoheadleftarrow [t'']$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_3, \Omega_3)$.

So $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_1, \Omega_1)$. □

From the above lemma, we know the relation "$\preceq$" is transitive.

In Decomposition Instantiation, we need to call some standard XOR unification algorithm. For proving termination, we require the standard XOR unification algorithm to have a property that we will give below.

Chapter 3 introduced a standard XOR unification algorithm $\mathfrak{A}_{XOR}$ which will generate a complete set of unifiers for a given XOR unification problem Γ. The inference system

$\mathfrak{I}_{XOR}$ has three rules *Purification, Variable Substitution*, and *Decomposition*. $\mathfrak{A}_{XOR}$ will apply inference rules to a triple: $\Gamma \| \Delta \| \Lambda$, where Γ is a set of equations to be solved, Δ is set of disequations which has the same meaning as in $\mathfrak{I}_{AXO}$ and Λ is a set of equations which are solved. Roughly, $\mathfrak{A}_{XOR}$ has the following procedure:

1. Convert every $s \stackrel{?}{=} t \in \Gamma$ to $s + t \stackrel{?}{=} 0$.

2. Apply Purification until it cannot be applied any more.

3. Apply Variable Substitution until it cannot be applied any more.

4. Apply Decomposition and go back to Step 3.

5. At last, there is a set of triples generated. For each triple, if Γ is empty, output a unifier from Λ; otherwise, abort that triple.

For this algorithm, we have the following lemma:

Lemma 5.30 *Let Γ be an XOR unification problem. Then for the standard unification algorithm $\mathfrak{A}_{XOR}$, we have:*

- $\mathfrak{A}_{XOR}(\Gamma)$ *outputs a complete set of standard XOR unifiers of Γ, which is $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$;*

- *For each σ_i, $nET(Terms(\sigma_i), [\![\sigma_i]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$.*

Before proving this lemma, we note that $\mathfrak{A}_{XOR}$ will call the syntactic unification procedure $\mathfrak{I}_{SYN}$ which has the following inference rules:

1. $\Gamma \cup \{t \stackrel{?}{=} t\} \Rightarrow \Gamma$;

2. $\Gamma \cup \{f(s_0, \ldots, s_k) \stackrel{?}{=} f(t_0, \ldots, t_k)\} \Rightarrow \Gamma \cup \{s_0 \stackrel{?}{=} t_0, \ldots, s_k \stackrel{?}{=} t_k\}$;

3. $\Gamma \cup \{f(s_0, \ldots, s_k) \overset{?}{=} g(t_0, \ldots, t_m)\} \Rightarrow \perp$ if $f \neq g$ or $k \neq m$;

4. $\Gamma \cup \{f(s_0, \ldots, s_k) \overset{?}{=} x\} \Rightarrow \Gamma \cup \{x \overset{?}{=} f(s_0, \ldots, s_k)\}$;

5. $\Gamma \cup \{x \overset{?}{=} t\} \Rightarrow \Gamma\{x \leftarrow t\} \cup \{x \overset{?}{=} t\}$ if $x \in Vars(\Gamma)$ and $x \notin Vars(t)$;

6. $\Gamma \cup \{x \overset{?}{=} f(s_0, \ldots, s_k)\} \Rightarrow \perp$ if $x \in Vars(f(s_0, \ldots, s_k))$.

If no rules are applicable to a set of equations Γ, then $\mathfrak{I}_{SYN}$ will output $\sigma = \{x \leftarrow T \mid x \overset{?}{=} T \in \Gamma\}$.

$\mathfrak{I}_{SYN}$ will generate the most general unifier σ of a syntactic unification problem Γ if Γ is unifiable. Then we have the following lemma:

Lemma 5.31 *Let Γ be a standard XOR unification problem but without $\oplus$'s occurrence such that Γ is unifiable. Then:*

- *the most general unifier σ generated by $\mathfrak{I}_{SYN}(\Gamma)$ contains no fresh variables.*
- *$nET(Terms(\sigma), [\![\sigma]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$.*

Proof. Since Γ contains no $\oplus$, this problem is a syntactic unification problem. So we can apply $\mathfrak{I}_{SYN}$ to get a most general unifier.

Since each inference does not generate any fresh variables, $\mathfrak{I}_{SYN}(\Gamma)$ will not generate any fresh variables.

For proving $nET(Terms(\sigma), [\![\sigma]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$, we only need to prove for every non-failure inference rule $\Gamma_1 \models_{XOR} \Gamma_2$, we have $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Since there are no XOR function symbols in Γ, we will omit XOR from $\models_{XOR}$.

Rule 1: $t \simeq t$ is a trivial identity. $Terms(\Gamma_2) \subseteq Terms(\Gamma_1)$ and $[\![\Gamma_2]\!] = [\![\Gamma_1]\!]$. so for every $s \in Terms(\Gamma_2)$, we have $[s] \twoheadleftarrow [s]$.

Hence $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 2: Since $\{s_i \simeq t_i \mid i = 0, \cdots k\} \models f(s_0, \cdots, s_k) \simeq f(t_0, \cdots, t_k)$, we have $[\![\Gamma_2]\!] \models [\![\Gamma_1]\!]$. Since $f(s_0, \cdots, s_k)$, $f(t_0, \cdots, t_k)$ are $\in Terms(\Gamma_1)$, $s_i \in Terms(\Gamma_1)$ and $t_i \in Terms(\Gamma_1)$ for $i = 0$ to k. So we can get $[s_i] \twoheadleftarrow [s_i]$ and $[t_i] \twoheadleftarrow [t_i]$.

Hence $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 4: $\simeq$ is symmetric, $Terms(\Gamma_2) = Terms(\Gamma_1)$ and $[\![\Gamma_2]\!] = [\![\Gamma_1]\!]$, so for every $s \in Terms(\Gamma_2)$, we have $[s] \twoheadleftarrow [s]$.

Hence $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 5: Let $s = s' \in [\![\Gamma_1]\!]$. If $s = s'$ is $x = t$, then we have $[\![\Gamma_2]\!] \models s = s'$.

Otherwise, then $s[x \leftarrow t] = s'[x \leftarrow t]$ is in $[\![\Gamma_2]\!]$. If x does not occur in s and s', then $s = s' \in [\![\Gamma_2]\!]$. If x is in $Vars(s)$ or $Vars(s')$, then let $s = s[x]$ or $s' = s'[x]$, so we will have $s[t] = s'[t] \in [\![\Gamma_2]\!]$. Because we also have $x = t \in [\![\Gamma_2]\!]$, $[\![\Gamma_2]\!] \models s[x] = s'[x]$. Hence $[\![\Gamma_2]\!] \models [\![\Gamma_1]\!]$.

For the other condition, we need to prove for every equivalence class $[s]$ in $nET(Terms(\Gamma_2), [\![\Gamma_2]\!])$, we always can find some equivalence class $[s']$ in $nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$ such that $[s] \twoheadleftarrow [s']$.

We know $\Gamma_2 = \Gamma_1\sigma \cup \{x \overset{?}{=} t\}$ where $\sigma = [x \leftarrow t]$. Suppose s has the form of $s''\sigma$, where $[s''] \in nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

If x does not occur in s'', then $s''\sigma = s'' = s$, and $[s'']$ is also in $nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$, so $[s] \twoheadleftarrow [s'']$ since $s \simeq_{[\![\Gamma_2]\!]} s''$.

If x occurs in s'', let s'' be $s''[x]$. Then we have $s''[x]\sigma = s''[t]$. Since we have $x \overset{?}{=} t \in \Gamma_2$, $s''[x] \simeq_{[\![\Gamma_2]\!]} s''[t]$. So $[s''[x]]\sigma \twoheadleftarrow [s''[x]]$ which is $[s''[t]] \twoheadleftarrow [s''[x]]$.

Hence in this case $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 6: As same as Rule 3.

In summary $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

$\square$

we know $\mathfrak{A}_{XOR}(\Gamma)$ outputs a complete set of standard XOR unifiers of Γ, which is $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$. Let us look at the procedure of $\mathfrak{A}_{XOR}$:

The first step of $\mathfrak{A}_{XOR}$ converts $s \overset{?}{=} t$ into $s + t \overset{?}{=} 0$. So $Terms(\Gamma)$ and $[\![\Gamma]\!]$ do not change. So we only need to consider the other steps.

$\mathfrak{A}_{XOR}$ starts from $\Gamma \| \emptyset \| \emptyset$ and if some unifier is found, then $\mathfrak{A}_{XOR}$ ends at $\emptyset \| \Delta \| \Lambda$ and outputs $\sigma_\Lambda = \{x \leftarrow T \mid x \overset{?}{=} T \in \Lambda\}$. So $Terms(\Lambda) = Terms(\sigma_\Lambda)$ and $[\![\Lambda]\!] = [\![\sigma_\Lambda]\!]$.

Hence for proving Lemma 5.30, we need to prove $nET(Terms(\Lambda), [\![\Lambda]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$. We will show this is true for each rule.

Proof. Let $\Gamma_1 \| \Delta_1 \| \Lambda_1$ and $\Gamma_2 \| \Delta_2 \| \Lambda_2$ be two sets of triples, such that $\Gamma_1 \| \Delta_1 \| \Lambda_1 \Rightarrow_{\mathfrak{I}_{XOR}} \Gamma_2 \| \Delta_2 \| \Lambda_2$ by applying some inference rule from $\mathfrak{I}_{XOR}$. We want to prove $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Case 1. This rule is Purification:

$$\frac{\Gamma \cup \{S \oplus t[s] \overset{?}{=} 0\} \| \Delta \| \Lambda}{\Gamma \cup \{S \oplus t[v] \overset{?}{=} 0\} \cup \{v \oplus s \overset{?}{=} 0\} \| \Delta \| \Lambda}$$

where Γ is a set of equations, S a term, t a simple term, s a proper subterm of t with $\oplus$ on the top, and v is a fresh variable.

The difference between $\Gamma_1 \cup \Lambda_1$ and $\Gamma_2 \cup \Lambda_2$ is that $S \oplus t[s] \overset{?}{=} 0 \in \Gamma_1$ but not in Γ_2 and $S \oplus t[v] \overset{?}{=} 0$ and $v \oplus s \overset{?}{=} 0$ are in Γ_2 but not in Γ_1.

For the first condition in Lemma 6.27, it is enough to prove $[\![\{S \oplus t[v] \overset{?}{=} 0, v \oplus s \overset{?}{=} 0\}]\!] \models_{XOR} [\![\{S \oplus t[s] \overset{?}{=} 0\}]\!]$. This is true because $v \oplus s \overset{?}{=} 0 \models_{XOR} v \overset{?}{=} s$.

Let $t'[v]$ be a uninterpreted function subterm in $t[v]$. So $t'[s] \in Terms(t[s]) \subseteq Terms(\Gamma_1 \cup \Lambda_1)$. Because we have $v \oplus s \overset{?}{=} 0 \in \Gamma_2$, $t'[v] \simeq_{[\![\Gamma_2 \cup \Lambda_2]\!]} t'[s]$, which means $t'[v] \twoheadleftarrow t'[s]$ from $nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$ to $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!])$

Hence $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Case 2. This rule is Variable Substitution:

$$\frac{(\Gamma \cup \{x \oplus \mathbf{S} \overset{?}{=} 0\})\|\Delta\|\Lambda}{(\Gamma\sigma)\|(\Delta\sigma)\|\Lambda\sigma \cup \{x \overset{?}{=} \mathbf{S}\}}$$

where $\sigma = [x \leftarrow \mathbf{S}]$.

For the first condition, we want to prove $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{XOR} [\![\Gamma_1 \cup \Lambda_1]\!]$. It is enough to prove for every identity $T = 0 \in [\![\Gamma_1 \cup \Lambda_1]\!]$, we have $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{XOR} T = 0$.

If $T = x \oplus S$, then $T\sigma = 0$ and $0 \neq 0$ is a trivial identity. Let $T \neq x \oplus S$. Because $T = 0 \in [\![\Gamma_1 \cup \Lambda_1]\!]$, $T\sigma = 0 \in [\![\Gamma_2 \cup \Lambda_2]\!]$. If x does not occur in T, then $T\sigma = T$. Then $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{XOR} T = 0$.

If x occurs in T, then $T[S] = 0$ and $x \oplus S = 0$ are in $[\![\Gamma_2 \cup \Lambda_2]\!]$, Hence $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{XOR} T[x] = 0$.

So $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{XOR} [\![\Gamma_1 \cup \Lambda_1]\!]$.

For the second condition, we need to prove for every equivalence class $[t]$ in $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!])$, we always can find an equivalence class $[s]$ in $nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$ such that $[t] \twoheadleftarrow [s]$.

Because $\Gamma_2 = \Gamma_1\sigma$ and $\Lambda_2 = \Lambda_1\sigma \cup \{x \oplus S = 0\}$, where $\sigma = [x \leftarrow S]$, for t, there are two possibilities:

1. t can be written as $t'\sigma$, where $[t'] \in nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$. If x does not occur in t', then $t'\sigma = t' = t$, and $[t']$ is also in $nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$, so $[t] \twoheadleftarrow [t']$ since $t \simeq_{[\![\Gamma_2 \cup \Lambda_2]\!]} t'$.

If x occurs in t', let t' be $t'[x]$ and $t'[x]\sigma = t'[S]$. Since we have $x \oplus S = 0 \in \Lambda_1$, $t'[x] \simeq_{[\![\Gamma_2 \cup \Lambda_2]\!]} t'[S]$. So $[t'[x]\sigma] \twoheadleftarrow [t'[x]]$ which is $[t'[S]] \twoheadleftarrow [t'[x]]$.

2. t is a subterm in S. Since $x \oplus S = 0 \in \Gamma_1$, $[t]$ is also in $nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$. So $[t] \twoheadleftarrow [t]$.

Hence in this case $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Case 3. The rule is Decomposition Instantiation:

If $f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \neq^? 0 \notin \Delta$,

$$\frac{\Gamma \cup \{\mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \overset{?}{=} 0\} \| \Delta \| \Lambda}{(\Gamma\sigma \cup \{\mathbf{S} \overset{?}{=} 0\}\sigma) \| (\Delta\sigma) \| (\Lambda\sigma \cup [\sigma]) \qquad \bigvee \qquad \Gamma_1' \| \Delta_1' \| \Lambda}.$$

where (i) $\sigma = mgu(s_1 \overset{?}{=} t_1, s_2 \overset{?}{=} t_2, \cdots, s_m \overset{?}{=} t_m)$ (ii) $\Gamma_1' = \Gamma \cup \{\mathbf{S} \oplus f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \overset{?}{=} 0\}$. (iii) $\Delta_1' = \Delta \cup \{f(s_1, s_2, \cdots, s_m) \oplus f(t_1, t_2, \cdots, t_m) \neq^? 0\}$

For the first branch, we know σ is the most general unifier of the syntactic unification problem $\{s_1 \overset{?}{=} t_1, s_2 \overset{?}{=} t_2, \cdots, s_m \overset{?}{=} t_m\}$. Let this syntactic unification problem be Γ. From Lemma 6.30, $nET(Terms(\sigma), [\![\sigma]\!]) \preceq nET(Terms(\Gamma_3), [\![\Gamma_3]\!])$. Recall $[\sigma] = \{x \overset{?}{=} T \mid x \leftarrow T \in \sigma\}$, we have $nET(Terms([\sigma]), [\![[\sigma]]\!]) \preceq nET(Terms(\Gamma_3), [\![\Gamma_3]\!])$. Let $\sigma = [x_1 \leftarrow u_1, \cdots, x_n \leftarrow u_n]$.

First we prove $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{XOR} [\![\Gamma_1 \cup \Lambda_1]\!]$.

If $T[x_1, \cdots, x_n] = 0 \in [\![\Gamma_1 \cup \Lambda_1]\!]$, then $T[x_1, \cdots, x_n]\sigma = 0 \in [\![\Gamma_2 \cup \Lambda_2]\!]$, which is $T[u_1, \cdots, u_n] = 0 \in [\![\Gamma_2 \cup \Lambda_2]\!]$. Because

$$\begin{aligned} &\{T[u_1, \cdots, u_n] = 0, x_1 \oplus u_1 = 0, \cdots, x_n \oplus u_n = 0\} \\ &\models_{XOR} \{T[x_1, \cdots, x_n] = 0\} \end{aligned}$$

and

$$\{S = 0, x_1 \oplus u_1 = 0, \cdots, x_n \oplus u_n = 0\}$$
$$\models_{XOR} \{S = 0, s_1 = t_1, \cdots, s_n = t_n\}$$
$$\models_{XOR} \{S = 0, f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n)\}$$
$$\models_{XOR} \{S = 0, f(s_1, \cdots, s_n) \oplus f(t_1, \cdots, t_n) = 0\}$$
$$\models_{XOR} \{S \oplus f(s_1, \cdots, s_n) \oplus f(t_1, \cdots, t_n) = 0\}$$

we have $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{XOR} [\![\Gamma_1 \cup \Lambda_1]\!]$.

Then we use a similar analysis to Case 2, and we get that for every equivalence class $[t]$ in $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!])$, there exists an equivalence class $[s]$ in $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!])$ such that $[t] \twoheadleftarrow [s]$. Hence $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

For the second branch, $\Gamma_1 \cup \Lambda_1 = \Gamma_2 \cup \Lambda_2$, so $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Hence, we get if $\Gamma_1 \| \Delta_1 \| \Lambda_1 \Rightarrow_{\mathfrak{I}_{XOR}} \Gamma_2 \| \Delta_2 \| \Lambda_2$ by applying some inference rule from $\mathfrak{I}_{XOR}$ then $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$. From Lemma 6.28, we can get:

If the algorithm starts from $\Gamma \| \emptyset \| \emptyset$ and if some unifier is found, then it ends at $\emptyset \| \Delta \| \Lambda$, we have $nET(Terms(\emptyset \cup \Lambda), [\![\emptyset \cup \Lambda]\!]) \preceq nET(Terms(\Gamma \cup \emptyset), [\![\Gamma \cup \emptyset]\!])$, which is $nET(Terms(\Lambda), [\![\Lambda]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$.

□

Lemma 5.32 *Let $\sigma \| \Upsilon \| \Delta$ be a set triple. If there exists another $\sigma' \| \Upsilon' \| \Delta'$, such that $\sigma \| \Upsilon \| \Delta \Rightarrow_{\mathfrak{I}_{AXO}} \sigma' \| \Upsilon' \| \Delta'$ via Decomposition Instantiation by applying the standard XOR* *unification algorithm with the property in Lemma 5.30, then* $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq nET(Terms(\sigma), [\![\sigma]\!])$ *and* $|nET(Terms(\sigma'), [\![\sigma']\!])| < |nET(Terms(\sigma), [\![\sigma]\!])|$.

Proof. From Decomposition Instantiation, $\sigma' = \sigma \circ \delta$, where δ is a unifier of $s \overset{?}{=} t$, where t and s are in $Terms(\sigma)$.

First we prove $[\![\sigma']\!] \models_{XOR} [\![\sigma]\!]$. Let $\delta = [v_1 \leftarrow T_1, \cdots, v_n \leftarrow T_n]$. Then if $T[v_1, \cdots, v_n] = 0 \in [\![\sigma]\!]$, then $T[v_1, \cdots, v_n]\delta = 0 \in [\![\sigma']\!]$ which is $T[T_1, \cdots, T_n] = 0 \in [\![\sigma']\!]$. And $v_i \oplus T_i = 0 \in [\![\sigma']\!]$ since $[\![\delta]\!] \subseteq [\![\sigma']\!]$. Because $\{T[T_1, \cdots, T_n] = 0, v_i \oplus T_i = 0\} \models_{XOR} T[v_1, \cdots, v_n] = 0$, $[\![\sigma']\!] \models_{XOR} T[v_1, \cdots, v_n] = 0$. Hence $[\![\sigma']\!] \models_{XOR} [\![\sigma]\!]$.

For the second condition of Lemma 6.27. Let $u \in Terms(\sigma \circ \delta)$, then u has two possibilities:

1. $u = u'\delta$, where $u' \in Terms(\sigma)$. Similar to the proof above, $u \twoheadleftarrow u'$ from $nET(Terms(\sigma'), [\![\sigma']\!])$ to $nET(Terms(\sigma), [\![\sigma]\!])$.

2. $u \in Terms(\delta)$. From Lemma 5.30, $nET(Terms(\delta), [\![\delta]\!]) \preceq nET(Terms(s \overset{?}{=} t), [\![s \overset{?}{=} t]\!])$. So there exists an equivalence class $[u']$ such that $[u] \twoheadleftarrow [u']$ from $nET(Terms(\delta), [\![\delta]\!])$ to $nET(Terms(s \overset{?}{=} t), [\![s \overset{?}{=} t]\!])$. Since $[\![\delta]\!] \models_{XOR} u \oplus u' = 0$ and $[\![\sigma \circ \delta]\!] \models_{XOR} [\![\delta]\!]$, $[\![\sigma \circ \delta]\!] \models_{XOR} u \oplus u'$. But $u' \in Terms(\sigma)$, so $[u] \twoheadleftarrow [u']$ from $nET(Terms(\sigma'), [\![\sigma']\!])$ to $nET(Terms(\sigma), [\![\sigma]\!])$.

In summary, $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq nET(Terms(\sigma), [\![\sigma]\!])$. From Lemma 6.27, we have $|nET(Terms(\sigma''), [\![\sigma']\!])| \leq |nET(Terms(\sigma), [\![\sigma]\!])|$. But in $nET(Terms(\sigma), [\![\sigma]\!])$, $[s] \neq [t]$, which means $s \not\simeq_{[\![\sigma]\!]} t$ and in $nET(Terms(\sigma'), [\![\sigma']\!])$, $[s\delta] = [t\delta]$, which means $s\delta \simeq_{[\![\sigma']\!]} t\delta$. Hence $|nET(Terms(\sigma'), [\![\sigma']\!])| < |nET(Terms(\sigma), [\![\sigma]\!])|$

□

Next, let use have a look at the change of support variables.

We call two support variables v_1 and v_2 *connected* if $(v_1, v_2) \in \Upsilon$. Otherwise, we say they are *disconnected*. If v_1 is connected to v_2, then we say v_1 is a *connection* of v_2 or v_2 is a *connection* of v_1. If a support variable has no connections, we call this variable an *isolated variable*. Otherwise, we call it a *connected variable*. If some inference rule introduces some support variable v, we say it *generates* v and v is *generated*.

Example 5.33 In Example 6.12:

$$\begin{aligned}\Gamma = \{\ & x \oplus y \oplus z =_\downarrow a, \\ & x \oplus y =_\downarrow x \oplus y, \\ & z \oplus a =_\downarrow z \oplus a\ \}\end{aligned}$$

$$\sigma = [x \leftarrow y \oplus z \oplus a]$$

After applying Splitting exhaustively, we get

$$\sigma' = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$

So far, v_1 and v_2 are generated and they are both isolated since here Υ is empty.

After applying Variable Branching for the case that y has a conflict at v_1, by letting $v_1 \leftarrow v_1' \oplus v$ and $v_2 \leftarrow v_2' \oplus v$, we get

Branch 1:

$$\sigma_1 = [x \leftarrow v_1' \oplus v_{12} \oplus a, y \leftarrow v_1' \oplus v_2', z \leftarrow v_2' \oplus v, v_1 \leftarrow v_1' \oplus v_{12}, v_2 \leftarrow v_2' \oplus v_{12}]$$

$$\Upsilon_1 = \{(v_1', v_{12}), (v_1', v_2'), (v_{12}, v_1'), (v_{12}, v_2'), (v_2', v_{12}), (v_2', v_1')\}$$

and

Branch 2:

$$\sigma_2 = [x \leftarrow v_1 \oplus a, y \leftarrow v_1 \oplus v_2, z \leftarrow v_2]$$
$$\Upsilon_2 = \{(v_2, v_1), (v_1, v_2)\}$$

In Branch 1, v_1', v_2' and v_{12} are generated. They are connected to each other. So none of these support variables are isolated. The connections of v_1' are $\{v_2', v_{12}\}$. The connections of v_2' are $\{v_1', v_{12}\}$. The connections of v_{12} are $\{v_1', v_2'\}$.

In Branch 2, no new support variables are generated. v_1 and v_2 become connected to each other.

In our algorithm, we keep generating support variables. Let us have a look at how these support variables are generated and how the connections related to these support variables change in different inference rules:

- Before applying any inference rules, we will call some standard XOR unification algorithm and get an XOR unifier σ. In σ, every support variable is isolated.

- Splitting: this inference rule uses a fresh support variable to replace an original variable. So this rule will generate an isolated variable.

- Non-variable Branching: this inference rule uses a fresh support variable v' to replace an old support variable v. All the connections of v become the connections of v'.

- Variable Branching: this inference rule uses three fresh support variables v_1', v_2' and v_{12} to replace two old support variables v_1 and v_2. All the connections of v_1 become the connections of v_1' and all the connections of v_2 become the connections of v_2'. Furthermore, all the connections of v_1 and the ones of v_2 both become the connections of v_{12}.

- Useless Branching: this inference rule is similar to Non-Variable Branching which uses a fresh support variable v' to replace an old support variable v. So all the connections of v become the connections of v'.

- Decomposition Instantiation: this inference rule may introduce some new support variables because the standard XOR unification algorithm may introduce some. These support variables will replace some original variables or support variables. Here, these support variables are isolated.

- Elimination Instantiation: this inference rule sets some support variable v to 0. The connections of v disappear.

If all the connections of v become the connections of v', we say v' *inherits* from v'. From the above analysis, we know there are three cases for the change of support variables.

- Isolated variables are generated.

- A fresh support variable v' inherits from only one old support variable v. We call v' a *clone* of v. For a trivial case, we say v is clone of himself.

- A fresh variable v_{12} inherits from two old variables v_1 and v_2. Here, we call v_{12} a *child* of v_1 and v_2, v_1 and v_2 *parents* of v_{12}, and v_{12} an *offspring* variable.

From the above definition, since clone variables inherits all the connections of their original variables, so we say a clone of an offspring variable is still an offspring variable.

If v is not an offspring variable, we call v a *root* variable. Then, a clone of a root variable is also a root variable.

As we said, clone variables inherit all the old variable's connections when it replace this old variable. It will not change the number of root or offspring variables. And here we want to see how many offspring variables may be generated when the number of root variables is fixed, so we introduce the following set:

$$\begin{aligned}\lceil v \rceil \ = \{v' \mid \ & \text{There is a sequence } \{v^1, \cdots, v^k\}, \text{ such that} \\ & v^i \text{ is clone of } v^{i-1} \text{ for } i = 2 \cdots k \text{ and} \\ & v^1 = v \text{ and } v^k = v' \ \}\end{aligned}$$

We call $\lceil v \rceil$ the *clone* set of v.

For example, when we apply Variable Branching on two variables v_1 and v_2, we will get two clone variables v_1' and v_2' and a child $v_{1,2}$. Then $v_1' \in \lceil v_1 \rceil$ and $v_2' \in \lceil v_2 \rceil$. But $v_{1,2} \notin \lceil v_1 \rceil$ and $v_{1,2} \notin \lceil v_2 \rceil$. Of course, $v_{1,2} \in \lceil v_{1,2} \rceil$.

Lemma 5.34 *The following statements are true:*

1. *If $v' \in \lceil v \rceil$ and v'' is a clone of v', then $v'' \in \lceil v \rceil$.*

2. *For $\mathfrak{A}_{AXO}$, if $v' \in \lceil v \rceil$ and $v' \in Vars(Range(\sigma))$, then $Vars(Range(\sigma)) \cap \lceil v \rceil = \{v'\}$. Namely, this means at a state of the process of $\mathfrak{A}_{AXO}$, either an offspring(root) variable or one of its clone variables can occur in the range of σ, but not both.*

Proof. Since $v' \in \lceil v \rceil$, so there exists $v_1, \cdots, v_n$, such that v_i is a clone of v_{i-1} and $v_1 = v$, $v_n = v'$. So the sequence $v_, \cdots, v_{n+1}$ where $v_{n+1} = v''$ shows that $v'' \in \lceil v \rceil$.

For the second statement, we consider the following cases based on each rule in $\mathfrak{I}_{AXO}$:

- Before applying any inference rules, we will call some standard XOR unification algorithm and get an XOR unifier σ. In σ, every support variable v is isolated. v is not a clone of any other variable. Hence $\lceil v \rceil \cap Vars(Range(\sigma)) = \{v\}$.

- Splitting: this inference rule uses a fresh support variable to replace an original variables. So this rule will generate an isolated variable v. Because v is not a clone of any other variable, $\lceil v \rceil \cap Vars(\mathit{Range}(\sigma)) = \{v\}$.

- Non-variable Branching: this inference rule uses a fresh support variable v' to replace an old support variable v. All the connections of v become the connections of v'. v' is a clone of v. Here v is removed from $Vars(\mathit{Range}(\sigma))$. If $v \in \lceil v'' \rceil$, then $v' \in \lceil v'' \rceil$ and v' is in $Vars(\mathit{Range}(\sigma))$. So $|\lceil v'' \rceil \cap Vars(\mathit{Range}(\sigma))|$ will not increase and v' is in $\lceil v'' \rceil \cap Vars(\mathit{Range}(\sigma))$.

- Variable Branching: this inference rule uses three fresh support variables v_1', v_2' and v_{12} to replace two old support variables v_1 and v_2. v_1' and v_2' are clone variables and v_{12} is an offspring variable but not a clone variable of other variables. So $\lceil v_{12} \rceil \cap Vars(\mathit{Range}(\sigma)) = \{v_{12}\}$. Similar to Non-Variable Branching, if $v_1 \in \lceil v^1 \rceil$ and $v_2 \in \lceil v^2 \rceil$, then $|\lceil v^1 \rceil \cap Vars(\mathit{Range}(\sigma))|$ will not increase, v_1' is in $\lceil v \rceil \cap Vars(\mathit{Range}(\sigma))$, and $|\lceil v^2 \rceil \cap Vars(\mathit{Range}(\sigma))|$ will not increase and v_2 is in $\lceil v^2 \rceil \cap Vars(\mathit{Range}(\sigma))$.

- Useless Branching: this inference rule is similar to Non-Variable Branching

- Decomposition Instantiation: this inference rule may introduce some new support variables because the standard XOR unification algorithm may introduce some. These support variables will replace some original variables or support variables. Here, these support variables are isolated. So there variables v are not clone variables of any other variables. So $\lceil v \rceil \cap Vars(\mathit{Range}(\sigma)) = \{v\}$.

- Elimination Instantiation: this inference rule sets some support variable v to 0. The connections of v disappear. So this rule makes some support variable not in $Vars(\mathit{Range}(\sigma))$ anymore.

So from the above analysis, if some isolated variable or offspring variable v is generated,

$Vars(Range(\sigma)) \cap \lceil v \rceil = \{v\}$. If some clone variable v' is generated, $|Vars(Range(\sigma)) \cap \lceil v \rceil|$ will not change and v' is the only variable in $Vars(Range(\sigma)) \cap \lceil v \rceil$. $\square$

For convenience, we give the following recursive notation for offspring variables.

- Root variables will have the index of one integer. Namely, we denote root variables as v_n.

- A variable is a child of $v_{i_1,\cdots,i_a}$ and $v_{j_1,\cdots,j_b}$, then we denote this offspring variable as $v_{i_1,\cdots,i_a,j_1,\cdots,j_b}$.

For a variable $v_{i_1,\cdots,i_m}$, we call m the *length* of the indices of $v_{i_1,\cdots,i_m}$ and i_k *index digit*.

So from Lemma 5.34, we can let all the members in the clone set of a variable $v_{i_1,\cdots,i_k}$ have the same index of $i_1, \cdots, i_k$. We can see this convention is compatible with the recursive notation for offspring variables. Since we have $Vars(Range(\sigma)) \cap \lceil v_{i_1,\cdots,i_k} \rceil$ only has one element, without loss of generality and for convenience, we denote this element as $\bar{v}_{i_1,\cdots,i_k}$.

We let

$$\begin{aligned}
S_1 &= \{\lceil v_1 \rceil, \cdots, \lceil v_n \rceil\} \\
S_2 &= \{\lceil v_{1,2} \rceil, \cdots, \lceil v_{1,n} \rceil, \lceil v_{2,1} \rceil, \cdots, \lceil v_{n,n-1} \rceil\} \\
&\cdots \\
S_n &= \{\lceil v_{1,\cdots,n} \rceil, \cdots \lceil v_{n,\cdots 1} \rceil\} \\
S &= S_1 \cup S_2 \cup \cdots \cup S_n.
\end{aligned}$$

S_i contains all the clone sets of the offspring variables which have i as the length of indices. For any $\lceil v_{j_1,\cdots,j_i} \rceil \in S_i$, we have $j_k \neq j_l$ for any $k \neq l$ and $k, l = 1 \cdots, i$. So S_1 is the clone set of the root variables. We call S the *family set* of support variables.

For example, if $n = 3$, we have

$$
\begin{aligned}
S_1 &= \{\lceil v_1 \rceil, \lceil v_2 \rceil, \lceil v_3 \rceil\} \\
S_2 &= \{\lceil v_{1,2} \rceil, \lceil v_{1,3} \rceil, \lceil v_{2,1} \rceil, \lceil v_{2,3} \rceil, \lceil v_{3,1} \rceil, \lceil v_{3,2} \rceil\} \\
S_3 &= \{\lceil v_{1,2,3} \rceil, \lceil v_{1,3,2} \rceil, \lceil v_{2,1,3} \rceil, \lceil v_{2,3,1} \rceil, \lceil v_{3,1,2} \rceil, \lceil v_{3,2,1} \rceil\} \\
S &= S_1 \cup S_2 \cup S_3.
\end{aligned}
$$

Because we want to see how many offspring variables may be generated when the number of root variables is fixed, we want to prove that $|S|$ will be an upper bound of the number of offspring variables which may be generated by Variable Branching.

Before proving this, we need the following lemmas.

Lemma 5.35 *If a variable $v_{i_1,\cdots i_k}$ is generated, then for every $j = 1,\cdots k$, $v_{i_1,\cdots,i_k}$ is connected to $\bar{v}_{i_j}$.*

Proof. Use induction on the length of the indices of the variable.

Base case: $v_{i,j}$ is connected to $\bar{v}_i$ and $\bar{v}_j$ because it is generated by applying Variable Branching on $\bar{v}_i$ and $\bar{v}_j$.

Induction hypothesis: Assume the lemma is true if $v_{i_1,\cdots,i_n}$ was generated for $n < k$.

Let $n = k$, then from Variable Branching, we know Variable Branching was applied on $\bar{v}_{i_1,\cdots,i_m}$ and $\bar{v}_{i_{m+1},\cdots,i_k}$ for some m. From the induction hypothesis, $v_{i_1,\cdots,i_m}$ is connected with $\bar{v}_{i_1},\cdots$ and $\bar{v}_{i_m}$. Because $\bar{v}_{i_1,\cdots,i_m}$ is a clone of $v_{i_1,\cdots,i_m}$, $\bar{v}_{i_1,\cdots,i_m}$ inherits from $v_{i_1,\cdots,i_m}$. So $\bar{v}_{i_1,\cdots,i_m}$ connects to $\bar{v}_{i_1},\cdots$ and $\bar{v}_{i_m}$. Similarly $\bar{v}_{i_{m+1},\cdots,i_k}$ is connected with $\bar{v}_{i_{m+1}},\cdots$ and $\bar{v}_{i_k}$

Because we know the child will inherit from its parents, $v_{i_1,\cdots,i_k}$ is connected to $\bar{v}_{i_1},\cdots$ and $\bar{v}_{i_k}$.

□

Lemma 5.36 *Let $v_{i_1,\cdots,i_n}$ and $v_{j_1,\cdots,j_m}$ be two variables. If there are k and l, such that $i_k = j_l$, then $\bar{v}_{i_1,\cdots,i_n}$ and $\bar{v}_{j_1,\cdots,j_m}$ are connected.*

Proof. Use induction on the sum of the length of indices of two variables.

Base case $m+n=3$. For $v_{i,j}$ and v_i, since $v_{i,j}$ is connected to $\bar{v}_i$ and $\bar{v}_j$, this lemma is true.

Induction hypothesis, suppose when $m+n<k$, this lemma is true.

Let $m+n=k$. Without loss of generality, suppose $v_{i_1,\cdots,i_n}$ was generated first. After that, when $v_{j_1,\cdots,j_m}$ was generated, Variable Branching applies on $\bar{v}_{j_1,\cdots,j_g}$ and $\bar{v}_{j_{g+1},\cdots,j_m}$ for some g. Without loss of generality, let $l\leq g$, then $\bar{v}_{j_1,\cdots,j_g}$ is connected with $\bar{v}_{i_1,\cdots,i_n}$ which inherits from $v_{i_1,\cdots,i_n}$ by the induction hypothesis. Since $v_{j_1,\cdots,j_m}$ inherits from $\bar{v}_{j_1,\cdots,j_g}$, $v_{j_1,\cdots,j_m}$ will be connected with $\bar{v}_{i_1,\cdots,i_n}$ when $v_{j_1,\cdots,j_m}$ was generated. So $\bar{v}_{j_1,\cdots,j_m}$ is connected with $\bar{v}_{i_1,\cdots,i_n}$ after $v_{j_1,\cdots,j_m}$ was generated.

□

From the above lemma, we know Variable Branching will never apply on two variables which have some same index digit.

Note: Here $|S| = \Sigma_{i=1}^{n}|S_i| = \Sigma_{i=1}^{n}\mathbf{P}_n^i$, where $\mathbf{P}_n^i$ is the ith-permutation of n, and $|S_1|=n$. So when the clone set of root variables is given, the size of the family set of the support variables is determined.

Then, next, we can prove the termination of $\mathfrak{I}_{AXO}$.

Theorem 5.37 *Let $\sigma\|\Upsilon\|\Delta$ be a set triple. Then there exists a set triple $\sigma'\|\Upsilon'\|\Delta'$, such that $\sigma\|\Upsilon\|\Delta \overset{*}{\Rightarrow}_{\mathfrak{I}_{AXO}} \sigma'\|\Upsilon'\|\Delta'$ and σ' is an asymmetric unifier or no rules in $\mathfrak{I}_{AXO}$ are applicable to $\sigma'\|\Upsilon'\|\Delta'$.*

Proof. First, we introduce several well-founded orderings.

- $Var(\Gamma)$-the set of variables in the unification problem Γ(original variables).
- $Var(dom(\sigma))$-the set of the variables in the domain of a substitution σ.
- *Range*(σ)-the range of the substitution σ.

- $Simple(\sigma)$-the set of all simple terms in the range of the substitution σ.
- $Root(\sigma)$ - the set of root variables in σ.
- $Family(\sigma)$ - the family set of the variables in σ. So $|Family(\sigma)| = \Sigma_{i=1}^{n}\mathbf{P}_{n}^{i}$, where $n = |Root(\sigma)|$
- $Sup(\sigma)$-the set of all support variables in σ. Hence, the variables in $Sup(\sigma)$ are the variables generated by the algorithm. This set is a subset of $Family(\sigma)$. So $|Family(\sigma)| - |Sup(\sigma)| \geq 0$
- $Zero(\sigma) = \{v \mid v \in Faimily(\sigma) \text{ and } v\sigma = 0\}$.
- $Pair(\sigma) = Sup(\sigma) \times Simple(\sigma)$.
- $|\Upsilon|$-The number of pairs in Υ.
- $Par(\Gamma)$ is the set of all disequations of the form $s \oplus t \neq^{?} 0$, such that s and t have the same top uninterpreted function symbol, where $s, t \in Terms(\Gamma)$.
- $Par(\Gamma/\Delta) = Par(\Gamma) - \Delta$.

Our well-founded order is the lexicographic combination of

$$\begin{aligned} M(\Gamma, \sigma, \Upsilon, \Delta) =& (|Var(\Gamma) - Var(dom(\sigma))|, |nET(Terms(\sigma), [\![\sigma]\!])| \\ & |Par(\Gamma/\Delta)|, |Family(\sigma)| - |Zero(\sigma)|, |Family(\sigma)| - |Sup(\sigma)|, \\ & |Pair(\sigma) - \Upsilon|). \end{aligned}$$

We are going to prove every time we apply a rule in $\mathfrak{I}_{AXO}$, this measurement decreases by a natural number. Since every component of this measurement is greater than or equal to zero, finally, the algorithm will stop, which means an asymmetric unifier is found or no rules are applicable to $\sigma' \| \Upsilon' \| \Delta'$.

Splitting: Some original variable is solved in each step, so $Var(Dom(\Gamma))$ increases. Hence, $|Var(\Gamma) - Var(dom(\sigma))|$ decreases.

Non-Variable Branching: For the first branch, no original variable is involved, $|Var(\Gamma) - Var(dom(\sigma))|$ does not change.

$|nET(Terms(\sigma), [\![\sigma]\!])|$ does not increase because $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq |nET(Terms(\sigma), [\![\sigma]\!])|$. It can be proved in a similar way to the proof of Case 2 in Lemma 5.30.

If $t\sigma \oplus t'\sigma \neq 0$, but $t\sigma' \oplus t\sigma' = 0$, then because $\sigma'[v' \leftarrow v \oplus s] = \sigma$ up to the variables in σ, then $t\sigma \oplus t'\sigma = 0$, which is a contradiction. And $t[v]$ is replaced by $t[v' \oplus s]$, so the number of uninterpreted function terms will not change. Hence $|Par(\Gamma/\Delta)|$ does not change.

Form the proof in the Non-Variable Branching part in Lemma 5.34, $|Family(\sigma)|$ does not change. No variables are set to be 0, so $|Family(\sigma)| - |Zero(\sigma)|$ does not change.

$|Family(\sigma)| - |Sup(\sigma)|$ does not change because $|Sup(\sigma)|$ does not change.

Let us have a look at $|Simple(\sigma)|$. Let $t \in Simple(\sigma)$. If t is a constant, $t\sigma$ is also a constant, which is still one term. If t is $f(t_1, \cdots, t_n)$, then $t\sigma = f(t_1\sigma, \cdots, t_n\sigma)$, which is also one term. If t is a variable, then $t\sigma$ has two cases: $t\sigma = t$ which is still one term, or $t\sigma = v' \oplus s$, which becomes two terms v' and s. But s is in $Simple(\sigma)$ already, so only v' replaces t as a term. Hence $|Simple(\sigma)|$ does not change. So $|Pair(\sigma)|$ does not change.

In Υ, we replace every (v, t) by (v', t) and add (v', s) into Υ, where $(v, s) \notin \Upsilon$. So $|\Upsilon|$ increases. Hence, $|Pair(\sigma)| - |\Upsilon|$ decreases.

For the second branch nothing changes except we add (v, s) into Υ. So only $|Pair(\sigma)| - |\Upsilon|$ decreases.

Variable Branching: For the first branch, no original variable is involved, $|Var(\Gamma) - Var(dom(\sigma))|$ does not change.

$|nET(Terms(\sigma), [\![\sigma]\!])|$ does not increase because $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq |nET(Terms(\sigma), [\![\sigma]\!])|$. It can be proved in a similar way to the proof of Case 2 in Lemma 5.30.

$|Par(\Gamma/\Delta)|$ does not change.

The new generated support variables are two clone variables and one offspring variable. From the proof in the Variable Branching part in Lemma 5.34 and Lemma 5.36, this offspring variable's index has no duplicated index digits, so this offspring variable is still in $|Family(\sigma)|$. Hence $|Family(\sigma)|$ does not change. And no variables are set to be 0, so $|Faimly(\sigma)| - |Zero(\sigma)|$ does not change.

Because we use *three* new support variables to replace *two* old support variables(some offspring variable is generated), and $|Sup(\sigma)|$ increases. Hence $|Family(\sigma)| - |Sup(\sigma)|$ decreases.

Useless Branching: This analysis is the same as Non-Variable Branching.

Decomposition Instantiation: If some original variable is solved, $|Var(\Gamma) - Var(dom(\sigma))|$ decreases. Otherwise $|Var(\Gamma) - Var(dom(\sigma))|$ does not change.

From Lemma 5.30, $|nET(Terms(\sigma), [\![\sigma]\!])|$ decreases.

The other branch adds some disequations into Δ, So only $|Par(\Gamma/\Delta)|$ decreases.

Elimination Instantiation: No original variable is involved, $|Var(\Gamma) - Var(dom(\sigma))|$ does not change.

$|nET(Terms(\sigma), [\![\sigma]\!])|$ does not increase because $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq |nET(Terms(\sigma), [\![\sigma]\!])|$. It can be proved in a similar way to the proof of Case 2 in Lemma 5.30 because we add a new substitution $[v \leftarrow 0]$, for some support variable v, into σ.

All $t[v]$ are replaced by $t[0]$. If some disequations become equation, then $|Par(\Gamma/\Delta)|$ decreases. Otherwise, since the number of uninterpreted function terms will not change, $|Par(\Gamma/\Delta)|$ does not increase.

$|Family(\sigma)|$ does not increase since no new variable is generated. Some support variable is set to be 0, so $|Family(\sigma)| - |Zero(Sup(\sigma))|$ decreases.

In summary, every time we apply a rule in $\mathfrak{I}_{\mathfrak{AXO}}$, $M(\Gamma, \sigma, \Upsilon, \Delta)$ decreases.

$\square$

The above theorem means our algorithm is terminating.

5.7 Soundness and Completeness

The algorithm will output a set of substitutions Σ_σ by inputting a standard unifier σ of Γ.

Intuitively, the meaning of soundness is that every member of Σ_σ is an instance of σ and is an asymmetric unifier of Γ; the meaning of completeness is if θ is an instance of σ and an asymmetric unifier, then there exists a substitution δ in Σ_σ, such that $\delta \leq_{XOR}^{\Gamma} \theta$.

In this section, we will give several lemmas which can deduce soundness and completeness of the inference system $\mathfrak{I}_{AXO}$, then give the formal statement of soundness and completeness.

The main idea of proving soundness and completeness of $\mathfrak{I}_{AXO}$ is to prove $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$ if $\sigma \| \Upsilon \| \Delta \overset{*}{\Rightarrow}_{\mathfrak{I}_{AXO}} \bigvee_i (\sigma_i \| \Upsilon_i \| \Delta_i)$.

For convenience, we use x, y and z to denote the original variables and v_i and w_i to denote the support variables.

Before proving soundness and completeness, we need to know when we cannot apply any rules in $\mathfrak{I}_{AXO}$, what form a triple $\sigma \| \Upsilon \| \Delta$ is in. In the next three lemmas, we give these forms at three difference points: the Splitting and Branching rules are not applicable; Decomposition Instantiation is not applicable and no rules are applicable.

Lemma 5.38 *Let $\sigma \| \Upsilon \| \Delta$ be a triple. If the Splitting and Branching rules are not applicable to $\sigma \| \Upsilon \| \Delta$, then*

1. *σ is an asymmetric XOR unifier of Γ; or*

2. *σ is not an asymmetric XOR unifier of Γ but satisfies the following conditions:*

   (a) *If $x \leftarrow s \oplus T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus t] \in \Gamma$, where s and t are single non-variable terms, such that $t\sigma = s$, then for each simple support variable v in T, we have either $(v, s) \in \Upsilon$ or $v \in Vars(s)$; and*

   (b) *If $x \leftarrow v \oplus T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus y] \in \Gamma$, such that $y\sigma = v \oplus S$, then*

      - *for each simple term t in $T \cup S$, we have either $(v, t) \in \Upsilon$ or $v \in Vars(t)$; and*
      - *for any top variable w in $T \cup S$, we have (w, v) and (v, w) in Υ.*

Proof. We only need to prove if we cannot apply any inference rules in the Splitting and Branching parts and σ is not an asymmetric unifier of Γ, then there are only two cases.

Case 2.a: If $(v, s) \notin \Upsilon$ and $v \notin Vars(s)$, then Non-Variable Branching is applicable.

Case 2.b: Part I: If $(v, s) \notin \Upsilon$ and $v \notin Vars(s)$, then Useless Branching is applicable.

Part II: Let (v, w) be a pair of two variables. Only Variable Branching will add a pair of two variables into Υ. In Variable Branching, when some (v, w) is added into Υ, (w, v) is added into Υ too. So $(w, v) \in \Upsilon$ if only if $(v, w) \in \Upsilon$. So this case can be restated as for any top variable w in $T \cup S$, we have (v, w) in Υ'. But If $(v, w) \notin \Upsilon$, then Variable Branching is applicable. □

Lemma 5.39 *Let $\sigma \| \Upsilon \| \Delta$ be a triple. If Splitting, Branching rules and Decomposition Instantiation are not applicable to $\sigma \| \Upsilon \| \Delta$, then:*

1. *σ is an asymmetric XOR unifier of Γ; or*

2. *σ is not an asymmetric XOR unifier of Γ but satisfies the following conditions:*

(a) *If* $x \leftarrow s \oplus T \in \sigma$ *and* $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus t] \in \Gamma$*, where* s *and* t *are two simple non-variable terms, such that* $t\sigma = s$*, then*

- *for each top variable* v *in* T*, we have either* $(v, s) \in \Upsilon$ *or* $v \in Vars(s)$*; and*
- *for every simple term* $t' \in T$ *with the same top function symbol as* s*, we have* $s \oplus t' \neq^{?} 0 \in \Delta$*.*

(b) *If* $x \leftarrow v \oplus T \in \sigma$ *and* $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus y] \in \Gamma$*, such that* $y\sigma = v \oplus S$*, then*

- *for each simple term* t *in* $T \cup S$*, we have either* $(v, t) \in \Upsilon$ *or* $v \in Vars(t)$*; and*
- *for any top variable* w *in* $T \cup S$*, we have* (w, v) *and* (v, w) *in* Υ*.*

Proof. From Lemma 5.38, all we need to prove is the second part of Case 2.a. This can be proved by looking at the conditions of Decomposition Instantiation. Recall that we will throw away any branch which violates Υ or Δ. $\square$

Lemma 5.40 *Let* $\sigma \| \Upsilon \| \Delta$ *be a triple. If no rules are applicable to* $\sigma \| \Upsilon \| \Delta$*, then:*

1. σ *is an asymmetric XOR unifier of* Γ*; or*
2. σ *is not an asymmetric XOR unifier of* Γ *but there exist* $x \leftarrow s \oplus T \in \sigma$ *and* $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus t] \in \Gamma$*, where* s *and* t *are two simple non-variable terms, such that* $t\sigma = s$*, and*
   - *for each top variable* v *in* T*, we have either* $(v, s) \in \Upsilon$ *or* $v \in Vars(s)$*; and*
   - *for every simple term* $t' \in T$ *with the same top function symbol as* s*, we have* $s \oplus t \neq^{?} 0 \in \Delta'$*.*

Proof. Since no rules in the Splitting and Branching parts are applicable and Decomposition Instantiation is not applicable either, from Lemma 5.39, all we need to prove is that Elimination Instantiation will remove Case 2.b in Lemma 5.39

- If $x \leftarrow v \oplus T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus y] \in \Gamma$, such that $y\sigma = v \oplus S$, then
  - for each simple term t in $T \cup S$, we have either $(v,t) \in \Upsilon$ or $v \in Vars(t)$; and
  - for any top variable w in $T \cup S$, we have (w,v) and (v,w) in Υ.

In this case, x has a conflict at v in σ, Elimination Instantiation will let $v \leftarrow 0$. So the conditions for applying Elimination Instantiation are satisfied in the above case and Elimination Instantiation will apply until the above case disappears. □

The next lemma shows that no step of applying inference rules in $\mathfrak{I}_{AXO}$ loses any asymmetric unifier of Γ.

Lemma 5.41 *Let $\sigma\|\Upsilon\|\Delta$, $\sigma'\|\Upsilon'\|\Delta'$, $\sigma''\|\Upsilon''\|\Delta''$ and $\sigma_i\|\Upsilon_i\|\Delta_i$ be triples, where $i = 1, \cdots, n$.*

1. *if $\sigma\|\Upsilon\|\Delta \Rrightarrow_{\mathfrak{I}_{AXO}} \sigma'\|\Upsilon'\|\Delta'$ via Splitting, then $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.*
2. *if $\sigma\|\Upsilon\|\Delta \Rrightarrow_{\mathfrak{I}_{AXO}} (\sigma'\|\Upsilon'\|\Delta' \quad \bigvee \quad \sigma''\|\Upsilon''\|\Delta'')$ via Non-Variable Branching, Variable Branching or Useless Branching then $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma') \cup \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma'')$.*
3. *if $\sigma\|\Upsilon\|\Delta \Rightarrow_{\mathfrak{I}_{AXO}} (\bigvee_{i=1}^{n}\{\sigma_i\|\Upsilon_i\|\Delta_i\}) \quad \bigvee \quad \sigma'\|\Upsilon'\|\Delta'$ via Decomposition Instantiation, then $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = (\bigcup_{i=1}^{n} \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)) \cup \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.*
4. *if $\sigma\|\Upsilon\|\Delta \Rightarrow_{\mathfrak{I}_{AXO}} \sigma'\|\Upsilon'\|\Delta'$ via Elimination Instantiation, then $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.*

Proof. **Splitting**: Splitting transforms the unifier σ into σ', such that an original top variable in *Range*(σ) is moved into *Dom*(σ'). σ' is equivalent to σ modulo XOR. if σ' violates $s \oplus t \neq^{?} 0$, that means $s\sigma' \downarrow = t\sigma' \downarrow$. Since σ' is equivalent to σ, there exists θ, such that $\sigma'\theta = \sigma$. So $s\sigma'\theta\downarrow = t\sigma'\theta\downarrow$, which is equivalent to say $s\sigma\downarrow = t\sigma\downarrow$. This is a contradiction. So it will not violate any disequation in Δ'.

In every pair (v, s) in Υ, v is a support variable. If some substitution θ violates (v, s), then $v\theta\downarrow = s\theta \oplus T'$. But in Splitting, $\sigma' = \sigma\theta$, where $\theta = [y \leftarrow v' \oplus T]$ where v' is a fresh support variable. This means $v\theta\downarrow = v$. So the result of Splitting satisfies Υ.

Hence $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.

Non-Variable Branching: Let $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ such that $\delta = \sigma\theta$. Since (v, s) is not in Υ, $v(\sigma \circ \theta)$ will have two possibilities:

- $v(\sigma \circ \theta)\downarrow = s\theta \oplus T$. This means δ violates (v, s). Since $\sigma' = \sigma[v \leftarrow v' \oplus s]$ and $\sigma\theta = \delta$, $\sigma'[v' \leftarrow T]\theta = \delta$. Therefore $([v \leftarrow v' \oplus s][v' \leftarrow T]) \circ \theta = \theta$. So:
  - δ is an instance of σ'.
  - $\sigma' \circ [v' \leftarrow T] \circ \theta$ satisfies $\Upsilon([v \leftarrow v' \oplus s] \circ [v' \leftarrow T] \circ \theta)$, since $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $[v \leftarrow v' \oplus s][v' \leftarrow T] \circ \theta = \theta$.
  - $\sigma' \circ [v' \leftarrow T] \circ \theta$ satisfies $(v', s)([v \leftarrow v' \oplus s][v' \leftarrow T] \circ \theta)$ since $T \oplus s\theta$ is irreducible
  - $\sigma' \circ [v' \leftarrow T] \circ \theta$ satisfies $\Delta([v \leftarrow v' \oplus s] \circ [v' \leftarrow T] \circ \theta)$, since $\sigma \circ \theta$ satisfies $\Delta\theta$ and $[v \leftarrow v' \oplus s][v' \leftarrow T] \circ \theta = \theta$.

  This means $\delta \in \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.

- $v(\sigma \circ \theta)\downarrow = S$ and s is not a single term of S. So δ satisfies (v, s), which means $\delta \in \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma'')$.

Therefore $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq (\mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma') \cup \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma''))$.

On the other hand, since $\sigma'[v' \leftarrow v\oplus] = \sigma$ up to the variables in σ, σ is equivalent to σ'. Hence $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \supseteq (\mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma') \cup \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma''))$ can be proven from the definition of $\supseteq$.

Variable Branching: Similar to Non-Variable Branching, let $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ such that $\delta = \sigma\theta$. Since (v_1, v_2) is not in Υ, $v_1(\sigma \circ \theta)$ and $v_2(\sigma \circ \theta)$ will have two possibilities:

- $v_1(\sigma \circ \theta) \downarrow = S\theta \oplus T$ and $v_2(\sigma \circ \theta) \downarrow = S\theta \oplus T'$ and T and T' have no common parts. This means δ violates (v_1, v_2) and (v_2, v_1). Since $\sigma' = \sigma[v_1 \leftarrow v \oplus v_1', v_2 \leftarrow v \oplus v_2']$ and $\sigma\theta = \delta$, $\sigma'[v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S]\theta = \delta$. Therefore $([v_1 \leftarrow v \oplus v_1', v_2 \leftarrow v \oplus v_2'] \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S]) \circ \theta = \theta$. So:
  - δ is an instance of σ'.
  - $\sigma' \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S] \circ \theta$ satisfies $\Upsilon([v \leftarrow v' \oplus s] \circ [v' \leftarrow T] \circ \theta)$, since $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $[v_1 \leftarrow v \oplus v_1', v_2 \leftarrow v \oplus v_2'] \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S] \circ \theta = \theta$.
  - $\sigma' \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S] \circ \theta$ satisfies $\{(v_1', v), (v_1', v_2'), (v, v_1'), (v, v_2'), (v_2', v), (v_2', v_1')\}([v_1 \leftarrow v \oplus v_1', v_2 \leftarrow v \oplus v_2'] \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S]) \circ \theta$, since $T \oplus S\theta$, $T' \oplus S$ is irreducible and T and T' have no common part.
  - $\sigma' \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S] \circ \theta$ satisfies $\Delta([v_1 \leftarrow v \oplus v_1', v_2 \leftarrow v \oplus v_2'] \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S] \circ \theta)$, since $\sigma \circ \theta$ satisfies $\Delta\theta$ and $[v_1 \leftarrow v \oplus v_1', v_2 \leftarrow v \oplus v_2'] \circ [v_1' \leftarrow T, v_2' \leftarrow T', v \leftarrow S] \circ \theta = \theta$.

  This means $\delta \in \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.

- $v_1(\sigma \circ \theta) \downarrow$ and $v_1(\sigma \circ \theta) \downarrow$ have no common parts. So δ satisfies (v_1, v_2) and (v_2, v_1), which means $\delta \in \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma'')$.

Therefore $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq (\mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma') \cup \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma''))$.

On the other hand, since $\sigma'[v \leftarrow 0, v_1' \leftarrow v_1, v_2' \leftarrow v_2] = \sigma$ up to the variables in σ, σ is equivalent to σ'. Hence $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \supseteq (\mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma') \cup \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma''))$ can be proven by the definition of the $\supseteq$.

Useless Branching: Similar to Non-Variable Branching.

Decomposition Instantiation: Let $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ such that $\delta = \sigma\theta$. Since $s \oplus t \neq^{?} 0$ is not in Δ, $s(\sigma \circ \theta)$ and $t(\sigma \circ \theta)$ will have two possibilities:

- $s(\theta) \downarrow = t(\theta) \downarrow$. Since $\{\theta_1, \cdots, \theta_n\}$ is the complete set of standard XOR unifiers of $s \oplus t =_{\downarrow} 0$, there exists i and θ', such that $\theta_i\theta' = \theta$. So $\sigma \circ \theta_i \circ \theta' = \sigma \circ \theta = |_{Vars(\Gamma)}\delta$. Therefore:

  - δ is an instance of $\sigma \circ \theta_i$, since θ is an instance of σ_i.
  - $\sigma \circ \theta_i \circ \theta'$ satisfies $\Upsilon(\theta_i \circ \theta')$ since $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $\theta_i \circ \theta' = \theta$
  - $\sigma \circ \theta_i \circ \theta'$ satisfies $\Delta(\theta_i \circ \theta')$ since $\sigma \circ \theta$ satisfies $\Delta\theta$ and $\theta_i \circ \theta' = \theta$

  This means $\delta \in \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$.

- $s(\sigma \circ \theta) \downarrow \neq t(\sigma \circ \theta) \downarrow$, so δ satisfies $s \oplus t \neq^{?} 0$. So $\delta \in \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma'')$.

Therefore, $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq (\bigcup_{i=1}^{n} \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)) \cup \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$. The other direction is straight forward. So $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = (\bigcup_{i=1}^{n} \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)) \cup \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$

Elimination Instantiation: This rule makes some support variable 0. From Lemma 5.39, if the rules in Splitting, Branching and Decomposition Instantiation are not applicable, and σ is not an asymmetric XOR unifier of Γ then $\sigma \| \Upsilon \| \Delta$ satisfies the following conditions:

1. If $x \leftarrow s \oplus T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus t] \in \Gamma$, where s and t are two simple non-variable terms, such that $t\sigma = s$, then

   - for each top variable v in T, we have either $(v, s) \in \Upsilon$ or $v \in Vars(s)$; and
   - for every simple term $t' \in T$ with the same top function symbol as s, we have $s \oplus t \neq^{?} 0 \in \Delta'$.

2. If $x \leftarrow v \oplus T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus y] \in \Gamma$, such that $y\sigma = v \oplus S$, then

   - for each simple term t in $T \cup S$, we have either $(v, t) \in \Upsilon$ or $v \in Vars(t)$; and
   - for any top variable w in $T \cup S$, we have (w, v) and (v, w) in Υ.

Let $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists θ, such that $\delta = \sigma\theta$ and $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $\Delta\theta$. We need to consider the above two cases:

Case 1: In this case, $x\delta = s\delta \oplus T\delta$. Since $t\sigma = s$, $t\sigma\theta = s\theta$. Because δ is an asymmetric unifier, we need to prove $x\delta\downarrow$ does not contain $s\delta$ as a simple term. Since s is not a variable but a simple term, $s\delta$ is a simple term too. We need some simple term s' in T such that $s'\delta \downarrow = s\delta \oplus S$.

If s' is a non-variable simple term, then s' should have the same top function symbol as s has. However, we already know $s' \oplus s \neq^? 0 \in \Delta$. So no such s' exists.

If s' is a variable v, then $v\delta \downarrow = s\delta \oplus S$, which will violates $(v, s) \in \Upsilon$. So no such v exists.

Hence, in this case, δ does not exist.

Case 2. In this case, $x\delta = v\delta \oplus T\delta$ and $y\delta = v\delta \oplus S\delta$. $x\delta \oplus y\delta = v\delta \oplus T\delta \oplus v\delta \oplus S\delta$ should be irreducible. So $v\delta$ should disappear. Let $v\delta \downarrow = t_1 \oplus \cdots \oplus t_n$. So we need t_i to disappear in $x\delta$ or $y\delta$. If t_i is cancelled by some non-variable simple term $t\delta$, then it violates $(v, t)\theta$. If t_i is cancelled by some variable w, then it violates $(v, w)\theta$. So the only method to solve this case is to let $v\delta = 0$ for every conflict.

Hence if some original variable x has a conflict at v in the premise, δ make this v 0. Therefore, δ is an instance of σ' since σ' just makes one of v's, at which some original variable has a conflict, 0s. Let $\sigma'\theta' = \delta$. Since $\sigma[v \leftarrow 0] = \sigma'$, $\sigma[v \leftarrow 0]\theta' = \delta$. So $[v \leftarrow 0] \circ \theta' = \theta$. So

- δ is an instance of σ'.
- $\sigma \circ [v \leftarrow 0] \circ \theta'$ satisfies $\Upsilon([v \leftarrow 0] \circ \theta')$, because $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $\theta = [v \leftarrow 0] \circ \theta'$.

- $\sigma \circ [v \leftarrow 0] \circ \theta'$ satisfies $\Delta([v \leftarrow 0] \circ \theta')$, because $\sigma \circ \theta$ satisfies $\Delta\theta$ and $\theta = [v \leftarrow 0] \circ \theta'$.

Hence, $\delta \in \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$, which means $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$. The other direction is straightforward. So $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.

□

From the above lemma, we proved that in every step where we apply a rule in $\mathfrak{I}_{AXO}$, any asymmetric unifier which is an instance of the premise will be an instance of one of branches, which means we do not lose any asymmetric unifier. Next, we will prove if there is no rule we can apply to some branch, then there is no asymmetric unifier in that branch.

Lemma 5.42 *Let $\sigma \| \Upsilon \| \Delta$ be a triple and Γ be an asymmetric unification problem and σ be a standard unifier but not an asymmetric unifier of Γ. If there is not another $\sigma' \| \Upsilon' \| \Delta'$, such that $\sigma \| \Upsilon \| \Delta \Rightarrow_{\mathfrak{I}_{AXO}} \sigma' \| \Upsilon' \| \Delta'$, then $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \emptyset$.*

Proof. If $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, from Lemma 5.40 we know if σ is not an asymmetric unifier of Γ, then there are the following cases if we cannot apply any rule to $\sigma \| \Upsilon \| \Delta$.

1. there exist $x \leftarrow s \oplus T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus t] \in \Gamma$, where s and t are two simple non-variable terms, such that
   - $t\sigma = s$; and
   - for each top variable v in T, we have either $(v, s) \in \Upsilon$ or $v \in Vars(s)$; and
   - for every simple term $t' \in T$ with the same top function symbol as s, we have $s \oplus t \neq^? 0 \in \Delta'$.

In this case $x\delta = s\delta \oplus T\delta$. Since $t\sigma = s$, $t\sigma\theta = s\theta$. Because δ is an asymmetric unifier, we need that $x\delta\downarrow$ does not contain $s\delta$ as a simple term. Since s is not a variable but a simple term, $s\delta$ is a simple term too. We need some simple term s' in T such that $s'\delta\downarrow = s\delta \oplus S$.

If s' is a non-variable simple term, then s' should have the same top function symbol as s has. However, we already know $s' \oplus s \neq^? 0 \in \Delta$. So no such s' exists.

If s' is a variable v, then $v\delta \downarrow = s\delta \oplus S$, which will violate $(v, s) \in \Upsilon$. So no such v exists.

Hence, in this case, δ does not exist.

□

So from the above lemma, we know we can throw out the branch $\sigma \| \Upsilon \| \Delta$ if we cannot apply any rules to it and σ is not an asymmetric unifier of Γ. Hence, we have the following theorem:

Theorem 5.43 *Let Γ be an asymmetric unification problem and σ a standard unifier of Γ. If there exists a set of triples $\bigvee_i(\sigma_i \| \Upsilon_i \| \Delta_i)$, such that $\sigma \| \emptyset \| \emptyset \overset{*}{\Rightarrow}_{\mathfrak{I}_{AXO}} \bigvee_i(\sigma_i \| \Upsilon_i \| \Delta_i)$ and no rules are applicable any more, then*

- $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$*; and*
- *the complete set of asymmetric unifiers of Γ from σ is $\{\sigma_j \mid \sigma_j \| \Upsilon_j \| \Delta_j \in \bigvee_i(\sigma_i \| \Upsilon_i \| \Delta_i)$ and σ_j is an asymmetric unifier $\}$.*

Proof. We can prove this theorem by induction via Lemma 6.37 and 6.38. □

From the above theorem, we can get the following two corollaries:

Corollary 5.44 (Soundness) Let Γ be an asymmetric unification problem and σ a standard unifier of Γ. If the algorithm SEARCHING(Γ, σ) outputs Σ_σ, then any member σ_i of Σ_σ is an asymmetric unifier of Γ and an instance of σ.

Corollary 5.45 (Completeness) Let Γ be an asymmetric unification problem and σ a standard unifier of Γ. If the algorithm SEARCHING(Γ, σ) outputs Σ_σ, then for any asymmetric unifier δ of Γ, which is an instance of σ, there exist a $\sigma_i \in \Sigma_\sigma$, such that $\sigma_i \leq^{\Gamma}_{XOR} \delta$.

5.8 Special Rules - For Efficiency

All of the rules in this section have the highest priority, which means they can be applied any time they are applicable. Using these rules will not affect the soundness and completeness of our algorithm. They are used for improving the efficiency.

Cancellation Failure

$$\frac{\sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Condition:

- There is an equation $\mathbf{u} =_{\downarrow} \mathbf{v}[s \oplus t] \in \Gamma$, such that $s\sigma = t\sigma$, where s and t are two non-sum terms.

Example 5.46

$$\Gamma = \{x \oplus a =_{\downarrow} b \oplus y, \quad z =_{\downarrow} f(x \oplus y) \oplus f(b \oplus a)\}$$

$$\sigma = [x \leftarrow a \oplus b \oplus y, z \leftarrow 0]$$

Not Enough Terms Failure

$$\frac{\sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- there is an equation $\mathbf{u} =_{\downarrow} \mathbf{v} \in \Gamma$ such that the number of simple terms in $\mathbf{u}$ is less then the number of terms in $\mathbf{v}$ and there are no top variables in $\mathbf{u}$.

Example 5.47

$$\Gamma = \{f(x) \oplus f(a) =_{\downarrow} x \oplus y \oplus c\}$$

$$\sigma = [y \leftarrow x \oplus c \oplus f(x) \oplus f(a)]$$

Identity Failure

$$\frac{\sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- there is an equation $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus S]$, such that $x\sigma = 0$, where S is not empty.

Decomposition Failure

$$\frac{[x \leftarrow s \oplus S] \cup \sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- s is a simple non-variable term;
- for every top variable v in S, $(v, s) \in \Upsilon$;
- $\mathbf{u} =_{\downarrow} \mathbf{v}[x \oplus t] \in \Gamma$ and $t\sigma = s$;
- For every term t' in S which has the same uninterpreted function symbol as s, $s \oplus t' \neq 0 \in \Delta$.

Example 5.48

$$\Gamma = \{f(a) \oplus f(b) =_{\downarrow} x \oplus f(y), f(a) \oplus f(b) =_{\downarrow} z \oplus f(y)\}$$

$$\sigma = [x \leftarrow f(a) \oplus f(b) \oplus f(y), z \leftarrow f(a) \oplus f(b) \oplus f(y)]$$

$$\Upsilon = \emptyset$$

$$\Delta = \{f(a) \oplus f(y) \neq^? 0, f(b) \oplus f(y) \neq^? 0\}$$

Example 5.49

$$\Gamma = \{y \oplus b =_{\downarrow} x \oplus a, y \oplus b =_{\downarrow} y \oplus b, f(a) \oplus f(b) =_{\downarrow} z \oplus f(y)\}$$

$$\sigma = [x \leftarrow v \oplus a \oplus b, y \leftarrow v, z \leftarrow f(a) \oplus f(b) \oplus f(v)]$$

$$\Upsilon = \{(v, a)\}$$

$$\Delta = \emptyset$$

Non-Variable Failure

$$\frac{[x_0 \leftarrow s \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_n \oplus S] \cup \sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- For all x_i, x_i has a conflict at s in the premise;
- no top variables are in S;
- s is not a variable;
- no terms in S have the same top function symbol as s.

Example 5.50

$$\Gamma = \{y \oplus b =_{\downarrow} x \oplus a, y \oplus a =_{\downarrow} y \oplus a\}$$

$$\sigma = [x \leftarrow y \oplus a \oplus b]$$

Eager Decomposition Instantiation

$$\frac{[x_0 \leftarrow s \oplus t \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_n \oplus S] \cup \sigma \| \Upsilon \| \Delta}{([x_0 \leftarrow x_1 \oplus x_2 \oplus \cdots \oplus x_n \oplus S] \cup \sigma)\theta \| \Upsilon\theta \| \Delta\theta}$$

where:

- $\theta = mgu(t, s)$

Conditions:

- For all x_i, $\mathbf{u} =_{\downarrow} \mathbf{v}[x_i \oplus s] \in \Gamma$;
- there are no top variables in $Vars(\Gamma)$;
- s is not a variable and s and t have the same top function symbol;
- $s \oplus t \neq^? 0 \notin \Delta$.

Example 5.51

$$\Gamma = \{z \oplus f(b) =_{\downarrow} x \oplus f(y), z \oplus f(y) =_{\downarrow} z \oplus f(y)\}$$

$$\sigma = [x \leftarrow f(y) \oplus z \oplus f(b)]$$

$$\Upsilon = \emptyset$$

$$\Delta = \emptyset$$

Then we will get

$$\sigma_1 = [x \leftarrow f(z), y \leftarrow b]$$
$$\Upsilon_1 = \emptyset$$
$$\Delta_1 = \emptyset$$

here σ_1 is an asymmetric unifier.

Theorem 5.52 *Let Γ be an asymmetric unification problem and σ a standard unifier of Γ. If there exists a set of triples $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$, such that $\sigma\|\emptyset\|\emptyset \Longrightarrow_{\mathfrak{I}_{AXO}} \bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i)$ via a special rule, then $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$.*

Proof. For all the failure rules, $\bigvee_i(\sigma_i\|\Upsilon_i\|\Delta_i) = Failure$, so $\bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i) = \emptyset$.

Cancellation Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ, such that $\delta = \sigma\theta$. Since $s\sigma = t\sigma$, $s\sigma\theta = t\sigma\theta$. δ is not an asymmetric unifier.

Hence $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \emptyset$.

Not Enough Terms Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$ and there exists an equation $t_1 \oplus \cdots \oplus t_n =_\downarrow s_1 \oplus \cdots s_m \in \Gamma$, where t_i is a simple non-variable term and $m > n$. Since t_i is not a variable, $t_i\delta$ is a non-variable simple term too. So $(t_1\delta \oplus \cdots \oplus t_n\delta)\downarrow$ has at most n simple terms. Hence $(s_1\delta \oplus \cdots \oplus s_m\delta)\downarrow$ should have at most n simple terms, which means $s_1\delta \oplus \cdots \oplus s_m\delta$ is reducible since $m > n$. This shows that δ is not an asymmetric unifier. So $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \emptyset$.

Identity Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ, such that $\delta = \sigma\theta$. Since $x\sigma = 0$,

$x\sigma\theta = 0$. So δ is not an asymmetric unifier.

Decomposition Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ, such that $\delta = \sigma\theta$.

In this case $x\delta = s\delta \oplus S\delta$. Since $t\sigma = s$, $t\sigma\theta = s\theta$. Because δ is an asymmetric unifier, we need that $x\delta\downarrow$ does not contain $s\delta$ as a simple term. Since s is not an variable but a simple term, $s\delta$ is a simple term too. We need some simple term s' in T such that $s'\delta\downarrow = s\delta \oplus T$.

If s' is a non-variable simple term, then s' should have the same top function symbol as s has. However, we already know $s' \oplus s \neq^? 0 \in \Delta$. So no such s' exists.

If s' is a variable v, then $v\delta\downarrow = s\delta \oplus S$, which will violate $(v, s) \in \Upsilon$. So no such v exists.

Hence, in this case, δ does not exist.

Non-Variable Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ, such that $\delta = \sigma\theta$.

In this case $x\delta = s\delta \oplus S\delta$. Since $t\sigma = s$, $t\sigma\theta = s\theta$. Because δ is an asymmetric unifier, we need $x\delta\downarrow$ does not contain $s\delta$ as a simple term. Since s is not an variable but a simple term, $s\delta$ is a simple term too. We need some simple term s' in T such that $s'\delta\downarrow = s\delta \oplus T$.

If s' is an non-variable simple term, then s' should have the same top function symbol as s has. However, we already know no term in S have the same top function symbol as s.

If s' is a variable x_i, then $x\delta\downarrow = s\delta \oplus S$, but $x_i\delta\downarrow = s\delta \oplus S$ is not an asymmetric unifier because x_i also has a conflict at s.

Hence, in this case, δ does not exist.

Eager Decomposition Instantiation

Let $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ such that $\delta = \sigma\theta$. Since $s \oplus t$ is not in Δ, $s(\sigma \circ \theta)$ and $t(\sigma \circ \theta)$ will have two cases:

- $s(\theta)\downarrow = t(\theta)\downarrow$. Since $\{\theta_1, \cdots, \theta_n\}$ is the complete set of standard XOR unifiers of $s \oplus t =_{\downarrow} 0$, there exist i and θ', such that $\theta_i\theta' = \theta$, which means θ is an instance of σ_i.

So $\sigma \circ \theta_i \circ \theta' = \sigma \circ \theta = |_{Vars(\Gamma)}\delta$. Therefore:

  - δ is an instance of $\sigma \circ \theta_i$, since θ is an instance of σ_i.
  - $\sigma \circ \theta_i \circ \theta'$ satisfies $\Upsilon(\theta_i \circ \theta')$ since $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $\theta_i \circ \theta' = \theta$
  - $\sigma \circ \theta_i \circ \theta'$ satisfies $\Delta(\theta_i \circ \theta')$ since $\sigma \circ \theta$ satisfies $\Delta\theta$ and $\theta_i \circ \theta' = \theta$

  This means $\delta \in \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$.

- $s(\sigma \circ \theta) \downarrow \neq t(\sigma \circ \theta) \downarrow$, so it satisfies $s \oplus t \neq^? 0\theta$. So $\delta \in \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma'')$.

Therefore, $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq (\bigcup_{i=1}^{n} \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)) \cup \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$. The other direction is straight forward. So $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = (\bigcup_{i=1}^{n} \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)) \cup \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$

$\square$

From the above theorem, we know none of the special rules will affect the completeness and soundness of $\mathfrak{I}_{AXO}$. Since except Special Decomposition Instantiation, all of rules go to failure directly, and we can use similar analysis in Decomposition Instantiation to prove Special Decomposition Instantiation decreases $M(\Gamma, \sigma, \Upsilon, \Delta)$, the termination is will not be affected either.

5.9 Implementation

We already implemented our asymmetric exclusive OR unification algorithm in Maude and it is being built into Maude-NPA by Santiago Escobar et al [22]. The preliminary result is very encouraging. All of the following test results are from [22].

As we mentioned in Chapter 1, a standard unification algorithm will have conflicts with the optimization of search space reduction in Maude-NPA. So here, they implement three real protocols which involve exclusive OR in three different ways: using a standard unification algorithm without optimization, using an asymmetric unification algorithm without optimization and using an asymmetric unification algorithm with optimization.

Here RP is the running protocol in Example 5.1, WEPP is the Wired Equivalent Privacy Protocol [2], and TMN is the Telecommunications Management Network protocol [59, 39]. All of these three protocols use exclusive OR. These experiments were run on an Intel Xeon machine with 4 cores and 24GB of memory, using Maude 2.7.

In Table 5.1, *states* means the number of states the Maude-NPA searches and *seconds* is the time to search. The item *steps* is the depth of the search branch. *Finite analysis* in the table to indicate whether the analysis terminates. Theoretically, without optimization, the process of the analysis will not terminate when an attack does not exist if this analysis is complete. So here we indicate "NO" for the results of three protocols without optimization. "t/o" means it did not finish within a time interval of two hours.

states/seconds	1 step	2 Steps	3 Steps	4 Steps	5 Steps	Final Analysis
RP standard	2/0.08	5/0.16	13/0.86	49/3.09	267/17.41	NO
RP w/o opt.	1/0.03	45/1.08	114/2.26	1175/37.25	13906/4144.30	t/o in 6 steps
RP w opt.	4/0.59	7/0.59	7/1.92	7/1.89	7/3.02	at Step 10
WEPP standard	5/0.09	9/0.42	26/1.27	106/5.80	503/34.76	NO
WEPP w/o opt.	4/0.05	9/0.12	26/0.64	257/144.65	2454/612.08	t/o with 7 steps
WEPP w opt.	2/0.36	2/0.20	1/0.80	2/1.42	1/0.03	at Step 5
TMN standard	5/0.11	15/0.55	99/3.82	469/25.68	timeout	NO
TMN w/o opt.	4/0.06	24/0.53	174/3.63	1079/170.29	9737/1372.55	t/o with 7 steps
TMN w opt.	3/0.42	6/9.85	9/1.78	9/4.43	8/3.20	at Step 21

Table 5.1: Experiment Result For Asymmetric XOR Unification algorithm

From this table, we can see that without optimization, using the standard unification algorithm will reduce the search space and increase the efficiency very much. This is because using asymmetric unification needs to calculate the variants, which will generate many more states. But after using the optimization, the efficiency was increased much and the number of states are "drastically reduced"! Most of the states are pruned by optimization and analysis is terminated in a finite number of steps, ”whereas no such finiteness is even theoretically possible without optimizations” [22].

5.10 Conclusion and Future Work

In this chapter, we introduced inference rules for search for a complete set of asymmetric unifier of an asymmetric XOR-unification problem from its standard unifier. We proved these inference rules to be sound, complete and terminating. We also introduced auxiliary rules to avoid applying non-deterministic rules too early, and to make the inference system more efficient.

We implemented it in Maude and it is being incorporated into Maude-NPA. The preliminary test results shows that our algorithm helps prune most of the infeasible states.

For future work, we will also think about other important theories, like exclusive OR with homomorphism and Abelian groups with homomorphism etc.

Chapter 6

General Asymmetric Abelian Group Unification

6.1 Introduction

In this chapter, we have developed a sound, complete and terminating set of inference rules for asymmetric Abelian groups unification problems along with uninterpreted function symbols.

Generally, we start from a standard unifier of the unification problem (we can get this by using the algorithm presented in Chapter 4), and search for corresponding asymmetric unifiers fulfilling the constraints from the asymmetric unification problem.

The technique is basically as same as it in Chapter 5. But unlike the algorithm in Chapter 5, the technique used here will not terminate because of the property of Abelian groups. For avoiding loops, we adopt a *bound set* to remember the depth of the branch such that we can prevent the looping by setting a bound of the depth of the branch by a user. The disadvantage of this is that it will make our algorithm incomplete. When this happens, we will call the variant narrowing algorithm to get the complete set of unifiers(the theory of Abelian groups has the finite variant property [14]).

Similar to the algorithm to solve the asymmetric exclusive OR unification problem, if there is no equivalent asymmetric unifier, we use inference rules *Useless Branching, Decomposition Instantiation* and *Elimination Instantiation* to help us find all the possible instances. Formally, we give four rules which are easy to implement and efficient. For generating a smaller complete set of unifiers and avoiding unnecessary non-deterministic steps, we also give several auxiliary rules which can be used to increase the efficiency without losing soundness, completeness and termination.

Here we give the outline of this chapter. In Section 2, we give the preliminary knowledge about the Abelian group theory related to asymmetric unification problems. In Section 3, we present the problem format and the meaning of each part in the problem. Next, we present our inference system in three parts: 1.a preprocessing step (Splitting) that transforms a standard unifier into an equivalent unifier which only contains support variables as pure term in the range is given in Section 4.1; 2. the inference rules which are used to search for equivalent asymmetric unifiers are given in Section 4.2; 3. Section 4.3 presents the inference rules which collect all the possible instances. The algorithm on how to find the asymmetric unifiers using the inference rules is give in Section 5. The proofs of termination, soundness and completeness of our inference system are given in Sections 6 and 7 respectively. Section 8 gives some auxiliary rules. The conclusion is given in Section 9.

6.2 Preliminaries

The Abelian groups(AG) theory has the property of $\mathcal{ACUI}\backslash\sqsubseteq$, namely:

- $x + (y + z) \approx (x + y) + z$ [Associativity].

- $x + y \approx y + x$ [Commutativity];
- $x + 0 \approx x$ [Unity];
- $x + (-x) \approx 0$ [exist of Inverse];

We divide the Abelian group theory to E and R, where E is $\mathcal{AC}$, namely:

- $x + y \approx y + x$;
- $x + (y + z) \approx (x + y) + z$.

and R is a convergent rewriting system of $\mathcal{UI}$, namely

- $x + 0 \to x$;
- $x + (-x) \to 0$;
- $x + y + (-x) \to y$;
- $-(-x) \to x$;
- $-0 \to 0$;
- $-(x + y) \to (-x) + (-y)$;

For a unifier σ of Γ, we have the following notation:

- We call the variables which are in the unification problem Γ *original variables* and the variables, which are not in the unification problems but in the range of σ, *support variables.*
- S a *sum* term if its normal form has the form $s_1 + \cdots + s_n$, where $n > 1$. Otherwise, we call it a *simple* term. If a simple term occurs as a non-strict sub-term in an assignment, substitution or unification problem, we call it is a *top* term in that assignment, substitution or unification problem.

- $Simple(\sigma) = \{t \mid x \leftarrow t + T \in \sigma \text{ and } t \neq 0\}$.

For a substitution σ, the assignments of support variables in σ will not affect the result of $\Gamma\sigma$. So for convenience, we will omit the assignments of support variables in σ unless we want to emphasize the assignments for support variables.

Example 6.1 If we have :

$$\Gamma = \{x + f(x + y + z) =_{\downarrow} 0\}$$
$$\delta = [x \leftarrow -f(v), y \leftarrow -z + v + f(v)]$$
$$\sigma = [z \leftarrow a, v \leftarrow b]$$

Then

$$\delta\sigma = \{x \leftarrow -f(b), y \leftarrow -a + b + f(b), z \leftarrow a\}$$

and

$$\delta \circ \sigma = \{x \leftarrow -f(b), y \leftarrow -a + b + f(b), z \leftarrow a, \mathbf{v} \leftarrow \mathbf{b}\}$$

We will rewrite every term to its normal form. In the following, when we compare two terms, we will compare them modulo $\mathcal{AC}$ if we do not indicate. For convenience, we will omit AG and Γ if they are clear. If we do not specially indicate, we will use lower case letters s, t to denote simple terms, and capital letters S, T to denote terms which may be a simple term, sum term or identity.

6.3 Problem Format

Because of the similarity of the properties of XOR and AG, we here will transplant the inference system $\mathfrak{I}_{AXO}$ to build our inference system $\mathfrak{I}_{AAG}$. Hence the idea of $\mathfrak{I}_{AAG}$ is

- The whole idea is transform members of a complete set of standard AG unifiers to a complete set of asymmetric AG unifiers.
- The whole process has two phases: Branching and Instantiation.

But there are still some differences. This is due to the difference between XOR and AG. If we simply borrow all the inference rules, we will have the following difficulties:

- Non-Variable Branching will generate loops and will not stop.
- Variable Branching will generate infinitely many support variables and will not terminate.

For conquering the first difficulty, besides σ, Υ and Δ, we will add another component Ψ, whose purpose is to control the branching steps within some bound N to prevent loops. Here N is a constant which is given by the user. So in our algorithm N will be an implicit global constant. For conquering the second difficulty, we change the way of generates support variables. This will simplify our inference system because we can combine Non-Variable Branching and Variable Branching together.

These modification will result in our inference system being not complete. Once these two cases can happen, we will call Variant Narrowing [26] , which is an inefficient but complete algorithm for solving the asymmetric unification problem, to solve the original problem. Hence, we also need an indicator to show whether these two cases happen. Since it has two values, we use a special boolean variable $\daleth$ to indicate this.

Next, we give some basic definition related to the inference rules. We introduce a set of inference rules to transform members of a complete set of standard AG unifiers to a complete set of asymmetric AG unifiers. Our inference rules involve quintuples and have the following form:

$$\frac{\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth}{\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'}$$

Here σ is a standard AG unifier of Γ. Υ is called a *constraint set*, which is a set of pairs in which each member has the form (v, s), where v is a variable and s is a simple term or a variable. We also call (v, s) a constraint pair. Δ is called a *disequation* set, in which every member has the form $s+t \neq^? 0$, where s and t have the same uninterpreted function symbol at the top. Ψ is called a *Bound Set*, which is a multi-set of pairs in which each member has the form (v, s), where v is a variable and s is a simple term or a variable. $\daleth$ is a boolean variable as we discussed above.

If $\Upsilon = \{(v_1, s_1), \cdots, (v_n, s_n)\}$, then the result of applying a substitution θ to Υ is $\Upsilon\theta = \{(v_i, s_i\theta\downarrow) \mid (v_i, s_i) \in \Upsilon\}$. If $s_i\theta\downarrow$ is not simple but of the form $t_1 + \cdots + t_n$, we will rewrite $(v_i, t_1 + \cdots + t_n)$ to $(v_i, t_1), \cdots, (v_i, t_n)$.

Similar to Υ, if $\Psi = \{(v_1, s_1), \cdots, (v_n, s_n)\}$, then the result of applying a substitution θ to Ψ is $\Psi\theta = \{(v_i, s_i\theta\downarrow) \mid (v_i, s_i) \in \Upsilon\}$. If $s_i\theta\downarrow$ is not simple but of the form $t_1 + \cdots + t_n$, we will rewrite $(v_i, t_1 + \cdots + t_n)$ to $(v_i, t_1), \cdots, (v_i, t_n)$. Since Ψ is a multi-set, we use $Count[(v, s), \Psi]$ to denote the number of (v, s) in Ψ and we let $Count[(v, s), \Psi] = 0$ if $(v, s) \notin \Psi$. Without ambiguity, we use $Count[\Psi]$ to denote the number of $\max\{Count[(v, s), \Psi] \mid (v, s) \in \Psi\}$. The reason we introduce Ψ in our inference rule is to control the steps of applying branching rules. This indicates a kind of status of applying inference rules. Later, we will give more details about Ψ.

For convenience, we have the following notation:

- $\Upsilon \cup \{(v,s)\}$ is abbreviated by $\Upsilon \cup (v,s)$
- $\Psi \cup \{(v,s)\}$ is abbreviated by $\Psi \cup (v,s)$
- $\Upsilon[v'/v]$ means the set $\{(y,t) \in \Upsilon \mid y \neq v\} \cup \{(v',t) \mid (v,t) \in \Upsilon\}$
- $\Psi[v'/v]$ means the set $\{(y,t) \in \Psi \mid y \neq v\} \cup \{(v',t) \mid (v,t) \in \Psi\}$

Definition 6.2 (Violation, Satisfaction) *Let δ be a substitution and (v,s) a constraint pair. If $v\delta\downarrow - s\delta\downarrow$ is irreducible, then we say δ **violates** the pair (v,s). Otherwise, we say δ **satisfies** Υ. If δ violates at least one pair in Υ, we say δ **violates** Υ. Otherwise, we say δ **satisfies** Υ.*

Similarly, we have

Definition 6.3 (Violation, Satisfaction) *Let δ be a substitution and $s-t \neq^? 0$ a disequation. If $(s\delta - t\delta)\downarrow = 0$, then we say δ **violates** the disequation $s-t \neq^? 0$. Otherwise, we say δ **satisfies** Δ. If δ violates at least one disequation in Δ, we say δ **violates** Δ. Otherwise, we say δ **satisfies** Δ.*

The quintuple $\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner$ denote the following set of substitutions:

Definition 6.4 (Instance of $(\Gamma, \sigma, \Upsilon, \Delta)$)

$$\begin{aligned} \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \quad & \{\delta \mid \delta \textit{ is an asymmetric AG unifier of } \Gamma; \\ & \quad \textit{and there exists } \theta \textit{ such that } \sigma\theta = \delta; \\ & \quad \textit{and } \sigma \circ \theta \textit{ satisfies } \Upsilon\theta; \\ & \quad \textit{and } \sigma \circ \theta \textit{ satisifes } \Delta\theta.\} \end{aligned}$$

We will use **Failure** as a special quintuple, such that $\mathfrak{Inst}(\textbf{Failure}) = \emptyset$.

In the following, for a unification problem Γ and an AG unifier σ, we say in the assignment $x \leftarrow t + T \in \sigma$, some original variable x has a *conflict* at some simple term t, if:

- there exists $\mathbf{u} =_{\downarrow} \mathbf{v}[x + s] \in \Gamma$ (resp. $\mathbf{u} =_{\downarrow} \mathbf{v}[x - s] \in \Gamma$) and
- there exists T' such that $s\sigma = -t + T'$ (resp. $s\sigma = t + T'$),

where s and t are simple terms and T' might be empty.

Example 6.5 $\sigma = [x \leftarrow f(a) + y, z \leftarrow a]$, and $\Gamma = \{y =_{\downarrow} x - f(z), z - a =_{\downarrow} 0\}$.

x has a conflict at $f(a)$ since $x\sigma = f(a) + y$ and $f(z)\sigma = f(a)$.

Example 6.6 $\sigma = [x \leftarrow v + a, y \leftarrow v - b]$, and $\Gamma = \{a + b =_{\downarrow} x - y\}$.

x has a conflict at v, since $x\sigma = v + a$, and $y\sigma = v - b$.

6.4 Inference System $\mathfrak{I}_{AAG}$

All of our inference rules are don't care nondeterministic rules and we divide our rules into three parts: **Splitting**, **Branching** and **Instantiation**. Some of the rules have the following form:

$$\frac{\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth}{\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth' \quad \bigvee \quad \sigma'' \| \Upsilon'' \| \Delta'' \| \Psi'' \| \daleth''}$$

Here $\bigvee$ means we divide one quintuple $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ into two quintuples $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$ and $\sigma'' \| \Upsilon'' \| \Delta'' \| \Psi'' \| \daleth''$. $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$ and $\sigma'' \| \Upsilon'' \| \Delta'' \| \Psi'' \| \daleth''$ are two independent problems with the same unification problem Γ. Without ambiguity, we use $\bigvee_{i=1}^{n}(\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i)$ to denote

$$\sigma_1 \| \Upsilon_1 \| \Delta_1 \| \Psi_1 \| \daleth_1 \bigvee \sigma_2 \| \Upsilon_2 \| \Delta_2 \| \Psi_2 \| \daleth_2 \bigvee \cdots \bigvee \sigma_n \| \Upsilon_n \| \Delta_n \| \Psi_n \| \daleth_n.$$

6.4.1 Part I - Splitting

There will be only one rule in this part, which is called Splitting. This rule will transform an AG unifier σ to an equivalent AG unifier σ' such that all the top variables in $Range(\sigma')$ are support variables.

Splitting

Splitting

$$\frac{[x \leftarrow \pm y + S + T] \cup \sigma \| \Upsilon \| \Delta \| \Psi \| \daleth}{([x \leftarrow \pm y + S + T] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta \| \Psi\theta \| \daleth}$$

where $\theta = \{y \leftarrow v \mp S\}$ and v is a fresh support variable.

Conditions:

- $x, y \in Vars(\Gamma)$;
- y does not occur as a non-top term in S.

Here, S and T can be chosen in any way we want since this rule is do not care nondeterministic. However, if x has a conflict at some simple term s in $S + T$, then for efficiency in our implementation, we will put s into T, unless $y \in Vars(s)$.

Example 6.7 We have the problem

$$\Gamma = \{y + f(a) =_{\downarrow} x + a\}$$

and let $\sigma = [x \leftarrow y + f(a) - a]$. Then

$$\frac{[x \leftarrow y + f(a) - a] \| \emptyset \| \emptyset \| \emptyset \| false}{[x \leftarrow v + f(a), y \leftarrow v + a] \| \emptyset \| \emptyset \| \emptyset \| false}$$

Example 6.8 We have the problem

$$\Gamma = \{x + z =_{\downarrow} y + a, x + a =_{\downarrow} x + a, x + z =_{\downarrow} x + z\}$$

and let $\sigma = [x \leftarrow y - z + a]$.

For convenience, we omit Υ, Ψ, Δ, and $\urcorner$ here.

First, we apply Splitting and get

$$\sigma_1 = [x \leftarrow v_1 + a, y \leftarrow v_1 + z]$$

Second, we apply Splitting again and get

$$\sigma_2 = [x \leftarrow v_1 + a, y \leftarrow v_1 + v_2, z \leftarrow v_2]$$

As we will see in the next section, Splitting will be applied first. After exhaustively applying Splitting, there will be no original variables in the range of σ. So from the next section, we will assume that all the top variables which appear in the range of σ are support variables.

6.4.2 Part II - Branching

There are three rules in this part. The main idea is to try to transform a unifier into an equivalent one without conflicts.

Branching

We will have two branching rules. The first Branching inference rule is called *Branching*, which solves the case that some original variable x has a conflict at some simple term s by dividing the quintuple to two quintuples: the substitutions in the first quintuple assume some variable v can cancel s and the substitutions in the second quintuple assume v cannot cancel s.

For example, if we have $x \leftarrow v_1 + v_2 + a$ in σ, but we also have $x - a =_{\downarrow} x - a$ in Γ, then x has a conflict at a. So our rule tries to see if v_1 can cancel a or v_2 can cancel a in two steps.

It will also be applicable if s is a variable. For example, suppose we have $x \leftarrow v_1 + v_2 + v_3, y \leftarrow v_1$ in σ and $x - y =_{\downarrow} x - y$ in Γ. We will try to see whether v_2 or v_3 can cancel v_1.

Since applying a substitution to a variable will turn the variable into a sum term, it is also possible that sum $v_1 + v_3$ cancels v_1. In asymmetric XOR unification, we will let $v_2 \leftarrow v_2' + v$ and $v_1 \leftarrow v_1' + v$. But unlike the inference system $\mathfrak{I}_{AXO}$, this case will generate infinitely many support variables here. So this case will not be considered. This is why we can combine Non-Variable Branching and Variable Branching in asymmetric XOR unification into one Branching rule.

Formally:

Branching

$$\frac{\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner}{\sigma \circ \theta \| \Upsilon' \theta \| \Delta \theta \| \Psi' \theta \| \urcorner' \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, \mp s)\} \| \Delta \| \Psi \| \urcorner}$$

where

- there exists an assignment $[x \leftarrow \pm v + s + S] \in \sigma$;
- $\Upsilon' = \Upsilon[v'/v] \cup (v', \pm s)$;
- $\Psi' = \Psi[v'/v] \cup (v', \mp s)$;
- $\theta = [v \leftarrow v' \mp s]$;

- $\daleth' = True$ if $Count[(v, \mp s), \Psi] = N - 1$ or s is a variable, otherwise, $\daleth' = \daleth$.

Conditions:

- s is a simple term but not a variable and $v \notin Vars(s)$;
- x has a conflict at s in σ;
- $(v, \mp s) \notin \Upsilon$.
- $Count[(v, \mp s), \Psi] < N$ for the bound N.

Note: Here, $\pm s$ means there are two cases s and $-s$; $\mp s$ means corresponded two cases $-s$ and s.

Example 6.9 We continue Example 6.8.

$$\Gamma = \{x + z =_{\downarrow} y + a, x + a =_{\downarrow} x + a, x + z =_{\downarrow} x + z\}$$

Let $\sigma = [x \leftarrow v_1 + a, y \leftarrow v_1 + v_2, z \leftarrow v_2]$.

For convenience, we omit Δ here.

Since x has a conflict at a, we can apply the Branching rule and get:

Branch 1:

$$\sigma_1 = [x \leftarrow v_3, y \leftarrow v_3 - a + v_2, z \leftarrow v_2]$$

$$\Upsilon_1 = \{(v_3, a)\}$$

$$\Psi_1 = \{(v_3, -a)\}$$

where $\theta = [v_1 \leftarrow v_3 - a]$; and Branch 2

$$\begin{aligned}
&\sigma_2 = [x \leftarrow v_1 + a, y \leftarrow v_1 + v_2, z \leftarrow v_2] \\
&\Upsilon_2 = \{(v_1, -a)\} \\
&\Psi_2 = \emptyset
\end{aligned}$$

In Branch 1, y has a conflict at $-a$ and $(v_3, a) \in \Upsilon_1$, so apply Branching, let $v_2 \leftarrow v_4 + a$ and get:

Branch 1.1

$$\begin{aligned}
&\sigma_{1.1} = [x \leftarrow v_3, y \leftarrow v_3 + v_4, z \leftarrow v_4 + a] \\
&\Upsilon_{1.1} = \{(v_3, a), (v_4, -a)\} \\
&\Psi_{1.1} = \{(v_3, -a), (v_4, a)\}
\end{aligned}$$

where $\theta = [v_2 \leftarrow v_4 + a]$, and Branch 1.2

$$\begin{aligned}
&\sigma_{1.2} = [x \leftarrow v_3, y \leftarrow v_3 - a + v_2, z \leftarrow v_2] \\
&\Upsilon_{1.2} = \{(v_3, a), (v_2, a)\}
\end{aligned}$$

We can see $\sigma_{1.1}$ is an asymmetric unifier of Γ.

Here $\sigma_{1.1}$ is equivalent to σ. Branching will not apply to Branch 2 and 1.2 because the third condition for applying Branching are not satisfied.

Example 6.10

$$\Gamma = \{x + y + z =_{\downarrow} a, x + y =_{\downarrow} x + y\}$$

with $\sigma = [x \leftarrow a - y - z]$. After applying Splitting exhaustively, we get

$$\sigma = [x \leftarrow a - v_1 - v_2, \quad y \leftarrow v_1, \quad z \leftarrow v_2]$$

For convenience, we omit Δ here.

x has a conflict with v_1, so we can apply Branching and get:

Branch 1:

$$\begin{aligned}
&\sigma_1 = [x \leftarrow a - v_3, \quad y \leftarrow v_1, \quad z \leftarrow v_3 - v_1, \quad v_2 \leftarrow v_3 - v_1];\\
&\Upsilon_1 = \{(v_3, v_1)\};\\
&\Psi_1 = \{(v_3, -v_1)\}.
\end{aligned}$$

and Branch 2

$$\begin{aligned}
&\sigma_2 = [x \leftarrow a - v_1 - v_2, \quad y \leftarrow v_1, \quad z \leftarrow v_2];\\
&\Upsilon_2 = \{(v_2, -v_1)\};\\
&\Psi_2 = \emptyset.
\end{aligned}$$

We will stop here because σ_1 is an asymmetric unifier for Γ. Furthermore, we cannot apply Branching on Branch 2 since $(v_2, -v_1)$ is in Υ.

The next example will explain how Ψ works. In the next example, we let 2 be the bound.

Example 6.11

$$\begin{aligned}
\Gamma &= \{x - y =_{\downarrow} a, \quad y - w_1 - w_2 =_{\downarrow} 0, \quad z - a =_{\downarrow} 0 \\
&\quad x - z =_{\downarrow} x - z, \quad y + z =_{\downarrow} y + z\}. \\
\sigma &= [x \leftarrow v_1 + v_2 + a, \quad y \leftarrow v_1 + v_2 \\
&\quad z \leftarrow a, \quad w_1 \leftarrow v_1, \quad w_2 \leftarrow v_2]; \\
\Upsilon &= \emptyset; \\
\Psi &= \emptyset; \\
\daleth &= false;
\end{aligned}$$

Here x has a conflict with a, after applying Branching, we get:

Branch 1:

$$\begin{aligned}
\sigma_1 &= [x \leftarrow v_3 + v_2, \quad y \leftarrow v_3 + v_2 - a, \\
&\quad z \leftarrow a, \quad w_1 \leftarrow v_3 - a, \quad w_2 \leftarrow v_2, \\
&\quad v_1 \leftarrow v_3 - a]; \\
\Upsilon_1 &= \{(v_3, a)\}; \\
\Psi_1 &= \{(v_3, -a)\}; \\
\daleth_1 &= false.
\end{aligned}$$

and Branch 2:

$$
\begin{aligned}
\sigma_2 = [&x \leftarrow v_1 + v_2 + a, \quad y \leftarrow v_1 + v_2 \\
&z \leftarrow a, \quad w_1 \leftarrow v_1, \quad w_2 \leftarrow v_2]; \\
\Upsilon_2 = \{&(v_1, -a)\}; \\
\Psi_2 = \emptyset&; \\
\daleth_2 = f&alse;
\end{aligned}
$$

For Branch 1, the conflict is y with a, applying Branching to get:

Branch 1.1

$$
\begin{aligned}
\sigma_{1.1} = [&x \leftarrow v_3 + v_4 + a, \quad y \leftarrow v_3 + v_4, \\
&z \leftarrow a, \quad w_1 \leftarrow v_3 - a, \quad w_2 \leftarrow v_4 + a, \\
&v_1 \leftarrow v_3 - a, \quad v_2 \leftarrow v_4 + a]; \\
\Upsilon_{1.1} = \{&(v_3, a), \quad (v_4, -a)\}; \\
\Psi_{1.1} = \{&(v_3, -a), (v_4, a)\}; \\
\daleth_{1.1} = \ & false.
\end{aligned}
$$

and Branch 1.2

$$\begin{aligned}
\sigma_{1.2} = [&x \leftarrow v_3 + v_2,\quad y \leftarrow v_3 + v_2 - a,\\
&z \leftarrow a,\quad w_1 \leftarrow v_3 - a,\quad w_2 \leftarrow v_2,\\
&v_1 \leftarrow v_3 - a];\\
\Upsilon_{1.2} &= \{(v_3, a),\quad (v_2, a)\};\\
\Psi_{1.2} &= \{(v_3, -a)\};\\
\daleth_{1.2} &= false.
\end{aligned}$$

We continue Branch 1.1. Because the conflict is x and a, apply Branching and get:

Branch 1.1.1

$$\begin{aligned}
\sigma_{1.1.1} = [&x \leftarrow v_5 + v_4,\quad y \leftarrow v_5 + v_4 - a,\\
&z \leftarrow a,\quad w_1 \leftarrow v_5 - a - a,\quad w_2 \leftarrow v_4 + a,\\
&v_1 \leftarrow v_5 - a - a,\quad v_2 \leftarrow v_4 + a, v_3 \leftarrow v_5 - a];\\
\Upsilon_{1.1.1} &= \{(v_5, a),\quad (v_4, -a)\};\\
\Psi_{1.1.1} &= \{(v_5, -a),\quad (v_4, a),\quad (v_5, -a)\};\\
\daleth_{1.1.1} &= true.
\end{aligned}$$

and Branching 1.1.2

$$\sigma_{1.1.2} = [x \leftarrow v_3 + v_4 + a, \quad y \leftarrow v_3 + v_4,$$

$$z \leftarrow a, \quad w_1 \leftarrow v_3 - a, \quad w_2 \leftarrow v_4 + a,$$

$$v_1 \leftarrow v_3 - a, \quad v_2 \leftarrow v_4 + a];$$

$$\Upsilon_{1.1.2} = \{(v_3, a), \quad (v_4, -a), \quad (v_3, -a)\};$$

$$\Psi_{1.1.2} = \{(v_3, -a), (v_4, a)\};$$

$$\daleth_{1.1.2} = false.$$

In Branch 1.1.1, we can see there are two $(v_5, -a)$s and reach the bound 2. So here we set $\daleth_{1.1.1} = true$.

Continuing Branch 1.1.1, we still can apply Branching and get:

Branch 1.1.1.1

$$\sigma_{1.1.1.1} = [x \leftarrow v_5 + v_6 + a, \quad y \leftarrow v_5 + v_6,$$

$$z \leftarrow a, \quad w_1 \leftarrow v_5 - a - a, \quad w_2 \leftarrow v_6 + a + a,$$

$$v_1 \leftarrow v_5 - a - a, \quad v_2 \leftarrow v_6 + a + a, v_3 \leftarrow v_5 - a, \quad v_4 \leftarrow v_6 + a];$$

$$\Upsilon_{1.1.1.1} = \{(v_5, a), \quad (v_6, -a)\};$$

$$\Psi_{1.1.1.1} = \{(v_5, -a), \quad (v_6, a), \quad (v_5, -a), \quad (v_6, a)\};$$

$$\daleth_{1.1.1.1} = true.$$

and Branch 1.1.1.2

$$\begin{aligned}\sigma_{1.1.1.2} = [&x \leftarrow v_5 + v_4, \quad y \leftarrow v_5 + v_4 - a,\\ &z \leftarrow a, \quad w_1 \leftarrow v_5 - a - a, \quad w_2 \leftarrow v_4 + a,\\ &v_1 \leftarrow v_5 - a - a, \quad v_2 \leftarrow v_4 + a, v_3 \leftarrow v_5 - a];\\ \Upsilon_{1.1.1.2} = \{&(v_5, a), \quad (v_4, -a), \quad (v_4, a)\};\\ \Psi_{1.1.1.2} = \{&(v_5, -a), \quad (v_4, a), \quad (v_5, -a)\};\\ \urcorner_{1.1.1.2} = \ &true.\end{aligned}$$

For Branch 1.1.1.1, we cannot apply Branching again though the conflicts are still there, because $(v_5, -a)$ occurs twice in Ψ already. We also cannot apply Branching on Branch 1.1.1.2 because (v_5, a) and (v_4, a) are both in Υ, such that no term in $v_5 + v_4 - a$ can cancel a.

Let us go back to Branch 1.1.2. Here we cannot apply Branching on it because $(v_3, -a)$ and $(v_4, -a)$ are both in Υ.

Then go back to Branch 1.2. Similarly, we still cannot apply Branching because (v_3, a), and (v_2, a) are both in Υ.

Going back to Branch 2, the conflict is still x with a, so apply Branching and get:

Branch 2.1:

$$\begin{aligned}
\sigma_{2.1} = [&x \leftarrow v_1 + v_3, \quad y \leftarrow v_1 + v_3 - a \\
&z \leftarrow a, \quad w_1 \leftarrow v_1, \quad w_2 \leftarrow v_3 - a \\
&v2 \leftarrow v_3 - a]; \\
\Upsilon_{2.1} = &\{(v_1, -a),\ (v_3, a)\}; \\
\Psi_{2.1} = &\{(v_3, -a)\}; \\
\daleth_{2.1} = &false;
\end{aligned}$$

and Branch 2.2:

$$\begin{aligned}
\sigma_2 = [&x \leftarrow v_1 + v_2 + a, \quad y \leftarrow v_1 + v_2 \\
&z \leftarrow a, \quad w_1 \leftarrow v_1, \quad w_2 \leftarrow v_2]; \\
\Upsilon_2 = &\{(v_1, -a), (v_2, -a)\}; \\
\Psi_2 = &\emptyset; \\
\daleth_2 = &false;
\end{aligned}$$

We still can apply Branching on Branch 2.1 and get:

Branch 2.1.1:

$$
\begin{aligned}
\sigma_{2.1.1} = [&x \leftarrow v_4 + v_3 + a, \quad y \leftarrow v_4 + v_3 \\
&z \leftarrow a, \quad w_1 \leftarrow v_4 + a, \quad w_2 \leftarrow v_3 - a \\
&v_2 \leftarrow v_3 - a, v_1 \leftarrow v_4 + a]; \\
\Upsilon_{2.1.1} = &\{(v_4, -a),\ (v_3, a)\}; \\
\Psi_{2.1.1} = &\{(v_3, -a), (v_4, a)\}; \\
\daleth_{2.1.1} = &false;
\end{aligned}
$$

and Branch 2.1.2

$$
\begin{aligned}
\sigma_{2.1.2} = [&x \leftarrow v_1 + v_3, \quad y \leftarrow v_1 + v_3 - a \\
&z \leftarrow a, \quad w_1 \leftarrow v_1, \quad w_2 \leftarrow v_3 - a \\
&v2 \leftarrow v_3 - a]; \\
\Upsilon_{2.1.2} = &\{(v_1, -a), \quad (v_3, a), \quad (v_1, a)\}; \\
\Psi_{2.1.2} = &\{(v_3, -a)\}; \\
\daleth_{2.1.2} = &false;
\end{aligned}
$$

For Branch 2.1.1, we apply Branching and get:

Branch 2.1.1.1:

$$\begin{aligned}\sigma_{2.1.1.1} = [&x \leftarrow v_4 + v_5, \quad y \leftarrow v_4 + v_5 - a \\ &z \leftarrow a, \quad w_1 \leftarrow v_4 + a, \quad w_2 \leftarrow v_5 - a - a \\ &v_2 \leftarrow v_5 - a - a, v_1 \leftarrow v_4 + a, v_3 \leftarrow v_5 - a]; \\ \Upsilon_{2.1.1.1} = \{&(v_4, -a), \quad (v_5, a)\}; \\ \Psi_{2.1.1.1} = \{&(v_5, -a), (v_4, a), (v_5, -a)\}; \\ \urcorner_{2.1.1.1} = {}&true;\end{aligned}$$

and Branch 2.1.1.2

$$\begin{aligned}\sigma_{2.1.1.2} = [&x \leftarrow v_4 + v_3 + a, \quad y \leftarrow v_4 + v_3 \\ &z \leftarrow a, \quad w_1 \leftarrow v_4 + a, \quad w_2 \leftarrow v_3 - a \\ &v_2 \leftarrow v_3 - a, v_1 \leftarrow v_4 + a]; \\ \Upsilon_{2.1.1.2} = \{&(v_4, -a), \quad (v_3, a), \quad (v_3, -a)\}; \\ \Psi_{2.1.1.2} = \{&(v_3, -a), (v_4, a)\}; \\ \urcorner_{2.1.1.2} = {}&false;\end{aligned}$$

In Branch 2.1.1.1, $(v_5, -a)$ occurs twice already in Ψ and (v_4, a) is in Υ, so Branching is not applicable. In Branch 2.1.1.2, $(v_4, -a)$ and $(v_3, -a)$ are both in Υ, then Branching is not applicable.

We go back to Branch 2.1.2. Because (v_3, a) and (v_1, a) are both in Υ, Branching is not applicable.

Going back to Branch 2.2, because $(v_1, -a)$ and $(v_2, -a)$ are both in Υ, Branching is not applicable.

So we stop here.

From the above example, $\sigma_{1.1}$ is a renaming of the original σ. If we do not have the bound, the procedure will go forever. If we give a smaller bound, like $N = 1$, it will not go through the branches after Branch 1.1.

Suppose we change the above example to

$$\begin{aligned}\Gamma = \{&x - y - w =_\downarrow a, \quad y - w_1 - w_2 - w_3 =_\downarrow 0, \quad z - a =_\downarrow 0 \\ &x - z =_\downarrow x - z, \quad y + z =_\downarrow y + z\}.\end{aligned}$$

with the following initial components:

$$\begin{aligned}\sigma = [&x \leftarrow v_1 + v_2 + v_3 + a, \quad y \leftarrow v_1 + v_2 \\ &z \leftarrow a, \quad w_1 \leftarrow v_1, \quad w_2 \leftarrow v_2, \quad w \leftarrow v_3];\\ \Upsilon = \emptyset&;\\ \Psi = \emptyset&;\\ \daleth = &\mathit{false};\end{aligned}$$

If we use the almost same order of applying Branching as the above example, we will find a solution in Branch 2.2.1. If we do not have Υ, we may go into a loop and will not get the solution.

On the other hand, if we have the following problem: $\{x - y =_\downarrow a + a + a,\ x - a =_\downarrow x - a$ with $[x \leftarrow v_1 + a + a + a, y \leftarrow v_1]$, then it is not hard to see if the bound is less than 3, we will not find the solution by Branching. So choosing a proper bound is also important.

If we stop due to the limit of the bound, we could not say there is no equivalent solution. This is why we need to call Variant Narrowing to solve the problem another way.

Example 6.12

$$\begin{aligned}\Gamma = \{&x + y + z =_{\downarrow} a,\\ &x + y =_{\downarrow} x + y,\\ &z + a =_{\downarrow} z + a\}\\ \sigma = [&x \leftarrow -y - z + a]\end{aligned}$$

After applying Splitting exhaustively, we can get

$$\sigma' = [x \leftarrow -v_1 + a, y \leftarrow v_1 - v_2, z \leftarrow v_2]$$

After applying Branching for the case that y has a conflict at v_1, by letting $v_2 \leftarrow v_3 + v_1$ we get

Branch 1:

$$\begin{aligned}\sigma_1 &= [x \leftarrow -v_1 + a, y \leftarrow -v_3, z \leftarrow v_3 + v_1];\\ \Upsilon_1 &= \{(v_3, -v_1)\};\\ \Psi &= \{(v_3, v_1)\}\end{aligned}$$

and Branch 2:

$$\sigma_2 = [x \leftarrow -v_1 + a, y \leftarrow v_1 - v_2, z \leftarrow v_2];$$

$$\Upsilon_2 = \{(v_2, v_1)\};$$

$$\Psi_2 = \emptyset.$$

We can see σ_1 is an asymmetric unifier of Γ.

Useless Branching

The next rule is called *Useless Branching.* It will be applied to the case that some original variable x has a conflict at some support variable v. But it will not solve the conflict directly. It is preparing for the instantiation part. In Useless Branching, we guess there is another simple non-variable term t, such that $t\theta \downarrow + v\theta \downarrow$ is irreducible.

For example if we have $x \leftarrow a + v$ and $y \leftarrow v$ in σ and $x + y =_{\downarrow} x + y$ in Γ we will guess in some substitution θ, $v\theta \downarrow = a\theta + S$ by letting $v \leftarrow v' + a$.

Formally, the rule is:

Useless Branching

$$\frac{\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner}{\sigma \circ \theta \| \Upsilon'\theta \| \Delta\theta \| \Psi'\theta \| \urcorner' \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, \mp s)\} \| \Delta \| \Psi \| \urcorner}$$

where

- There exists an assignment $x \leftarrow \pm v + s + S$ in σ;
- $\Upsilon' = \Upsilon[v'/v] \cup (v', \pm s)$;
- $\Psi' = (\Psi[v'/v] \cup (v', \mp s))$;

- $\theta = [v \leftarrow v' \mp s]$;
- $\urcorner' = True$ if $Count[(v, \mp s), \Psi] = N - 1$ or s is a variable, otherwise, $\urcorner' = \urcorner$.

Conditions:

- s is a simple term but not a variable and $v \notin Vars(s)$;
- x has a conflict at v in σ.
- $(v, \mp s) \notin \Upsilon$.
- $Count[(v, \mp s), \Psi] < N$ for the bound N.

Example 6.13 Let $\Gamma = \{a + b =_{\downarrow} x + y\}$ and we have $\sigma = [x \leftarrow a + b - y]$. After applying Splitting, we get (We omit Δ and $\urcorner$ here for convenience) :

$$\sigma = [x \leftarrow a + b - v_1, \quad y \leftarrow v_1];$$
$$\Upsilon = \emptyset;$$
$$\Psi = \emptyset.$$

We cannot apply Branching but can apply Useless Branching. We will get

Branch 1

$$\sigma_1 = [x \leftarrow b - v_2, \quad y \leftarrow v_2 + a, \quad v_1 \leftarrow v_2 + a];$$
$$\Upsilon_1 = \{(v_2, -a)\};$$
$$\Psi_1 = \{(v_2, a)\}.$$

and Branch 2

$$\sigma_2 = [x \leftarrow a + b - v_1, \quad y \leftarrow v_1];$$
$$\Upsilon_2 = \{(v_1, a)\};$$
$$\Psi_2 = \emptyset.$$

We still can apply Useless Branching on Branch 1 and get

Branch 1.1

$$\sigma_{1.1} = [x \leftarrow v_3, \quad y \leftarrow v_3 + a + b, \quad v_1 \leftarrow v_3 + a + b, \quad v_2 \leftarrow v_3 + b];$$
$$\Upsilon_{1.1} = \{(v_3, -a), \quad (v_3, -b)\};$$
$$\Psi_{1.1} = \{(v_3, a), \quad (v_3, b)\}.$$

and Branch 1.2

$$\sigma_{1.2} = [x \leftarrow b - v_2, \quad y \leftarrow v_2 + a, \quad v_1 \leftarrow v_2 + a];$$

$$\Upsilon_{1.2} = \{(v_2, -a), \quad (v_2, b)\};$$

$$\Psi_{1.2} = \{(v_2, a)\}.$$

Now, we have no rules to apply on Branch 1.1 and 1.2. Let us go back to Branch 2. We can apply Useless Branching on it and get:

Branch 2.1

$$\sigma_{2.1} = [x \leftarrow a - v_2, \quad y \leftarrow v_2 + b \quad v_1 \leftarrow v_2 + b];$$

$$\Upsilon_{2.1} = \{(v_2, a), \quad (v_2, -b)\};$$

$$\Psi_{2.1} = \{(v_2, b)\}.$$

and Branch 2.2

$$\sigma_{2.2} = [x \leftarrow a + b - v_1, \quad y \leftarrow v_1];$$

$$\Upsilon_{2.2} = \{(v_1, a), \quad (v_1, b)\};$$

$$\Psi_{2.2} = \emptyset.$$

For Branch 2.1 and 2.2, we do not have any rules applicable so far. So we will stop here and get four branches.

As we said above, Useless Branching is not used for searching for equivalent asymmetric unifiers but preparing for instantiation. So the above example will be continued in the following sections.

6.4.3 Part III - Instantiation

The following rules are called Instantiation Rules. They are used for solving the conflicts by instantiating some support variables.

Once we get an instance, the result may violate Υ or Δ. If this is true, we will throw out this instantiation branch.

Decomposition Instantiation

The first instantiation rule is called Decomposition Instantiation. It is used to solve the case that some original variable x has a conflict at some uninterpreted function term t.

For example if $x \leftarrow f(a) - f(y) + a$ is in σ and $x + f(y) =_{\downarrow} x + f(y)$ is in Γ, we will unify $f(a)$ and $f(y)$, so that the conflict disappears.

Formally, the rule is:

Decomposition Instantiation

$$\frac{\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner}{\bigvee_{i=1}^{n} (\sigma \circ \theta_i \| \Upsilon\theta_i \| \Delta\theta_i \| \Psi\theta_i \| \urcorner) \quad \bigvee \quad \sigma \| \Upsilon \| \Delta'' \| \Psi \| \urcorner}$$

where

- there exists an assignment $[x \leftarrow s - t + S]$ in σ.
- $\{\theta_1, \cdots, \theta_n\}$ is a complete set of standard AG unifiers of $s \overset{?}{=} t$.
- $\Delta'' = \Delta \cup \{s - t \neq^? 0\}$.

Conditions:

- s and t are not variables and have the same uninterpreted function symbol on the top;
- x has a conflict at s or t in the premise.

Example 6.14 Suppose we have

$$\Gamma = \{f(a) + f(b) =_{\downarrow} x + f(y)\};$$
$$\sigma = [x \leftarrow f(y) - f(a) - f(b)].$$

Since x has a conflict at $f(y)$, we apply Decomposition Instantiation by unifying $f(a)$ and $f(y)$. Then get:

Branch 1:

$$\sigma_1 = [x \leftarrow -f(b), y \leftarrow a]$$
$$\Delta_1 = \&\emptyset$$

and Branch 2:

$$\sigma_2 = [x \leftarrow f(y) + f(a) + f(b)]$$
$$\Delta_2 = \{f(y) - f(a) \neq^{?} 0\}$$

σ_1 is an asymmetric unifier of Γ. We keep it and look at Branch 2.

In Branch 2, we still can apply Decomposition Instantiation since x has a conflict at $f(y)$, and $f(b) - f(y) \neq^? 0 \notin \Delta$. We unify $f(y)$ and $f(b)$ and get:

Branch 2.1

$$\sigma_{2.1} = [x \leftarrow -f(a), y \leftarrow b]$$
$$\Delta_{2.1} = \{f(y) - f(a) \neq^? 0\}$$

and Branch 2.2

$$\sigma_{2.2} = [x \leftarrow f(y) - f(a) - f(b)]$$
$$\Delta_{2.2} = \{f(y) - f(a) \neq^? 0, f(y) - f(b) \neq^? 0\}$$

$\sigma_{2.1}$ is another asymmetric unifier. $\sigma_{2.2}$ is not an asymmetric unifier. However, we cannot apply any rules so far to Branch 2.2.

Elimination Instantiation

The second instantiation rule is called Elimination Instantiation. It is used to solve the case that some original variable x has a conflict at some support variable v.

For example if $x \leftarrow v + b, y \leftarrow v + d$ is in σ, $x - y =_{\downarrow} x - y$ is in Γ, and $(v, a), (v, d) \in \Upsilon$, we will let $v \leftarrow 0$, so that the conflict disappears.

formally, the rule is:

Elimination Instantiation

$$\frac{[x \leftarrow \pm v + S] \cup \sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner}{([x \leftarrow S] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta \| \Psi\theta \| \urcorner}$$

where

$$\theta = \{v \leftarrow 0\}$$

Conditions:

- x has a conflict at v.

Note: Since we let $v \leftarrow 0$, all the pairs of the form (v, s) in Υ (resp. Ψ) will be removed from Υ (resp. Ψ).

Example 6.15 We continue Example 6.13. For problem $\Gamma = \{a + b =_{\downarrow} x + y\}$, we got four branches:

Branch 1.1

$$\sigma_{1.1} = [x \leftarrow v_3, \quad y \leftarrow v_3 + a + b, \quad v_1 \leftarrow v_3 + a + b, \quad v_2 \leftarrow v_3 + b];$$

$$\Upsilon_{1.1} = \{(v_3, -a), \quad (v_3, -b)\};$$

$$\Psi_{1.1} = \{(v_3, a), \quad (v_3, b)\}.$$

Branch 1.2

$$\sigma_{1.2} = [x \leftarrow b - v_2, \quad y \leftarrow v_2 + a, \quad v_1 \leftarrow v_2 + a];$$

$$\Upsilon_{1.2} = \{(v_2, -a), \quad (b, v_2)\};$$

$$\Psi_{1.2} = \{(v_2, a)\}.$$

Branch 2.1

$$\sigma_{2.1} = [x \leftarrow a - v_2, \quad y \leftarrow v_2 + b \quad v_1 \leftarrow v_2 + b];$$

$$\Upsilon_{2.1} = \{(v_2, a), \quad (v_2, -b)\};$$

$$\Psi_{2.1} = \{(v_2, b)\}.$$

and Branch 2.2

$$\sigma_{2.2} = [x \leftarrow a + b - v_1, \quad y \leftarrow v_1];$$

$$\Upsilon_{2.2} = \{(v_1, a), \quad (v_1, b)\};$$

$$\Psi_{2.2} = \emptyset.$$

For each branch, we have no rules except Elimination Instantiation applicable. In Branch 1.1, x has a conflict at v_3, so we let $v_3 \leftarrow 0$ and get

$$\sigma_{1.1.1} = [x \leftarrow 0, \quad y \leftarrow a + b].$$

In Branch 1.2, x has a conflict at v_2, so we let $v_2 \leftarrow 0$ and get

$$\sigma_{1.2.1} = [x \leftarrow b, \quad y \leftarrow a].$$

In Branch 2.1, x has a conflict at v_2, so we let $v_2 \leftarrow 0$ and get

$$\sigma_{2.1.1} = [x \leftarrow a, \quad y \leftarrow b].$$

In Branch 2.1, x has a conflict at v_1, so we let $v_1 \leftarrow 0$ and get

$$\sigma_{2.2.1} = [x \leftarrow a + b, \quad y \leftarrow 0].$$

We can see, $\sigma_{1.2.1}$ and $\sigma_{2.1.1}$ are two non-equivalent asymmetric unifiers but $\sigma_{1.1.1}$ and $\sigma_{2.2.1}$ are not asymmetric unifiers because neither x nor y should not be 0 in Γ.

6.5 Algorithm for Searching for the Complete Set of Asymmetric unifiers via $\mathfrak{I}_{AAG}$

Our inference system $\mathfrak{I}_{AAG}$ has three parts: Splitting, Branching and Instantiation.

First, we introduce three sub-algorithms, in which, the rules in our three parts will be applied separately. Then we will give an algorithm to show how to call three algorithms and output a complete set of asymmetric unifiers.

The first algorithm is called SPLITTING:

SPLITTING

`INPUT` :

- the unification problem Γ
- an AG unifier σ;
- a constraint set Υ;
- a disequation set Δ.
- a bound set Ψ.
- a boolean variable $\daleth$, which is false.

`OUTPUT`

- a set of quintuples Σ, which contains all the results by applying rules in $\Im_{AAG}$.

`PROCEDURE`:

1. Check whether σ is an asymmetric unifier of Γ:
    - If it is an asymmetric unifier, **return** $\{\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth\}$.
    - If it is not an asymmetric unifier, check whether the Splitting rule is applicable:
        - If Splitting is applicable, Apply Splitting to $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ and get $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$. **Return** SPLITTING$(\Gamma, \sigma', \Upsilon', \Delta', \Psi', \daleth')$.
        - If Splitting is not applicable, **return** BRANCHING$(\Gamma, \sigma, \Upsilon, \Delta, \Psi, \emptyset, \daleth)$

The second algorithm is called BRANCHING. Here, we will call an algorithm named **VariantNarrowing(Γ)**, which returns a set of quintuples $\Sigma = \{\sigma\|\emptyset\|\emptyset\|\emptyset\|false \mid \sigma$ is an asymmetric unifier of $\Gamma\}$.

BRANCHING

`INPUT:`

- the unification problem Γ;
- an AG unifier σ;
- a constraint set Υ;
- a disequation set Δ;
- a bound set Ψ;
- a set of quintuples Σ.
- a boolean variable $\daleth$.

`OUTPUT:`

- a set of quintuples Σ, which represents the result of applying rules in the Branching part.

`PROCEDURE:`

```
1. Let $\Sigma' = \{\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth\}$.
2. For each member $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ in $\Sigma'$:
    (a) Check whether Branching rules are applicable to $\sigma\|\Upsilon\|\Delta\|\Psi$.
```

    i. If the Branching rule is applicable, apply Non-Variable Branching to $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ and get $\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'$ and $\sigma''\|\Upsilon''\|\Delta''\|\Psi''\|\daleth''$, put $\sigma''\|\Upsilon''\|\Delta''\|\Psi''\|\daleth''$ into Σ' and let $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth = \sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'$, go back to 2(a). Otherwise,

    ii. If the Useless-Variable Branching rule is applicable, apply Useless-Variable Branching to $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ and get $\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'$ and $\sigma''\|\Upsilon''\|\Delta''\|\Psi''\|\daleth''$, put $\sigma''\|\Upsilon''\|\Delta''\|\Psi''\|\daleth''$ into Σ' and let $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth = \sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'$, go back to 2(a). Otherwise,

    iii. check whether σ is an asymmetric unifier of Γ. If yes, **return** $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$, otherwise add $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ to Σ.

3. For any element $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ of Σ, if $\daleth = true$, **return** $VariantNarrowing(\Gamma)$. otherwise **Return** $Instantiation(\Gamma, \Sigma)$.

The third algorithm is called INSTANTIATION:

INSTANTIATION

`INPUT`

- the unification problem Γ;
- a set of quintuples Σ.

`OUTPUT`:

- a set of quintuples Σ', which represents the result of applying rules in the Instantiation part.

`PROCEDURE`

Given an empty set of quintuples Σ', for each member $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ of Σ,

1. Check whether Decomposition Instantiation is applicable:
    - if it is, apply Decomposition Instantiation to $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ and get $\sigma_1\|\Upsilon_1\|\Delta_1\|\Psi_1\|\daleth_1, \cdots \sigma_n\|\Upsilon_n\|\Delta_n\|\Psi_n\|\daleth_n$. For each $\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i$, check whether σ_i is an asymmetric unifier of Γ:
        - if it is an asymmetric unifier, add it to Σ' and go to the next member.
        - if it is not, union SPLITTING$(\Gamma, \sigma_i, \Upsilon_i, \Delta_i, \Psi_i, false)$ with Σ'
2. Check whether Elimination Instantiation is applicable:
    - If it is applicable, apply Elimination Instantiation to $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ until it is not applicable and get $\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'$. Check whether σ' is an asymmetric unifier of Γ:
        - if it is, add the quintuple to Σ' and go to the next member.

```
        – if it is not, union BRANCHING($\Gamma, \sigma', \Upsilon', \Delta', \emptyset, false$) with $\Sigma'$.
    • If it is not, go to the next member.

**Return** $\Sigma'$.
```

Next, we give out our algorithm *SEARCHING* which will call the above three algorithms.

```
INPUT

  • An asymmetric unification problem $\Gamma$;

  • A standard unifier $\sigma$ of $\Gamma$;

PROCEDURE

  1. Let $\Sigma' = \text{SPLITTING}(\Gamma, \sigma, \emptyset, \emptyset, \emptyset, false)$

  2. Let $\Sigma_\sigma = \{\sigma \mid \sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner \in \Sigma\}$.

  3. Return $\Sigma_\sigma$.
```

6.6 Termination

For discussing the termination, soundness and completeness, we need an assumption: the algorithm VariantNarrowing(Γ) will terminate and generate a set of quintuples $\bigvee_i \sigma_i \| \emptyset \| \emptyset \| \emptyset \| false$, where all the σ_i forms the complete set of asymmetric unifiers of Γ.

So when we discuss the termination, soundness and completeness of our algorithm, we will omit the discussion related to VariantNarrowing.

We will use the following notation in the following sections.

- $\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner \Rightarrow_{\Im_{AAG}} \sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \urcorner'$, means $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \urcorner'$ is deduced from $\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner$ by one step.

- $\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner \overset{*}{\Rightarrow}_{\Im_{AAG}} \sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \urcorner'$, means $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \urcorner'$ is deduced from

$\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ by zero or more steps.

- $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth \overset{+}{\Rightarrow}_{\Im_{AAG}} \sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'$ means $\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\daleth'$ is deduced from $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ by one or more steps.

As we can see in the inference rules, some rules divide $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ into two (Branching rules) or even more quintuples (Decomposition Instantiation). So for a quintuple $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$, after applying some inference rules, the result may be a disjunction of quintuples $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$, not just one quintuple. So we give the following notation:

- $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth \Rightarrow_{\Im_{AAG}} \bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$,

  where $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$ is a disjunction of quintuples, means after applying some inference rule to $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ once, $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ becomes $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$.

- $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth \overset{+}{\Rightarrow}_{\Im_{AAG}} \bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$,

  where $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$ is a disjunction of quintuples, means after applying some inference rules once or more than once, $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ becomes $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$, namely $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$ is the set of all branches we get after applying one or more than once the rules in $\Im_{AAG}$ to $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$.

- $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth \overset{*}{\Rightarrow}_{\Im_{AAG}} \bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$,

  where $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$ is a disjunction of quintuples, means after zero or more steps, $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$ becomes $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$, namely $\bigvee_{i=1}^{n}(\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\daleth_i)$ is the set of all branches we get after applying zero or more times the rules in $\Im_{AAG}$ to $\sigma\|\Upsilon\|\Delta\|\Psi\|\daleth$.

We will give a measurement of a quintuple and prove it will decrease every time we apply some inference rule to the quintuple, such that the termination of inference system $\mathfrak{I}_{AAG}$ is proven.

Before formally giving the measurement, we need some preparation. Let us start with some definitions and lemmas.

Definition 6.16 (Equivalent and Non-Equivalent Terms) *Let Ω be a set of identities. Let t and s be two terms. If $\Omega \models_{AG} t = s$, we say t and s are* equivalent *in Ω and denote it by $t \simeq_\Omega s$. Otherwise, we say t and s are two* non-equivalent terms *in Ω and denote it by $t \not\simeq_\Omega s$.*

$\simeq_\Omega$ is an equivalent relation.

Example 6.17 Let $\Omega = \{f(x) - f(a) = 0, v_1 - x - y = 0, v_2 - x - y = 0\}$. Then we will have the following equivalent terms:

- $f(x) \simeq_\Omega f(a)$.
- $x \simeq_\Omega a$.
- $g(x) \simeq_\Omega g(a)$, where g is some uninterpreted function symbol.
- $v_1 \simeq_\Omega x + y$;
- $v_1 \simeq_\Omega v_2$;
- $f(v_1) \simeq_\Omega f(v_2)$;
- $\cdots$

For an asymmetric unification problem Γ, we use $[\![\Gamma]\!]$ to denote the following set of identities:

$$\{s =_{AC} t \mid s =_{\downarrow} t \in \Gamma\}$$

Without ambiguity, for a substitution σ we use $[\![\sigma]\!]$ to denote the following set of identities:

$$\{x =_{AC} T \mid x \leftarrow T \in \sigma\}$$

Definition 6.18 (Semi-Uninterpreted Function Terms) *We say a term t is an* semi-uninterpreted function term *if*

- *t's top function symbol is an uninterpreted function symbol or $-$; and*
- *t is not a variable or an inverse of a variable; and*
- *t is not a constant or an inverse of a constant.*

Example 6.19 $f(x+y)$, $-f(a)$ and $f(g(x,a))$ are semi-uninterpreted function terms. x, a, $-y$ and $x+f(a)$ are not semi-uninterpreted function terms.

Let $Terms(\Gamma) = \{t \mid t$ or $-t$ occurs in Γ as a term or subterm $\}$, $Terms(\sigma) = \{t \mid t$ or $-t$ occurs in the range of σ as a term or subterm$\}$.

Example 6.20 if $\Gamma = \{x - f(a+y) = 0\}$, then $Terms(\Gamma) = \{x, -x, f(a+y), -f(a+y), a, -a, y, -y\}$

Given a set of terms Ξ and a set of identities Ω, the equivalence class of a term t in Ξ is denoted $[t]_\Omega$ and defined as the set:

$$[t]_\Omega^\Xi = \{s \mid s \in \Xi \text{ and } s \simeq_\Omega t\}$$

We let $nET(\Xi, \Omega) = \{[s]_{\Omega}^{\Xi} \mid s$ is an semi-uninterpreted function term$\}$ Since here Ξ and Ω is clear, we will simplify $[s]_{\Omega}^{\Xi}$ to $[s]$. So Here $nET(\Xi, \Omega)$ can be regarded as the set of equivalence classes of semi-uninterpreted function terms in Ξ up to the equivalence relation Ω.

Example 6.21

$$\begin{aligned}\Gamma = \{&f(g(x, y + f(a + c)) + y) =_{\downarrow} g(x, y),\\ &f(a + c) - f(z) =_{\downarrow} 0, \}.\end{aligned}$$

Then

$$\begin{aligned}nET(Terms(\Gamma), [\![\Gamma]\!]) = \{&[f(g(x, y + f(a + c)) + y)], \quad [-f(g(x, y + f(a + c)) + y)],\\ &[g(x, y + f(a + c))], \quad [-g(x, y + f(a + c))], \quad [f(z)], \quad [-f(z)]\}.\end{aligned}$$

Here $f(a + c) \in [f(z)]$ and $-f(a + c) \in [-f(z)]$ since $f(a + c) - f(z) = 0 \models f(z) \simeq_{\Omega} f(a + c)$.

Example 6.22

$$\begin{aligned}\sigma = \{&x \leftarrow f(f(v_1 + v_2)) - f(v_1) + g(v_2, a),\\ &y \leftarrow f(v_1 + v_2) + f(v_2) - g(b, a)\}.\end{aligned}$$

Then

$$\begin{aligned} nET(Terms(\sigma), [\![\sigma]\!]) = \{ & [f(f(v_1+v_2))], \quad [f(v_1+v_2)], \quad [f(v_1)], \\ & [g(v_2,a)], \quad [f(v_2)], \quad [g(b,a)] \\ & [-f(f(v_1+v_2))], \quad [-f(v_1+v_2)], \quad [-f(v_1)], \\ & [-g(v_2,a)], \quad [-f(v_2)], \quad [-g(b,a)]\}. \end{aligned}$$

Definition 6.23 (replacement of terms) *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities. Let s and t be two semi-uninterpreted function terms such that $[s] \in nET(\Xi_1, \Omega_1)$ and $[t] \in nET(\Xi_2, \Omega_2)$. If $\Omega_2 \models_{AG} \Omega_1$ and $s \simeq_{\Omega_2} t$, we call $[t]$ a* replacement *of $[s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$. In this case, we write $[t] \twoheadleftarrow [s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$.*

Sometimes, we will omit "*from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$*" if it is clear. Next, we will give some properties of replacements.

Lemma 6.24 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities satisfying $\Omega_2 \models \Omega_1$. Let s and t be two semi-uninterpreted function terms and $s \in \Xi_1$ and $t \in \Xi_2$. Let $s' \in [s]$ and $t' \in [t]$. If $s \simeq_{\Omega_2} t$ then $s' \simeq_{\Omega_2} t'$.*

This lemma shows that the property of replacements is an invariant of $\simeq_{\Omega}$.

Proof. Since for all t_j, we have $t \simeq_{\Omega_2} t'$, all we need to prove is $s' \simeq_{\Omega_2} t$.

Because $s' \simeq_{\Omega_1} s$ and $\Omega_2 \models_{AG} \Omega_1$, $s' \simeq_{\Omega_2} s$. And we already have $s \simeq_{\Omega_2} t$, so $s' \simeq_{\Omega_2} t$. □

Lemma 6.25 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities satisfying $\Omega_2 \models \Omega_1$. Let s, t_1 and t_2 be three semi-uninterpreted function terms such that $[s] \in nET(\Xi_1, \Omega_1)$ and $[t_1], [t_2] \in nET(\Xi_2, \Omega_2)$. If $[t_1] \twoheadleftarrow [s]$ and $[t_2] \twoheadleftarrow [s]$, then $[t_1] = [t_2]$, namely $t_1 \simeq_{\Omega_2} t_2$.*

Proof. Since $[t_1] \twoheadleftarrow [s]$ and $[t_2] \twoheadleftarrow [s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$, $s \simeq_{\Omega_2} t_1$ and $s \simeq_{\Omega_2} t_2$. So $t_1 \simeq_{\Omega_2} t_2$. $\square$

Lemma 6.26 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities satisfying $\Omega_2 \models \Omega_1$. Let $s_1, \cdots s_n$ and $t_1 \cdots t_n$ be semi-uninterpreted function terms such that $[s_i] \in nET(\Xi_1, \Omega_1)$ and $[t_i] \in nET(\Xi_2, \Omega_2)$. If*

- *$[t_i] \twoheadleftarrow [s_i]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$;*
- *for some semi-uninterpreted function symbol f, we have $[f(s_1, \cdots, s_n)] \in nET(\Xi_1, \Omega_1)$ and $[f(t_1, \cdots, t_n)] \in nET(\Xi_2, \Omega_2)$,*

then $[f(t_1, \cdots, t_n)]$ is a replacement of $[f(s_1, \cdots, s_n)]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$.

Proof. Since $[t_i] \twoheadleftarrow [s_i]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$, we have $t_i \simeq_{\Omega_2} s_i$. Hence $f(t_1, \cdots, t_n) \simeq_{\Omega_2} f(s_1, \cdots, s_n)$. $\square$

Lemma 6.27 *Let Ξ_1 and Ξ_2 be two sets of terms, and Ω_1 and Ω_2 be two sets of identities. If*

- *$\Omega_2 \models_{AG} \Omega_1$; and*
- *for every $t \in nET(\Xi_2, \Omega_2)$, there exists $s \in nET(\Xi_1, \Omega_1)$, such that $[t] \twoheadleftarrow [s]$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$,*

then $|nET(\Xi_2, \Omega_2)| \leq |nET(\Xi_1, \Omega_1)|$.

Proof. Let $t, t' \in nET(\Xi_2, \Omega_2)$ and $s, s' \in nET(\Xi_1, \Omega_1)$, such that $[t] \twoheadleftarrow [s]$ and $[t'] \twoheadleftarrow [s']$. Then it is enough to prove if $t \not\simeq_{\Omega_2} t'$, then $s \not\simeq_{\Omega_1} s'$.

However, if $s' \simeq_{\Omega_1} s$, from Lemma 6.24, we will have $t \simeq_{\Omega_2} t'$. This is a contradiction. $\square$

If $nET(\Xi_1, \Omega_1)$ and $nET(\Xi_2, \Omega_2)$ satisfies the conditions in Lemma 6.27, we write $nET(\Xi_2, \Omega_2) \preceq nET(\Xi_1, \Omega_1)$.

Lemma 6.28 *If $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_2, \Omega_2)$ and $nET(\Xi_2, \Omega_2) \preceq nET(\Xi_1, \Omega_1)$, then $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_1, \Omega_1)$.*

Proof. First, because $\Omega_3 \models_{AG} \Omega_2$ and $\Omega_2 \models_{AG} \Omega_1$, $\Omega_3 \models_{AG} \Omega_1$.

Second, let $[t] \in nET(\Xi_3, \Omega_3)$, then since $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_2, \Omega_2)$, there exists an equivalence class $[t']$ in $nET(\Xi_2, \Omega_2)$ such that $[t] \twoheadleftarrow [t']$ from $nET(\Xi_2, \Omega_2)$ to $nET(\Xi_3, \Omega_3)$. And we have $t' \simeq_{\Omega_3} t$

Since $nET(\Xi_2, \Omega_2) \preceq nET(\Xi_1, \Omega_1)$ and $[t'] \in nET(\Xi_2, \Omega_2)$, there exists an equivalence class $[t'']$ in $nET(\Xi_1, \Omega_1)$, such that $[t''] \twoheadleftarrow [t']$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_2, \Omega_2)$ and $t' \simeq_{\Omega_2} t''$. Since $\Omega_3 \models_{AG} \Omega_2$, $t' \simeq_{\Omega_3} t''$. So $t \simeq_{\Omega_3} t''$.

Hence for every $[t] \in nET(\Xi_3, \Omega_3)$, there exists a $[t''] \in nET(\Xi_1, \Omega_1)$, such that $[t] \twoheadleftarrow [t'']$ from $nET(\Xi_1, \Omega_1)$ to $nET(\Xi_3, \Omega_3)$.

So $nET(\Xi_3, \Omega_3) \preceq nET(\Xi_1, \Omega_1)$. □

From the above lemma, we know the relation "$\preceq$" is transitive.

In Decomposition Instantiation, we need to call some standard AG unification algorithm. For proving termination, we require the standard AG unification algorithm to have a property that we will give below.

Chapter 4 introduced a standard AG unification algorithm $\mathfrak{A}_{AG}$ which will generate a complete set of unifiers for a given AG unification problem Γ. The inference system $\mathfrak{I}_{AG}$ has three rules *Purification, Variable Substitution*, and *Decomposition.* $\mathfrak{A}_{AG}$ will apply inference rules to a triple: $\Gamma \| \Delta \| \Lambda$, where Γ is a set of equations to be solved, Δ is set of disequations which has the same meaning as in $\mathfrak{I}_{AAG}$ and Λ is a set of equations which are solved. Roughly, $\mathfrak{A}_{AG}$ has the following procedure:

1. Convert every $s \overset{?}{=} t \in \Gamma$ to $s - t \overset{?}{=} 0$.

2. Apply Purification until it cannot be applied any more.

3. Apply Variable Substitution until it cannot be applied any more. In this step, the algorithm of **UnconstrainedReduce** is applied repeatedly.

4. Apply Decomposition and go back to Step 3.

5. At last, there is a set of triples generated. For each triple, if Γ is empty, output a unifier from Λ; otherwise, abort that triple.

For this algorithm, we have the following lemma:

Lemma 6.29 *Let Γ be an AG unification problem. Then for the standard unification algorithm $\mathfrak{A}_{AG}$, we have:*

- *$\mathfrak{A}_{AG}(\Gamma)$ outputs a complete set of standard AG unifiers of Γ, which is $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$;*
- *For each σ_i, $nET(Terms(\sigma_i), [\![\sigma_i]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$.*

Before proving this lemma, we note that $\mathfrak{A}_{AG}$ will call the syntactic unification procedure $\mathfrak{I}_{SYN}$ which has the following inference rules:

1. $\Gamma \cup \{t \overset{?}{=} t\} \Rightarrow \Gamma$;

2. $\Gamma \cup \{f(s_0, \ldots, s_k) \overset{?}{=} f(t_0, \ldots, t_k)\} \Rightarrow \Gamma \cup \{s_0 \overset{?}{=} t_0, \ldots, s_k \overset{?}{=} t_k\}$;

3. $\Gamma \cup \{f(s_0, \ldots, s_k) \overset{?}{=} g(t_0, \ldots, t_m)\} \Rightarrow \bot$ if $f \neq g$ or $k \neq m$;

4. $\Gamma \cup \{f(s_0, \ldots, s_k) \overset{?}{=} x\} \Rightarrow \Gamma \cup \{x \overset{?}{=} f(s_0, \ldots, s_k)\}$;

5. $\Gamma \cup \{x \overset{?}{=} t\} \Rightarrow \Gamma\{x \leftarrow t\} \cup \{x \overset{?}{=} t\}$ if $x \in Vars(\Gamma)$ and $x \notin Vars(t)$;

6. $\Gamma \cup \{x \overset{?}{=} f(s_0, \ldots, s_k)\} \Rightarrow \bot$ if $x \in Vars(f(s_0, \ldots, s_k))$.

If no rules are applicable to a set of equations Γ, then $\mathfrak{I}_{SYN}$ will output $\sigma = \{x \leftarrow T \mid x \overset{?}{=} T \in \Gamma\}$.

$\mathfrak{I}_{SYN}$ will generate the most general unifier σ of a syntactic unification problem Γ if Γ is unifiable. Then we have the following lemma:

Lemma 6.30 *Let Γ be a standard AG unification problem but not containing $+$ such that Γ is unifiable. Then:*

- *the most general unifier σ generated by $\mathfrak{I}_{SYN}(\Gamma)$ contains no fresh variables.*
- *$nET(Terms(\sigma), [\![\sigma]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$.*

Proof. Since Γ contains no $+$, this problem is a syntactic unification problem. So we can apply $\mathfrak{I}_{SYN}$ to get a most general unifier.

Since each inference does not generate any fresh variables, $\mathfrak{I}_{SYN}(\Gamma)$ will not generate any fresh variables.

For proving $nET(Terms(\sigma), [\![\sigma]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$, we only need to prove for every non-failure inference rule $\Gamma_1 \models_{AG} \Gamma_2$, we have $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Since there are no AG function symbols in Γ, we will omit AG from $\models_{AG}$.

Rule 1: $t \simeq t$ is a trivial identity. $Terms(\Gamma_2) \subseteq Terms(\Gamma_1)$ and $[\![\Gamma_2]\!] = [\![\Gamma_1]\!]$. so for every $s \in Terms(\Gamma_2)$, we have $[s] \twoheadleftarrow [s]$.

Hence $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 2: Since $\{s_i \simeq t_i \mid i = 0, \cdots k\} \models f(s_0, \cdots, s_k) \simeq f(t_0, \cdots, t_k)$, we have $[\![\Gamma_2]\!] \models [\![\Gamma_1]\!]$. Since $f(s_0, \cdots, s_k)$, $f(t_0, \cdots, t_k)$ are $\in Terms(\Gamma_1)$, $s_i \in Terms(\Gamma_1)$ and $t_i \in Terms(\Gamma_1)$ for $i = 0$ to k. So we can get $[s_i] \twoheadleftarrow [s_i]$ and $[t_i] \twoheadleftarrow [t_i]$.

Hence $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 4: $\simeq$ is symmetric, $Terms(\Gamma_2) = Terms(\Gamma_1)$ and $[\![\Gamma_2]\!] = [\![\Gamma_1]\!]$, so for every $s \in Terms(\Gamma_2)$, we have $[s] \twoheadleftarrow [s]$.

Hence $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 5: Let $s = s' \in [\![\Gamma_1]\!]$. If $s = s'$ is $x = t$, then we have $[\![\Gamma_2]\!] \models s = s'$.

Otherwise, then $s[x \leftarrow t] = s'[x \leftarrow t]$ is in $[\![\Gamma_2]\!]$. If x does not occur in s and s', then $s = s' \in [\![\Gamma_2]\!]$. If x is in $Vars(s)$ or $Vars(s')$, then let $s = s[x]$ or $s' = s'[x]$, so we will have $s[t] = s'[t] \in [\![\Gamma_2]\!]$. Because we also have $x = t \in [\![\Gamma_2]\!]$, $[\![\Gamma_2]\!] \models s[x] = s'[x]$. Hence $[\![\Gamma_2]\!] \models [\![\Gamma_1]\!]$.

For the other condition, we need to prove for every equivalence class $[s]$ in $nET(Terms(\Gamma_2), [\![\Gamma_2]\!])$, we always can find some equivalence class $[s']$ in $nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$ such that $[s] \twoheadleftarrow [s']$.

We know $\Gamma_2 = \Gamma_1\sigma \cup \{x \stackrel{?}{=} t\}$ where $\sigma = [x \leftarrow t]$. Suppose s has the form of $s''\sigma$, where $[s''] \in nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

If x does not occur in s'', then $s''\sigma = s'' = s$, and $[s'']$ is also in $nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$, so $[s] \twoheadleftarrow [s'']$ since $s \simeq_{[\![\Gamma_2]\!]} s''$.

If x occurs in s'', let s'' be $s''[x]$. Then we have $s''[x]\sigma = s''[t]$. Since we have $x \stackrel{?}{=} t \in \Gamma_2$, $s''[x] \simeq_{[\![\Gamma_2]\!]} s''[t]$. So $[s''[x]]\sigma \twoheadleftarrow [s''[x]]$ which is $[s''[t]] \twoheadleftarrow [s''[x]]$.

Hence in this case $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

Rule 6: As same as Rule 3.

In summary $nET(Terms(\Gamma_2), [\![\Gamma_2]\!]) \preceq nET(Terms(\Gamma_1), [\![\Gamma_1]\!])$.

□

We know $\mathfrak{A}_{AG}(\Gamma)$ outputs a complete set of standard AG unifiers of Γ, which is $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$. Let us look at the procedure of $\mathfrak{A}_{AG}$:

The first step of $\mathfrak{A}_{AG}$ converts $s \stackrel{?}{=} t$ into $s - t \stackrel{?}{=} 0$. $Terms(\Gamma)$ and $[\![\Gamma]\!]$ do not change because $Terms(\Gamma)$ contains all the semi-uninterpreted function terms and their inverses. So we only need to consider the other steps.

$\mathfrak{A}_{AG}$ starts from $\Gamma \| \emptyset \| \emptyset$ and if some unifier is found, then $\mathfrak{A}_{AG}$ ends at $\emptyset \| \Delta \| \Lambda$ and outputs $\sigma_\Lambda = \{x \leftarrow T \mid x \stackrel{?}{=} T \in \Lambda\}$. So $Terms(\Lambda) = Terms(\sigma_\Lambda)$ and $[\![\Lambda]\!] = [\![\sigma_\Lambda]\!]$.

Hence for proving Lemma 6.29, we need to prove $nET(Terms(\Lambda), [\![\Lambda]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$. We will show this is true for each rule.

Proof. Let $\Gamma_1 \| \Delta_1 \| \Lambda_1$ and $\Gamma_2 \| \Delta_2 \| \Lambda_2$ be two sets of quintuples, such that $\Gamma_1 \| \Delta_1 \| \Lambda_1 \Rightarrow_{\mathfrak{I}_{AG}} \Gamma_2 \| \Delta_2 \| \Lambda_2$ by applying some inference rule from $\mathfrak{I}_{AG}$. We want to prove $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Case 1. This rule is Purification:

$$\frac{\Gamma \cup \{S + t[s] \stackrel{?}{=} 0\} \| \Delta \| \Lambda}{\Gamma \cup \{S + t[v] \stackrel{?}{=} 0\} \cup \{v - s \stackrel{?}{=} 0\} \| \Delta \| \Lambda}$$

where Γ is a set of equations, S a term, t a simple term, s a proper subterm of t with $+$ on the top, and v is a fresh variable.

The difference between $\Gamma_1 \cup \Lambda_1$ and $\Gamma_2 \cup \Lambda_2$ is that $S + t[s] \stackrel{?}{=} 0 \in \Gamma_1$ but not in Γ_2 and $S + t[v] \stackrel{?}{=} 0$ and $v - s \stackrel{?}{=} 0$ are in Γ_2 but not in Γ_1.

For the first condition in Lemma 6.27, it is enough to prove $[\![\{S + t[v] \stackrel{?}{=} 0, v - s \stackrel{?}{=} 0\}]\!] \models_{AG} [\![\{S + t[s] \stackrel{?}{=} 0\}]\!]$. This is true because $v - s \stackrel{?}{=} 0 \models_{AG} v \stackrel{?}{=} s$.

Let $t'[v]$ be a semi-uninterpreted function subterm in $t[v]$. So $t'[s] \in Terms(t[s]) \subseteq Terms(\Gamma_1 \cup \Lambda_1)$. Because we have $v + s \stackrel{?}{=} 0 \in \Gamma_2$, $t'[v] \simeq_{[\![\Gamma_2 \cup \Lambda_2]\!]} t'[s]$, which means $t'[v] \twoheadleftarrow t'[s]$ from $nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$ to $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!])$

Hence $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Case 2. The second rule is called Variable Substitution. Before discussing this part, we need to discuss **UnconstrainedReduce**, which is used in Variable Substitution. We want to prove when we get $\Gamma_2 \cup \Lambda_2$ after applying UnconstrainedReduce on $\Gamma_1 \cup \Lambda_1$, we have $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Referring to the UnconstrainedReduce in Chapter 4, the following two rules are essentially applied repeatedly:

Rule 1:

$$\frac{(\Gamma \cup \{c_1x_1 + \cdots + c_{min}x_{min} + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m \overset{?}{=} 0\})\|\Delta\|\Lambda\|\Psi}{\Gamma\theta\|\Delta\theta\|(\Lambda\theta \cup [\theta])}$$

where θ is

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

where x' is a fresh variable and others are all old variables. c_it_j is an abbreviation of $t_j + \cdots + t_j$, where t_j occurs c_i times.

Rule 2:

$$\frac{(\Gamma \cup \{cx + T \overset{?}{=} 0, \quad c'x + S \overset{?}{=} 0\})\|\Delta\|\Lambda\|\Psi}{\Gamma \cup \{cx + T \overset{?}{=} 0, \quad -c'T + cS \overset{?}{=} 0\}\|\Delta\|\Lambda}$$

We can see in Rule 1,

$$c_1x_1 + \cdots + c_{min}x_{min} + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m \overset{?}{=} 0$$

is replaced by

$$(c_1x_1 + \cdots + c_{min}x_{min} + \cdots + c_nx_n + c'_1t_1 + \cdots + c'_mt_m)\theta \overset{?}{=} 0$$

and $[\theta]$, which are

$$(c_1 \mod c_{min})x_1 + \cdots + (c_{min-1} \mod c_{min})x_{min-1}$$
$$c_{min}x' + (c_{min+1} \mod c_{min+1})x_{min+1} + \cdots (c_n \mod c_{min})x_n$$
$$(c'_1 \mod c_{min})t_1 + \cdots + (c'_m \mod c_{min})t_m \overset{?}{=} 0$$

and

$$\begin{aligned} x_{min} - (\ x' + (-\lfloor\frac{c_1}{c_{min}}\rfloor x_1) + \cdots + (-\lfloor\frac{c_{min-1}}{c_{min}}\rfloor x_{min-1}) \\ + (-\lfloor\frac{c_{min+1}}{c_{min}}\rfloor x_{min+1}) + \cdots + (-\lfloor\frac{c_n}{c_{min}}\rfloor x_n) + \\ (-\lfloor\frac{c'_1}{c_{min}}\rfloor t_1) + \cdots + (-\lfloor\frac{c'_m}{c_{min}}\rfloor t_m)) \overset{?}{=} 0 \end{aligned}$$

And in Rule 2, $\{cx + T \overset{?}{=} 0,\ \ c'x + S \overset{?}{=} 0\}$ is replaced by $\{cx + T \overset{?}{=} 0,\ \ -c'T + cS \overset{?}{=} 0\}$.

These procedure is similar to the procedure in Purification. So using a similar methods as in Case 1, we can prove $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

The rule of **Variable Substitution** applies these two rules together exhaustively. Since for each step, $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$, $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$ is true for Variable Substitution too.

Note: We here only consider the general cases of UnconstrainedReduce. There are some simpler cases like $c_{min} = 1$, which are easier to be proven.

Case 3. The rule is Decomposition Instantiation:

If $f(s_1, s_2, \cdots, s_m) - f(t_1, t_2, \cdots, t_m) \neq^? \notin \Delta$,

$$\frac{\Gamma \cup \{\mathbf{S} + cf(s_1, s_2, \cdots, s_m) + df(t_1, t_2, \cdots, t_m) \overset{?}{=} 0\} \| \Delta \| \Lambda}{(\Gamma\sigma \cup \{\mathbf{S} + (c+d)f(s_1, s_2, \cdots, s_m) \overset{?}{=} 0\}\sigma) \| (\Delta\sigma) \| (\Lambda\sigma \cup [\sigma]) \qquad \bigvee \qquad \Gamma'_1 \| \Delta'_1 \| \Lambda}.$$

where (i) $\sigma = mgu(s_1 \overset{?}{=} t_1, s_2 \overset{?}{=} t_2, \cdots, s_m \overset{?}{=} t_m)$ (ii) $\Gamma'_1 = \Gamma \cup \{\mathbf{S} + f(s_1, s_2, \cdots, s_m) +$

$f(t_1, t_2, \cdots, t_m) \overset{?}{=} 0\}$. (iii) $\Delta_1' = \Delta \cup \{f(s_1, s_2, \cdots, s_m) - f(t_1, t_2, \cdots, t_m) \neq^? 0\}$

For the first branch, we know σ is the most general unifier of the syntactic unification problem $\{s_1 \overset{?}{=} t_1, s_2 \overset{?}{=} t_2, \cdots, s_m \overset{?}{=} t_m\}$. Let this syntactic unification problem be Γ. From Lemma 6.30, $nET(Terms(\sigma), [\![\sigma]\!]) \preceq nET(Terms(\Gamma_3), [\![\Gamma_3]\!])$. Recall $[\sigma] = \{x \overset{?}{=} T \parallel x \leftarrow T \in \sigma\}$, we have $nET(Terms([\sigma]), [\![[\sigma]]\!]) \preceq nET(Terms(\Gamma_3), [\![\Gamma_3]\!])$. Let $\sigma = [x_1 \leftarrow u_1, \cdots, x_n \leftarrow u_n]$.

First we prove $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{AG} [\![\Gamma_1 \cup \Lambda_1]\!]$.

If $T[x_1, \cdots, x_n] = 0 \in [\![\Gamma_1 \cup \Lambda_1]\!]$, then $T[x_1, \cdots, x_n]\sigma = 0 \in [\![\Gamma_2 \cup \Lambda_2]\!]$, which is $T[u_1, \cdots, u_n] = 0 \in [\![\Gamma_2 \cup \Lambda_2]\!]$. Because

$$\begin{aligned} &\{T[u_1, \cdots, u_n] = 0, x_1 - u_1 = 0, \cdots, x_n - u_n = 0\} \\ &\models_{AG} \{T[x_1, \cdots, x_n] = 0\} \end{aligned}$$

and

$$
\begin{aligned}
&\{S + (c+d)f(s_1, s_2, \cdots, s_m) = 0, x_1 - u_1 = 0, \cdots, x_n - u_n = 0\} \\
&\models_{AG} \{S + (c+d)f(s_1, s_2, \cdots, s_m) = 0, s_1 = t_1, \cdots, s_n = t_n\} \\
&\models_{AG} \{S + (c+d)f(s_1, s_2, \cdots, s_m) = 0, f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n)\} \\
&\models_{AG} \{S + (c+d)f(s_1, s_2, \cdots, s_m) = 0, f(s_1, \cdots, s_n) - f(t_1, \cdots, t_n) = 0\} \\
&\models_{AG} \{S + (c+d)f(s_1, s_2, \cdots, s_m) + df(t_1, \cdots, t_n) - df(t_1, \cdots, t_n) = 0, \\
&\qquad f(s_1, \cdots, s_n) - f(t_1, \cdots, t_n) = 0\} \\
&\models_{AG} \{S + cf(s_1, s_2, \cdots, s_m) + df(s_1, \cdots, s_n) = 0\}
\end{aligned}
$$

we have $[\![\Gamma_2 \cup \Lambda_2]\!] \models_{AG} [\![\Gamma_1 \cup \Lambda_1]\!]$.

Then we use a similar analysis to Case 1, and we get that for every equivalence class $[t]$ in $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!])$, there exists an equivalence class $[s]$ in $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!])$ such that $[t] \twoheadleftarrow [s]$. Hence $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

For the second branch, $\Gamma_1 \cup \Lambda_1 = \Gamma_2 \cup \Lambda_2$, so $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$.

Hence, we get if $\Gamma_1 \| \Delta_1 \| \Lambda_1 \Rightarrow_{\mathfrak{I}_{AG}} \Gamma_2 \| \Delta_2 \| \Lambda_2$ by applying some inference rule from $\mathfrak{I}_{AG}$ then $nET(Terms(\Gamma_2 \cup \Lambda_2), [\![\Gamma_2 \cup \Lambda_2]\!]) \preceq nET(Terms(\Gamma_1 \cup \Lambda_1), [\![\Gamma_1 \cup \Lambda_1]\!])$. From Lemma 6.28, we can get:

If the algorithm starts from $\Gamma \| \emptyset \| \emptyset$ and if some unifier is found, then it ends at $\emptyset \| \Delta \| \Lambda$, we have $nET(Terms(\emptyset \cup \Lambda), [\![\emptyset \cup \Lambda]\!]) \preceq nET(Terms(\Gamma \cup \emptyset), [\![\Gamma \cup \emptyset]\!])$, which is $nET(Terms(\Lambda), [\![\Lambda]\!]) \preceq nET(Terms(\Gamma), [\![\Gamma]\!])$.

□

Lemma 6.31 *Let $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ be a set quintuple. If there exists another $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$, such that $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth \Rightarrow_{\Im_{AAG}} \sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$ via Decomposition Instantiation by applying the standard AG unification algorithm with the property in Lemma 6.29, then $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq nET(Terms(\sigma), [\![\sigma]\!])$ and $|nET(Terms(\sigma'), [\![\sigma']\!])| < |nET(Terms(\sigma), [\![\sigma]\!])|$.*

Proof. From Decomposition Instantiation, $\sigma' = \sigma \circ \delta$, where δ is a unifier of $s \overset{?}{=} t$, where t and s are in $Terms(\sigma)$.

First we prove $[\![\sigma']\!] \models_{AG} [\![\sigma]\!]$. Let $\delta = [v_1 \leftarrow T_1, \cdots, v_n \leftarrow T_n]$. Then if $T[v_1, \cdots, v_n] = 0 \in [\![\sigma]\!]$, then $T[v_1, \cdots, v_n]\delta = 0 \in [\![\sigma']\!]$ which is $T[T_1, \cdots, T_n] = 0 \in [\![\sigma']\!]$. And $v_i - T_i = 0 \in [\![\sigma']\!]$ since $[\![\delta]\!] \subseteq [\![\sigma']\!]$. Because $\{T[T_1, \cdots, T_n] = 0, v_i - T_i = 0\} \models_{AG} T[v_1, \cdots, v_n] = 0$, $[\![\sigma']\!] \models_{AG} T[v_1, \cdots, v_n] = 0$. Hence $[\![\sigma']\!] \models_{AG} [\![\sigma]\!]$.

For the second condition of Lemma 6.27. Let $u \in Terms(\sigma \circ \delta)$, then u has two possibilities:

1. $u = u'\delta$, where $u' \in Terms(\sigma)$. Similar to the proof above, $u \twoheadleftarrow u'$ from $nET(Terms(\sigma'), [\![\sigma']\!])$ to $nET(Terms(\sigma), [\![\sigma]\!])$.

2. $u \in Terms(\delta)$. From Lemma 6.29, $nET(Terms(\delta), [\![\delta]\!]) \preceq nET(Terms(s \overset{?}{=} t), [\![s \overset{?}{=} t]\!])$. So there exists an equivalence class $[u']$ such that $[u] \twoheadleftarrow [u']$ from $nET(Terms(\delta), [\![\delta]\!])$ to $nET(Terms(s \overset{?}{=} t), [\![s \overset{?}{=} t]\!])$. Since $[\![\delta]\!] \models_{AG} u - u' = 0$ and $[\![\sigma \circ \delta]\!] \models_{AG} [\![\delta]\!]$, $[\![\sigma \circ \delta]\!] \models_{AG} u - u'$. But $u' \in Terms(\sigma)$, so $[u] \twoheadleftarrow [u']$ from $nET(Terms(\sigma'), [\![\sigma']\!])$ to $nET(Terms(\sigma), [\![\sigma]\!])$.

In summary, $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq nET(Terms(\sigma), [\![\sigma]\!])$. From Lemma 6.27, we have $|nET(Terms(\sigma''), [\![\sigma']\!])| \leq |nET(Terms(\sigma), [\![\sigma]\!])|$. But in $nET(Terms(\sigma), [\![\sigma]\!])$, $[s] \neq [t]$, which means $s \not\simeq_{[\![\sigma]\!]} t$ and in $nET(Terms(\sigma'), [\![\sigma']\!])$, $[s\delta] = [t\delta]$, which means $s\delta \simeq_{[\![\sigma']\!]} t\delta$. Hence $|nET(Terms(\sigma'), [\![\sigma']\!])| < |nET(Terms(\sigma), [\![\sigma]\!])|$

□

Then, next, we can prove the termination of $\mathfrak{I}_{AAG}$.

Theorem 6.32 *Let* $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner$ *be a set quintuple. Then there exists a set quintuple* $\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\urcorner'$*, such that* $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner \overset{*}{\Rightarrow}_{\mathfrak{I}_{AAG}} \sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\urcorner'$ *and* σ' *is an asymmetric unifier or no rules in* $\mathfrak{I}_{AAG}$ *are applicable to* $\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\urcorner'$*.*

Proof. First, we introduce several well-founded orderings by introducing some notation.

- $Var(\Gamma)$-the set of variables in the unification problem Γ(original variables).
- $Var(dom(\sigma))$-the set of the variables in the domain of a substitution σ.

  So $|\mathbf{Var}(\mathbf{\Gamma}) - \mathbf{Var}(\mathbf{dom}(\sigma))|$, the number of elements which are in $Var(\Gamma)$ but not in $Var(dom(\sigma))$ with standard $\leq$ on natural numbers is a well-founded ordering.
- As the definition $nET(Terms(\sigma), [\![\sigma]\!])$, $|\mathbf{nET}(\mathbf{Terms}(\sigma), [\![\sigma]\!])|$ with standard $\leq$ on natural numbers is a well-founded ordering.
- $Range(\sigma)$-the range of the substitution σ.
- $Simple(\sigma)$-the set of all simple terms in the range of the substitution σ.
- $Sup(\sigma)$-the set of all support variables in σ. Hence, the variables in $Sup(\sigma)$ are the variables generated by the algorithm. $|\mathbf{Sup}(\sigma)|$ with standard $\leq$ on natural numbers is a well-founded ordering.
- $Pair(\sigma) = Sup(\sigma) \times Simple(\sigma)$.
- $Total(\sigma, \Psi) = \Sigma_{(v,s)\in Pair(\Sigma)} Count[(v, s), \Psi]$
- **N** - the bound which is given by the user.

  Since from the condition of the inference rules Branching and Useless Branching, $|N * Pair(\sigma) - Total(\sigma, \Psi)|$ (Ψ is a multi-set.) is always greater than or equal to zero,

$\mathbf{N} * \mathbf{Pair}(\sigma) - \mathbf{Total}(\sigma, \boldsymbol{\Psi})$ with standard $\leq$ on natural numbers is a well-founded ordering.

- $|\Upsilon|$-The number of pairs in Υ.

  Since Υ is not a multi-set, $|Pair(\sigma) - \Upsilon|$ is always greater than zero. So $|\mathbf{Pair}(\sigma) - \boldsymbol{\Upsilon}|$ with standard $\leq$ on natural numbers is a well-founded ordering.

- $Par(\Gamma)$ is the set of all disequations of the form $s + t \neq^? 0$, such that s and t have the same top semi-uninterpreted function symbol, where $s, t \in Terms(\Gamma)$.

- $Par(\Gamma/\Delta) = Par(\Gamma) - \Delta$. Since $|Par(\Gamma) - \Delta|$ is always greater than zero from the condition of Decomposition Instantiation, $|\mathbf{Par}(\boldsymbol{\Gamma}/\boldsymbol{\Delta})|$ with standard $\leq$ on natural numbers is a well-founded ordering.

Our well-founded order is the lexicographic combination of

$$\begin{aligned} M(\Gamma, \sigma, \Upsilon, \Delta) \quad =&(|Var(\Gamma) - Var(dom(\sigma))|, \quad |nET(Terms(\sigma), [\![\sigma]\!])| \\ &|Par(\Gamma/\Delta)|, \quad |Sup(\sigma)|, \\ &|Pair(\sigma) - \Upsilon|, \quad N * Pair(\sigma) - Total(\sigma, \Psi)). \end{aligned}$$

We are going to prove every time we apply a rule in $\mathfrak{I}_{AAG}$, this measurement decreases by a natural number. Since every component of this measurement is greater than or equal to zero, finally, the algorithm will stop, which means an asymmetric unifier is found or no rules are applicable to $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \urcorner'$.

Splitting: Some original variable is solved in each step, so $Var(Dom(\Gamma))$ increases. Hence, $|Var(\Gamma) - Var(dom(\sigma))|$ decreases.

Branching: For the first branch, no original variable is involved, $|Var(\Gamma) - Var(dom(\sigma))|$ does not change.

$|nET(Terms(\sigma), [\![\sigma]\!])|$ does not increase because $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq |nET(Terms(\sigma), [\![\sigma]\!])|$. It can be proven in a similar way to the proof of Case 2 in Lemma 6.29.

If $t\sigma - t'\sigma \neq 0$, but $t\sigma' - t\sigma' = 0$, then because $\sigma'[v' \leftarrow v + s] = \sigma$ up to the variables in σ, then $t\sigma + t'\sigma = 0$, which is a contradiction. And $t[v]$ is replaced by $t[v' + s]$, so the number of semi-uninterpreted function terms will not change. Hence $|Par(\Gamma/\Delta)|$ does not change.

$|Sup(\sigma)|$ does not change because every time we use a new support variable to replace an old support variable.

Let us have a look at $|Simple(\sigma)|$. Let $t \in Simple(\sigma)$. If t is a constant, $t\sigma$ is also a constant, which is still one term. If t is $f(t_1, \cdots, t_n)$, then $t\sigma = f(t_1\sigma, \cdots, t_n\sigma)$, which is also one term. If t is a variable, then $t\sigma$ has two cases: $t\sigma = t$ which is still one term, or $t\sigma = v' + s$, which becomes two terms v' and s. But s is in $Simple(\sigma)$ already, so only v' replaces t as a term. Hence $|Simple(\sigma)|$ does not change. So $|Pair(\sigma)|$ does not change.

In Υ, we replace every (v, t) by (v', t) and

- add $(v', -s)$ into Υ, if $(v, -s) \notin \Upsilon$ which makes $|\Upsilon|$ increases and $|Pair(\sigma)| - |\Upsilon|$ decreases; or
- $|Pair(\sigma)| - |\Upsilon|$ will not change if $(v, -s)$ is in Υ already.

In Ψ, we replace every (v, t) by (v', t) and add (v, s) into Ψ. So $Count[(v, s), \Psi]$ increases. Then $\Sigma_{(v,s)\in Pair(\sigma)} Count[(v, s), \Psi]$ increases such that $N * Pair(\sigma) - \Sigma_{(v,s)\in Pair(\sigma)} Count[(v, s), \Psi]$ decreases.

For the second branch nothing changes except we add (v, s) into Υ. So only $|Pair(\sigma)| - |\Upsilon|$ decreases.

Useless Branching: This analysis is the same as Non-Variable Branching.

Decomposition Instantiation: If some original variable is solved, $|Var(\Gamma) - Var(dom(\sigma))|$ decreases. Otherwise $|Var(\Gamma) - Var(dom(\sigma))|$ does not change.

From Lemma 6.29, $|nET(Terms(\sigma), [\![\sigma]\!])|$ decreases.

The other branch adds some disequations into Δ, So only $|Par(\Gamma/\Delta)|$ decreases.

Elimination Instantiation: No original variable is involved, $|Var(\Gamma) - Var(dom(\sigma))|$ does not change.

$|nET(Terms(\sigma), [\![\sigma]\!])|$ does not increase because $nET(Terms(\sigma'), [\![\sigma']\!]) \preceq |nET(Terms(\sigma), [\![\sigma]\!])|$. It can be proven in a similar way to the proof of Case 2 in Lemma 6.29 because we add a new substitution $[v \leftarrow 0]$, for some support variable v, into σ.

All $t[v]$ are replaced by $t[0]$. If some disequations become equations, then $|Par(\Gamma/\Delta)|$ decreases. Otherwise, since the number of semi-uninterpreted function terms will not change, $|Par(\Gamma/\Delta)|$ does not increase.

Some support variable is set to be 0, $|Sup(\sigma)|$ decreases.

In summary, every time we apply a rule in $\mathfrak{I}_{\mathfrak{AAG}}$, $M(\Gamma, \sigma, \Upsilon, \Delta)$ decreases.

$\square$

The above theorem means our algorithm is terminating.

6.7 Soundness and Completeness

The algorithm will output a set of substitutions Σ_σ by inputting a standard unifier σ of Γ.

Intuitively, the meaning of soundness is that every member of Σ_σ is an instance of σ and is an asymmetric unifier of Γ; the meaning of completeness is if θ is an instance of σ and an asymmetric unifier, then there exists a substitution δ in Σ_σ, such that $\delta \leq_{AG}^{\Gamma} \theta$.

In this section, we will give several lemmas which can deduce soundness and completeness of the inference system $\mathfrak{I}_{AAG}$, then give the formal statement of soundness and completeness.

The main idea of proving soundness and completeness of $\mathfrak{I}_{AAG}$ is to prove $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$ if $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth \overset{*}{\Rightarrow}_{\mathfrak{I}_{AAG}} \bigvee_{i=1}^{n} (\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i)$.

For convenience, we use x, y and z to denote the original variables and v_i and w_i to denote the support variables.

Before proving soundness and completeness, we need to know when we cannot apply any rules in $\mathfrak{I}_{AAG}$, what form a quintuple $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ is in. In the next three lemmas, we give these forms at three difference points: the Splitting and Branching rules are not applicable; Decomposition Instantiation is not applicable and no rules are applicable.

Lemma 6.33 *Let $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ be a quintuple. If the Splitting and Branching rules are not applicable to $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$, then*

1. *σ is an asymmetric AG unifier of Γ; or*

2. *σ is not an asymmetric AG unifier of Γ and*

   (a) *$\daleth = true$, or*

   (b) *$\daleth = false$. In this case, there exists $x \leftarrow s + T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x + t] \in \Gamma$, where s and t are simple terms, such that $t\sigma = -s + T'$ (T' might be 0), then for each simple support variable v, such that $\pm v \in T$, we have either $(v, \mp s) \in \Upsilon$ or $v \in Vars(s)$.*

Proof. We only need to prove if

- we cannot apply any inference rules in the Splitting and Branching parts; and
- σ is not an asymmetric unifier of Γ; and
- $\daleth = false$,

then there exists $x \leftarrow s+T \in \sigma$ and $\mathbf{u} =_{\downarrow}\mathbf{v}[x+t] \in \Gamma$, where s and t are simple terms, such that $t\sigma = -s + T'$ (T' might be 0), then for each simple support variable v, such that $\pm v$ in T, we have either $(v, \mp s) \in \Upsilon$ or $v \in Vars(s)$.

Let us look at the conditions of the Branching rule:

- s is a simple term but not a variable and $v \notin Vars(s)$;
- x has a conflict at s in σ;
- $(v, \mp s) \notin \Upsilon$.
- $Count[(v, \mp s), \Psi] < N$ for the bound N.

Because $\urcorner = false$, $Count[(v, \mp s), \Psi] < N$ for the bound N. If there is a support variable v such that $(v, \mp s)$ is not in Υ, and $v \notin Vars(s)$, then the Branching Rule will be applied.

$\square$

Lemma 6.34 *Let $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner$ be a quintuple. If the Splitting, Branching and Useless Branching rules are not applicable to $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner$, then*

1. *σ is an asymmetric AG unifier of Γ; or*
2. *σ is not an asymmetric AG unifier of Γ and*
   (a) *$\urcorner = true$, or*
   (b) *$\urcorner = false$. In this case, there exists $x \leftarrow s+T \in \sigma$ and $\mathbf{u} =_{\downarrow}\mathbf{v}[x+t] \in \Gamma$, where s and t are simple terms, such that $t\sigma = -s + T'$ (T' might be 0), then for each simple support variable v, such that $\pm v$ in T, we have either $(v, \mp s) \in \Upsilon$ or $v \in Vars(s)$. If s is a variable or an inverse of a variable, then for every simple term s' in T, we have $(s, \mp s') \in \Upsilon$.*

Proof. From Lemma 6.33, we only need to prove if

- we cannot apply any of the Splitting, Branching and Useless Branching rules; and
- σ is not an asymmetric unifier of Γ; and
- $\urcorner = false$; and
- s is a variable,

then for every simple term s' in T, we have $(s, \mp s') \in \Upsilon$. Similar to the proof in 6.33, it can be proven by looking at the conditions of applying Useless Branching.

$\square$

Lemma 6.35 *Let $\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner$ be a quintuple. If Splitting, Branching, Useless Branching and Decomposition Instantiation are not applicable to $\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner$, then:*

1. *σ is an asymmetric AG unifier of Γ; or*
2. *σ is not an asymmetric AG unifier of Γ and*
   (a) *$\urcorner = false$. In this case, there exists $x \leftarrow s + T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x + t] \in \Gamma$, where s and t are simple terms, such that $t\sigma = -s + T'$ (T' might be 0), then for each simple support variable v, such that $\pm v$ in T, we have either $(v, \mp s) \in \Upsilon$ or $v \in Vars(s)$. If s is a variable or an inverse of a variable, then for every simple term s' in T, we have $(s, \mp s') \in \Upsilon$. And*
   (b) *for every simple term t' with the same top function symbol as s, such that $\mp t' \in T$, we have $s - t' \neq^{?} 0 \in \Delta$.*

Proof. From our algorithm, after exhaustively applying Useless Branching, if $\urcorner = true$, we will call VariantNarrowing to solve the asymmetric unification problems. In this case, all the σs are asymmetric AG unifiers.

If$\daleth = false$. From Lemma 6.34, all we need to prove is Case 2.b. This can be proven by looking at the conditions of Decomposition Instantiation. Recall that we will throw away any branch which violates Υ or Δ.

$\square$

Lemma 6.36 *Let $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ be a quintuple. If no rules are applicable to $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$, then:*

1. *σ is an asymmetric AG unifier of Γ; or*
2. *σ is not an asymmetric AG unifier of Γ and*

   (a) *$\daleth = false$. In this case, there exists $x \leftarrow s + T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x + t] \in \Gamma$, where s and t are non variable simple terms, such that $t\sigma = -s + T'$ (T' might be 0), then*

   - *for each simple support variable v, such that $\pm v$ in T, we have either $(v, \mp s) \in \Upsilon$ or $v \in Vars(s)$. And*
   - *for every simple term t' with the same top function symbol as s, such that $\mp t' \in T$, we have $s - t' \neq^? 0 \in \Delta$.*

Proof. Comparing this lemma with Lemma 6.35, the only difference is s is not a variable. If s is a variable, then Elimination Instantiation will let $s \leftarrow 0$. So Elimination Instantiation will apply until every variable s disappears. $\square$

The next lemma shows that no step of applying inference rules in $\mathfrak{I}_{AAG}$ loses any asymmetric unifier of Γ.

Lemma 6.37 *Let $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$, $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$, $\sigma'' \| \Upsilon'' \| \Delta'' \| \Psi'' \| \daleth''$ and $\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i$ be quintuples, where $i = 1, \cdots, n$.*

1. *if* $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner \Rrightarrow_{\mathfrak{I}_{AAG}} \sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\urcorner'$ *via Splitting, then* $\mathfrak{Inst}(\Gamma,\Delta,\Upsilon,\sigma) = \mathfrak{Inst}(\Gamma,\Delta',\Upsilon',\sigma')$.

2. *if* $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner \Rrightarrow_{\mathfrak{I}_{AAG}} (\sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\urcorner' \quad \bigvee \quad \sigma''\|\Upsilon''\|\Delta''\|\Psi''\|\urcorner'')$ *via Branching, or Useless Branching then* $\mathfrak{Inst}(\Gamma,\Delta,\Upsilon,\sigma) = \mathfrak{Inst}(\Gamma,\Delta',\Upsilon',\sigma') \cup \mathfrak{Inst}(\Gamma,\Delta'',\Upsilon'',\sigma'')$.

3. *if* $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner \Rightarrow_{\mathfrak{I}_{AAG}} (\bigvee_{i=1}^{n}\{\sigma_i\|\Upsilon_i\|\Delta_i\|\Psi_i\|\urcorner_i\}) \quad \bigvee \quad \sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\urcorner'$ *via Decomposition Instantiation, then* $\mathfrak{Inst}(\Gamma,\Delta,\Upsilon,\sigma) = (\bigcup_{i=1}^{n}\mathfrak{Inst}(\Gamma,\Delta_i,\Upsilon_i,\sigma_i)) \cup \mathfrak{Inst}(\Gamma,\Delta',\Upsilon',\sigma')$.

4. *if* $\sigma\|\Upsilon\|\Delta\|\Psi\|\urcorner \Rightarrow_{\mathfrak{I}_{AAG}} \sigma'\|\Upsilon'\|\Delta'\|\Psi'\|\urcorner'$ *via Elimination Instantiation, then* $\mathfrak{Inst}(\Gamma,\Delta,\Upsilon,\sigma) = \mathfrak{Inst}(\Gamma,\Delta',\Upsilon',\sigma')$.

Proof. **Splitting**: Splitting transforms the unifier σ into σ', such that an original top variable in $Range(\sigma)$ is moved into $Dom(\sigma')$. σ' is equivalent to σ modulo AG. if σ' violates $s - t \neq^? 0$, that means $s\sigma' \downarrow = t\sigma' \downarrow$. Since σ' is equivalent to σ, there exists θ, such that $\sigma'\theta = \sigma$. So $s\sigma'\theta \downarrow = t\sigma'\theta \downarrow$, which is equivalent to saying $s\sigma \downarrow = t\sigma \downarrow$. This is a contradiction. So it will not violate any disequation in Δ'.

In every pair (v, s) in Υ, v is a support variable. If some substitution θ violates (v, s), then $v\theta \downarrow = s\theta + T'$. But in Splitting, $\sigma' = \sigma\theta$, where $\theta = [y \leftarrow v' + T]$ where v' is a fresh support variable. This means $v\theta \downarrow = v$. So the result of Splitting satisfies Υ.

Hence $\mathfrak{Inst}(\Gamma,\Delta,\Upsilon,\sigma) = \mathfrak{Inst}(\Gamma,\Delta',\Upsilon',\sigma')$.

Branching: Because in the Branching rule, there are two cases based on the sign of s, for convenience, here we only consider $x \leftarrow v + s + T$ but not $x \leftarrow -v + s + T$. The proof for the later case is almost the same.

Let $\delta \in \mathfrak{Inst}(\Gamma,\Delta,\Upsilon,\sigma)$, then there exists a θ such that $\delta = \sigma\theta$. Since $(v, -s)$ is not in Υ, $v(\sigma \circ \theta)$ will have two possibilities:

- $v(\sigma \circ \theta) \downarrow = -s\theta + T$. This means δ violates (v, s). Since $\sigma' = \sigma[v \leftarrow v' - s]$ and

$\sigma\theta = \delta$, $\sigma'[v' \leftarrow T]\theta = \delta$. Therefore $([v \leftarrow v' - s][v' \leftarrow T]) \circ \theta = \theta$. So:

- δ is an instance of σ'.
- $\sigma' \circ [v' \leftarrow T] \circ \theta$ satisfies $\Upsilon([v \leftarrow v' - s] \circ [v' \leftarrow T] \circ \theta)$, since $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $[v \leftarrow v' - s][v' \leftarrow T] \circ \theta = \theta$.
- $\sigma' \circ [v' \leftarrow T] \circ \theta$ satisfies $(v', s)([v \leftarrow v' - s][v' \leftarrow T] \circ \theta)$ since $T - s\theta$ is irreducible.
- $\sigma' \circ [v' \leftarrow T] \circ \theta$ satisfies $\Delta([v \leftarrow v' - s] \circ [v' \leftarrow T] \circ \theta)$, since $\sigma \circ \theta$ satisfies $\Delta\theta$ and $[v \leftarrow v' + s][v' \leftarrow T] \circ \theta = \theta$.

This means $\delta \in \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.

- $v(\sigma \circ \theta) \downarrow = S$ and $-s$ is not a single term of S. So δ satisfies $(v, -s)$, which means $\delta \in \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma'')$.

Therefore $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq (\mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma') \cup \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma''))$.

On the other hand, since $\sigma'[v' \leftarrow v + s] = \sigma$ up to the variables in σ, σ is equivalent to σ'. Hence $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \supseteq (\mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma') \cup \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma''))$ can be proven from the definition of $\supseteq$.

Useless Branching: Similar to Branching.

Decomposition Instantiation: Let $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ such that $\delta = \sigma\theta$. Since $s - t \neq^? 0$ is not in Δ, $s(\sigma \circ \theta)$ and $t(\sigma \circ \theta)$ will have two possibilities:

- $s(\theta) \downarrow = t(\theta) \downarrow$. Since $\{\theta_1, \cdots, \theta_n\}$ is the complete set of standard AG unifiers of $s + t =_\downarrow 0$, there exists i and θ', such that $\theta_i\theta' = \theta$. So $\sigma \circ \theta_i \circ \theta' = \sigma \circ \theta = |_{Vars(\Gamma)}\delta$. Therefore:
  - δ is an instance of $\sigma \circ \theta_i$, since θ is an instance of σ_i.
  - $\sigma \circ \theta_i \circ \theta'$ satisfies $\Upsilon(\theta_i \circ \theta')$ since $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $\theta_i \circ \theta' = \theta$

– $\sigma \circ \theta_i \circ \theta'$ satisfies $\Delta(\theta_i \circ \theta')$ since $\sigma \circ \theta$ satisfies $\Delta\theta$ and $\theta_i \circ \theta' = \theta$

This means $\delta \in \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$.

- $s(\sigma \circ \theta) \downarrow \neq t(\sigma \circ \theta) \downarrow$, so δ satisfies $s - t \neq^? 0$. So $\delta \in \mathfrak{Inst}(\Gamma, \Delta'', \Upsilon'', \sigma'')$.

Therefore, $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq (\bigcup_{i=1}^n \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)) \cup \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$. The other direction is straightforward. So $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = (\bigcup_{i=1}^n \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)) \cup \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$

Elimination Instantiation: This rule makes some support variable 0. From Lemma 6.35, if the rules in Splitting, Branching and Decomposition Instantiation are not applicable, and σ is not an asymmetric AG unifier of Γ then $\sigma \| \Upsilon \| \Delta \| \Psi \| \urcorner$ satisfies the following conditions:

1. $\urcorner = false$. In this case, there exists $x \leftarrow s + T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x + t] \in \Gamma$, where s and t are simple terms, such that $t\sigma = -s + T'$ (T' might be 0), then for each simple support variable v, such that $\pm v$ in T, we have either $(v, \mp s) \in \Upsilon$ or $v \in Vars(s)$. If s is a variable or an inverse of a variable, then for every simple term s' in T, we have $(s, \mp s') \in \Upsilon$. And

2. for every simple term t' with the same top function symbol as s, such that $\mp t' \in T$, we have $s - t' \neq^? 0 \in \Delta$.

Let $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists θ, such that $\delta = \sigma\theta$ and $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $\Delta\theta$. Then since $x\delta = s\delta + T\delta$, we have two cases to be considered:

Case 1. s is a non-variable simple term. Since $t\sigma = -s$, $t\sigma\theta = s\theta$. Because δ is an asymmetric unifier, we need to prove $x\delta \downarrow$ does not contain $s\delta$ as a simple term. Since s is not a variable but a simple term, $s\delta$ is a simple term too. We need some simple term $-s'$ in T such that $s'\delta \downarrow = s\delta + S$.

If s' is a non-variable simple term, then s' should have the same top function symbol as s has. However, we already know $s - s' \neq^? 0 \in \Delta$. So no such s' exists.

If s' is a variable v, then $v\delta \downarrow = s\delta + S$, which will violates $(v, -s) \in \Upsilon$. So no such v exists.

Case 2. s is a variable. Let it be v (Here we will not consider the case of $-v$, since these two cases are very similar). In this case, $x\delta = v\delta \oplus T\delta$ and $y\delta = -v\delta + S\delta$. $x\delta + y\delta = v\delta + T\delta - v\delta + S\delta$ should be irreducible. So $v\delta$ should disappear. Let $v\delta \downarrow = t_1 \oplus \cdots \oplus t_n$. So we need t_i to disappear in $x\delta$ or $y\delta$. If t_i is cancelled by some non-variable simple term $t\delta$, then it violates $(v, -t)\theta$.

If t_i is cancelled by some variable w, this means $T\delta$ contains another variable. From the Branching rule, we will know in this case, $\urcorner$ will be set as true and Elimination Instantiation will not be applied.

Hence if some original variable x has a conflict at v in the premise, δ make this v 0. Therefore, δ is an instance of σ' since σ' just makes one of v's, at which some original variable has a conflict, 0s. Let $\sigma'\theta' = \delta$. Since $\sigma[v \leftarrow 0] = \sigma'$, $\sigma[v \leftarrow 0]\theta' = \delta$. So $[v \leftarrow 0] \circ \theta' = \theta$. So

- δ is an instance of σ'.
- $\sigma \circ [v \leftarrow 0] \circ \theta'$ satisfies $\Upsilon([v \leftarrow 0] \circ \theta')$, because $\sigma \circ \theta$ satisfies $\Upsilon\theta$ and $\theta = [v \leftarrow 0] \circ \theta'$.
- $\sigma \circ [v \leftarrow 0] \circ \theta'$ satisfies $\Delta([v \leftarrow 0] \circ \theta')$, because $\sigma \circ \theta$ satisfies $\Delta\theta$ and $\theta = [v \leftarrow 0] \circ \theta'$.

Hence, in this case, δ does not exist. Therefore, $\delta \in \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$, which means $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) \subseteq \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$. The other direction is straightforward. So $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \mathfrak{Inst}(\Gamma, \Delta', \Upsilon', \sigma')$.

□

From the above lemma, we proved that in every step where we apply a rule in $\mathfrak{I}_{AAG}$, any asymmetric unifier which is an instance of the premise will be an instance of one of the branches, which means we do not lose any asymmetric unifier. Next, we will prove if there is no rule we can apply to some branch, then there is no asymmetric unifier in that branch.

Lemma 6.38 *Let* $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ *be a quintuple and* Γ *be an asymmetric unification problem and* σ *be a standard unifier but not an asymmetric unifier of* Γ*. If there is not another* $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$*, such that* $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth \Rightarrow_{\mathfrak{I}_{AAG}} \sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$*, then* $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \emptyset$*.*

Proof. If $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, from Lemma 6.36 we know if σ is not an asymmetric unifier of Γ, then there are the following cases if we cannot apply any rule to $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$.

1. $\daleth = false$. In this case, there exists $x \leftarrow s + T \in \sigma$ and $\mathbf{u} =_{\downarrow} \mathbf{v}[x + t] \in \Gamma$, where s and t are non variable simple terms, such that $t\sigma = -s + T'$ (T' might be 0), then
   - for each simple support variable v, such that $\pm v$ in T, we have either $(v, \mp s) \in \Upsilon$ or $v \in Vars(s)$. And
   - for every simple term t' with the same top function symbol as s, such that $\mp t' \in T$, we have $s - t' \neq^? 0 \in \Delta$.

Note: This proof is the same as the proof in Elimination Instantiation of Lemma 6.37.

Since $x\delta = s\delta + T\delta$, and s is a non-variable simple term, we have $t\sigma = -s$ and $t\sigma\theta = s\theta$. Because δ is an asymmetric unifier, we need to prove $x\delta \downarrow$ does not contain $s\delta$ as a simple term. Since s is not a variable but a simple term, $s\delta$ is a simple term too. We need some simple term $-s'$ in T such that $s'\delta \downarrow = s\delta + S$.

If s' is a non-variable simple term, then s' should have the same top function symbol as s has. However, we already know $s - s' \neq^? 0 \in \Delta$. So no such s' exists.

If s' is a variable v, then $v\delta \downarrow = s\delta + S$, which will violates $(v, -s) \in \Upsilon$. So no such v exists.

□

So from the above lemma, we know we can throw out the branch $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$ if we cannot apply any rules to it and σ is not an asymmetric unifier of Γ.

Let use recall the algorithm. If we get an asymmetric unifier when we apply Branching, we will throw out other branches and return the only one there. This is because of the following lemma:

Lemma 6.39 *Let* $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth$, $\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth'$ *and* $\sigma'' \| \Upsilon'' \| \Delta'' \| \Psi'' \| \daleth''$ *be quintuples. If* $\sigma \| \Upsilon \| \Delta \| \Psi \| \daleth \Rrightarrow_{\mathfrak{I}_{AAG}} (\sigma' \| \Upsilon' \| \Delta' \| \Psi' \| \daleth' \ \bigvee \ \sigma'' \| \Upsilon'' \| \Delta'' \| \Psi'' \| \daleth'')$ *via Branching, then* $\sigma' =_{AG}^{\Gamma} \sigma'' =_{AG}^{\Gamma} \sigma$.

Proof. Since for any substitution θ, $(\theta = \theta([v \leftarrow v' - s]))([v' \leftarrow v + s])$, this lemma is true. $\square$

Recall the assumption about VariantNarrowing we made in the beginning of the last section. If we regard calling VariantNarrowing as one inference step, we have the following Theorem:

Theorem 6.40 *Let* Γ *be an asymmetric unification problem and* σ *a standard unifier of* Γ. *If there exists a set of quintuples* $\bigvee_{i=1}^{n}(\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i)$, *such that* $\sigma \| \emptyset \| \emptyset \| \emptyset \| False \overset{*}{\Rrightarrow}_{\mathfrak{I}_{AAG}} \bigvee_{i=1}^{n}(\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i)$ *and no rules are applicable any more, then*

- $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$; *and*
- *the complete set of asymmetric unifiers of* Γ *from* σ *is*

$$\{\sigma_j \mid \sigma_j \| \Upsilon_j \| \Delta_j \| \Psi_j \| \daleth_j \in \bigvee_{i=1}^{n}(\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i) \text{ and}$$
$$\sigma_j \text{ is an asymmetric unifier } \}.$$

Proof. We can prove this theorem by induction via Lemma 6.37 and 6.38. $\square$

From the above theorem, we can get the following two corollaries:

Corollary 6.41 (Soundness) Let Γ be an asymmetric unification problem and σ a standard unifier of Γ. If the algorithm SEARCHING(Γ, σ) outputs Σ_σ, then any member σ_i of Σ_σ is an asymmetric unifier of Γ and an instance of σ.

Corollary 6.42 (Completeness) Let Γ be an asymmetric unification problem and σ a standard unifier of Γ. If the algorithm SEARCHING(Γ, σ) outputs Σ_σ, then for any asymmetric unifier δ of Γ, which is an instance of σ, there exist a $\sigma_i \in \Sigma_\sigma$, such that $\sigma_i \leq_{AG}^{\Gamma} \delta$.

6.8 Special Rules - For Efficiency

All of the rules in this section have the highest priority, which means they can be applied any time they are applicable. Using these rules will not affect the soundness and completeness of our algorithm. They are used for improving the efficiency.

Cancellation Failure

$$\frac{\sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Condition:

- There is an equation $\mathbf{u} =_{\downarrow} \mathbf{v}[s - t] \in \Gamma$, such that $s\sigma = t\sigma$, where s and t are two non-sum terms.

Example 6.43

$$\Gamma = \{x + a =_{\downarrow} b + y, \quad z =_{\downarrow} f(x - y) - f(b - a)\}$$

$$\sigma = [x \leftarrow -a + b + y, z \leftarrow 0]$$

Not Enough Terms Failure

$$\frac{\sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- there is an equation $\mathbf{u} =_{\downarrow} \mathbf{v} \in \Gamma$ such that the number of simple terms in $\mathbf{u}$ is less then the number of terms in $\mathbf{v}$ and there are no top variables in $\mathbf{u}$.

Example 6.44

$$\Gamma = \{f(x) + f(a) =_{\downarrow} x + y + c\}$$

$$\sigma = [y \leftarrow -x - c + f(x) + f(a)]$$

Identity Failure

$$\frac{\sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- there is an equation $\mathbf{u} =_{\downarrow} \mathbf{v}[x + S]$, such that $x\sigma = 0$, where S is not empty.

Decomposition Failure

$$\frac{[x \leftarrow s + S] \cup \sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- s is a simple non-variable term;
- for every top variable v in S, $(v, s) \in \Upsilon$;
- $\mathbf{u} =_{\downarrow} \mathbf{v}[x - t] \in \Gamma$ and $t\sigma = s$;
- For every term t' in S which has the same uninterpreted function symbol as s, $s - t' \neq 0 \in \Delta$.

Example 6.45

$$\Gamma = \{f(a) + f(b) =_{\downarrow} x + f(y), f(a) + f(b) =_{\downarrow} z + f(y)\}$$

$$\sigma = [x \leftarrow f(a) + f(b) - f(y), z \leftarrow f(a) + f(b) - f(y)]$$

$$\Upsilon = \emptyset$$

$$\Delta = \{f(a) - f(y) \neq^{?} 0, f(b) - f(y) \neq^{?} 0\}$$

Example 6.46

$$\Gamma = \{y + b =_{\downarrow} x + a, y + b =_{\downarrow} y + b, f(a) + f(b) =_{\downarrow} z + f(y)\}$$

$$\sigma = [x \leftarrow -v + a - b, y \leftarrow v, z \leftarrow f(a) + f(b) - f(v)]$$

$$\Upsilon = \{(v, a)\}$$

$$\Delta = \emptyset$$

Non-Variable Failure

$$\frac{[x_0 \leftarrow s + x_1 + x_2 + \cdots + x_n + S] \cup \sigma \| \Upsilon \| \Delta}{\textbf{Fail}}$$

Conditions:

- For all x_i, x_i has a conflict at s in the premise;
- no top variables are in S;
- s is not a variable;
- no terms in S have the same top function symbol as s.

Example 6.47

$$\Gamma = \{y + b =_{\downarrow} x + a, y + a =_{\downarrow} y + a\}$$

$$\sigma = [x \leftarrow y - a + b]$$

Eager Decomposition Instantiation

$$\frac{[x_0 \leftarrow s + t + x_1 + x_2 + \cdots + x_n + S] \cup \sigma \| \Upsilon \| \Delta}{([x_0 \leftarrow x_1 + x_2 + \cdots + x_n + S] \cup \sigma)\theta \| \Upsilon\theta \| \Delta\theta}$$

where:

- $\theta = mgu(t, s)$

Conditions:

- For all x_i, x_i has a conflict at s in the premise;
- there are no top variables in $Vars(\Gamma)$;

- s is not a variable and s and t have the same top function symbol;
- $s - t \neq^? 0 \notin \Delta$.

Example 6.48

$$\Gamma = \{z + f(b) =_{\downarrow} x + f(y), z + f(y) =_{\downarrow} z + f(y)\}$$

$$\sigma = [x \leftarrow -f(y) + z + f(b)]$$

$$\Upsilon = \emptyset$$

$$\Delta = \emptyset$$

Then we will get

$$\sigma_1 = [x \leftarrow f(z), y \leftarrow b]$$

$$\Upsilon_1 = \emptyset$$

$$\Delta_1 = \qquad \emptyset$$

here σ_1 is an asymmetric unifier.

Theorem 6.49 *Let Γ be an asymmetric unification problem and σ a standard unifier of Γ. If there exists a set of quintuples $\bigvee_{i=1}^{n}(\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i)$, such that $\sigma \| \emptyset \| \emptyset \| \emptyset \| False \Longrightarrow_{\mathfrak{I}_{AAG}} \bigvee_{i=1}^{n}(\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i)$ via a special rule, then $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i)$.*

Proof. For all the failure rules, $\bigvee_{i=1}^{n}(\sigma_i \| \Upsilon_i \| \Delta_i \| \Psi_i \| \daleth_i) = Failure$, so $\bigcup_i \mathfrak{Inst}(\Gamma, \Delta_i, \Upsilon_i, \sigma_i) = \emptyset$.

Cancellation Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ, such that $\delta = \sigma\theta$. Since $s\sigma = t\sigma$, $s\sigma\theta = t\sigma\theta$. δ is not an asymmetric unifier.

Hence $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \emptyset$.

Not Enough Terms Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$ and there exists an equation $t_1 + \cdots + t_n =_{\downarrow} s_1 + \cdots s_m \in \Gamma$, where t_i is a simple non-variable term and $m > n$. Since t_i is not a variable, $t_i\delta$ is a non-variable simple term too. So $(t_1\delta + \cdots + t_n\delta)\downarrow$ has at most n simple terms. Hence $(s_1\delta + \cdots + s_m\delta)\downarrow$ should have at most n simple terms, which means $s_1\delta + \cdots + s_m\delta$ is reducible since $m > n$. This shows that δ is not an asymmetric unifier. So $\mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma) = \emptyset$.

Identity Failure

Suppose $\delta \in \mathfrak{Inst}(\Gamma, \Delta, \Upsilon, \sigma)$, then there exists a θ, such that $\delta = \sigma\theta$. Since $x\sigma = 0$, $x\sigma\theta = 0$. So δ is not an asymmetric unifier.

Decomposition Failure

We can get the conclusion by using the same technique as in Lemma 6.38. So we do not give the details here.

Non-Variable Failure

We can use the same technique as in the proof for Lemma 6.38 to prove this part. So here we do not give the details.

Eager Decomposition Instantiation

We can use the same technique as in the Decomposition Instantiation part in Lemma 6.37 to prove this part. So we do not give the details here.

□

From the above theorem, we know none of the special rules will affect the completeness and soundness of $\mathfrak{I}_{AAG}$. Since except the Special Decomposition Instantiation, all of the rules go to failure directly, and we can use a similar analysis in Decomposition Instantiation to prove Special Decomposition Instantiation decreases $M(\Gamma, \sigma, \Upsilon, \Delta)$, the termination will not be affected either.

6.9 Conclusion and Future Work

We introduced inference rules for searching for a complete set of asymmetric unifier of an asymmetric AG-unification problem from its standard unifiers. For making this algorithm complete, we borrow variant narrowing as a supplement. We proved these inference rules to be sound, complete and terminating. We also introduced auxiliary rules to avoid applying non-deterministic rules too early, and to make the inference system more efficient.

For future work, we will also think about how to optimize(reduce the use of variant narrowing) this algorithm and will implement it in Maude. We will also consider other important theories, like exclusive OR with homomorphism and Abelian groups with homomorphism etc.

Chapter 7

Conclusion

In this thesis, we started from introducing formal methods of cryptographic protocol analysis and discussed the search method in Maude-NPA. We explained why unification algorithms play an important role in cryptographic protocol analysis. Then we used one chapter to give the preliminary knowledge about rewriting and unification. After that, Chapters 3, 4, 5 and 6 introduced two standard unification algorithms and two asymmetric unification algorithms.

The two standard unification algorithms are based on two important and common theories in cryptographic protocol primitives: exclusive OR with homomorphism and Abelian groups with homomorphism. Though some related papers solved the these problems [6, 3, 4, 10, 36, 62, 9, 5], some of them only solved the unification problem without considering uninterpreted function symbols and some of them solved them in theory but not efficiently in practice. The algorithms we introduced in Chapters 3 and 4 are simple and easy to implement. We also proved these inference systems are terminating, sound and complete. All of these algorithms have been implemented in Maude and the test results are good.

In a cryptographic protocol, often more than one or two theories are involved. Many times, we need to consider many different theories, like cancellation between decryption and encryption, Abelian rings, multiple homomorphism functions etc. In the future, we hope our work can inspire people to develop practically efficient unification algorithms for other theories.

The two asymmetric unification algorithms are for exclusive OR and Abelian groups. These algorithms are devised for helping optimization of state reduction [25], which is a technique of reducing search space used in the Maude-NPA, in order to replace the Variant Narrowing [26, 27] which is currently used in the Maude-NPA. Variant Narrowing is inefficient in practice due to the complexity of computing variants for exclusive OR and Abelian groups [14, 26] and the large complete set of unifiers for $\mathfrak{AC}$-unification. The algorithms, which we introduced in Chapters 5 and 6, exploit the corresponding standard unification algorithm to search for a complete set of asymmetric unifiers. Also, we proved our algorithms are sound, complete and terminating. The implementation of asymmetric exclusive OR unification algorithm is done and was incorporated into Maude-NPA. The preliminary test results are very encouraging.

The asymmetric unification problem was first defined in [22] and it is a special case of contextual unification. Contextual unification can be used not only in Maude-NPA, but also in other cryptographic protocol analysis tools. As a special case, how to use asymmetric unification problem to help reduce the search space in other cryptographic protocol analysis tools is one of the future work.

On the other side, in addition to optimizing the asymmetric Abelian group unification algorithm and implementing it, we also hope we can build other asymmetric unification algorithms for different theories, like exclusive OR with homomorphism and Abelian groups with homomorphism. Efficient E-unification algorithms for other common theories, like cancellation, are also worth for us to think about.

References

[1] Maude: http://maude.cs.uiuc.edu/.

[2] IEEE 802.11 Local and Metropolitan Area Networks: Wireless LAN Medium Access Control (MAC) and Physical (PHY) Specifications, 1999.

[3] Franz Baader. Unification in Commutative Theories. *J. Symb. Comput.*, 8(5):479–497, 1989.

[4] Franz Baader. Unification in Commutative Theories, Hilbert's Basis Theorem, and Gröbner Bases. *J. ACM*, 40(3):477–503, 1993.

[5] Franz Baader and Klaus U. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. In Deepak Kapur, editor, *CADE*, volume 607 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 1992.

[6] Franz Baader and Wayne Snyder. Unification Theory. In Robinson and Voronkov [51], pages 445–532.

[7] David A. Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[8] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.

[9] Alexandre Boudet, Evelyne Contejean, and Claude Marché. AC-Complete Unification and its Application to Theorem Proving. In Harald Ganzinger, editor, *RTA*, volume 1103 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 1996.

[10] Alexandre Boudet, Jean-Pierre Jouannaud, and Manfred Schmidt-Schauß. Unification in Boolean Rings and Abelian Groups. *J. Symb. Comput.*, 8(5):449–477, 1989.

[11] Ernest F. Brickell and Yacov Yacobi. On Privacy Homomorphisms (Extended Abstract). In David Chaum and Wyn L. Price, editors, *EUROCRYPT*, volume 304 of *Lecture Notes in Computer Science*, pages 117–125. Springer, 1987.

[12] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. *Theor. Comput. Sci.*, 338(1-3):247–274, 2005.

[13] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[14] Hubert Comon-Lundh and Stéphanie Delaune. The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In Jürgen Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[15] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A Survey of Algebraic Properties Used in Cryptographic Protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[16] Cas J. F. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 119–128. ACM, 2008.

[17] Stéphanie Delaune. An undecidability result for AGh. *Theor. Comput. Sci.*, 368(1-2):161–167, 2006.

[18] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in Key Distribution Protocols. *Commun. ACM*, 24(8):533–536, 1981.

[19] Nachum Dershowitz and David A. Plaisted. Rewriting. In Robinson and Voronkov [51], pages 535–610.

[20] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[21] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[22] Serdar Erbatur, Santiago Escobar, Deepak Kapur, Zhiqiang Liu, Christopher Lynch, Catherine Meadows, José Meseguer, Paliath Narendran, Sonia Santiago, and Ralf Sasse. Effective Symbolic Protocol Analysis via Equational Irreducibility Conditions. In *ESORICS*, pages 73–90, 2012.

[23] Santiago Escobar, Catherine Meadows, and José Meseguer. A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. *Theor. Comput. Sci.*, 367(1-2):162–202, 2006.

[24] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *FOSAD*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2007.

[25] Santiago Escobar, Catherine Meadows, and José Meseguer. State Space Reduction in the Maude-NRL Protocol Analyzer. *CoRR*, abs/1105.5282, 2011.

[26] Santiago Escobar, José Meseguer, and Ralf Sasse. Variant Narrowing and Equational Unification. *Electr. Notes Theor. Comput. Sci.*, 238(3):103–119, 2009.

[27] Santiago Escobar, Ralf Sasse, and José Meseguer. Folding Variant Narrowing and Optimal Variant Termination. In Peter Csaba Ölveczky, editor, *WRLA*, volume 6381 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2010.

[28] Qing Guo, Paliath Narendran, and David A. Wolfram. Unification and Matching Modulo Nilpotence. In McRobbie and Slaney [40], pages 261–274.

[29] Miki Hermann and Phokion G. Kolaitis. Unification Algorithms Cannot be Combined in Polynomial Time. In McRobbie and Slaney [40], pages 246–260.

[30] Yehuda Lindell Jonathan Katz. *Introduction to Modern Cryptography.* Chapman & Hall/CRC, Talylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487, 2008.

[31] Deepak Kapur and Paliath Narendran. Double-exponential Complexity of Computing a Complete Set of AC-Unifiers. In *LICS*, pages 11–21, 1992.

[32] Deepak Kapur, Paliath Narendran, and Lida Wang. An E-unification Algorithm for Analyzing Protocols That Use Modular Exponentiation. In Robert Nieuwenhuis, editor, *RTA*, volume 2706 of *Lecture Notes in Computer Science*, pages 165–179. Springer, 2003.

[33] Deepak Kapur, Paliath Narendran, and Lida Wang. A Unification Algorithm for Analysis of Protocols with Blinded Signatures. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *Lecture Notes in Computer Science*, pages 433–451. Springer, 2005.

[34] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 3nd Edition.* Addison-Wesley, 2004.

[35] Steve Kremer, Mark Ryan, and Ben Smyth. Election Verifiability in Electronic Voting Protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[36] D. Lankford, G.Butler, and B. Brady. Abelian Group Unification algorithms For Elementary Terms. *Contemporary Mathematics*, 29:193–199, 1984.

[37] Gavin Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[38] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996.

[39] Gavin Lowe and A. W. Roscoe. Using CSP to Detect Errors in the TMN Protocol. *IEEE Trans. Software Eng.*, 23(10):659–669, 1997.

[40] Michael A. McRobbie and John K. Slaney, editors. *Automated Deduction - CADE-13, 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, July 30 - August 3, 1996, Proceedings*, volume 1104 of *Lecture Notes in Computer Science*. Springer, 1996.

[41] Catherine Meadows. Formal Verification of Cryptographic Protocols: A Survey. In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *ASIACRYPT*, volume 917 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1994.

[42] Catherine Meadows. Language generation and verification in the NRL protocol analyzer. In *CSFW*, pages 48–61. IEEE Computer Society, 1996.

[43] Catherine Meadows. The NRL Protocol Analyzer: An Overview. *J. Log. Program.*, 26(2):113–131, 1996.

[44] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, page 500. CRC Press, 1996.

[45] John C. Mitchell. Finite-State Analysis of Security Protocols. In Alan J. Hu and Moshe Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 71–76. Springer, 1998.

[46] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, 1978.

[47] Roger M. Needham and Michael D. Schroeder. Authentication Revisited. *Operating Systems Review*, 21(1):7, 1987.

[48] Werner Nutt. Unification in Monoidal Theories. In Mark E. Stickel, editor, *CADE*, volume 449 of *Lecture Notes in Computer Science*, pages 618–632. Springer, 1990.

[49] J. C. Lopez P. and R. Monroy. Formal Support to Security Protocol Development: a Survey. *Journal of Computation and Systems*, 12(1):89–108, 2008.

[50] R. Rivest, L. Adleman, and M. Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation*, pages 169–177, 1978.

[51] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes).* Elsevier and MIT Press, 2001.

[52] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, 2009.

[53] Peter Y. A. Ryan and Steve A. Schneider. An Attack on a Recursive Authentication Protocol. A Cautionary Tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.

[54] Manfred Schmidt-Schauß. Unification in a Combination of Arbitrary Disjoint Equational Theories. *J. Symb. Comput.*, 8(1/2):51–99, 1989.

[55] Klaus U. Schulz. A Criterion for Intractability of E-unification with Free Function Symbols and Its Relevance for Combination Algorithms. In Hubert Comon, editor, *RTA*, volume 1232 of *Lecture Notes in Computer Science*, pages 284–298. Springer, 1997.

[56] Vitaly Shmatikov and Ulrich Stern. Efficient Finite-State Analysis for Large Security Protocols. In *CSFW*, pages 106–115, 1998.

[57] Simon Singh. *The Code Book.* Doubleday, 1999.

[58] Douglas R. Stinson. *Cryptography - Theory and Practice, Third Edition.* Chapman & Hall/CRC, 2006.

[59] Makoto Tatebayashi, Natsume Matsuzaki, and David B. Newman Jr. Key Distribution Protocol for Digital Mobile Communication Systems. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 324–334. Springer, 1989.

[60] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand Spaces: Why is a Security Protocol Correct? In *IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society, 1998.

[61] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 7(1):191–230, 1999.

[62] Max Tuengerthal, Ralf Küsters, and Mathieu Turuani. Implementing a Unification Algorithm for Protocol Analysis with XOR. *CoRR*, abs/cs/0610014, 2006.