Proving non-convertibility

Meurers, A.H.J.

Award date:
2011

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Formal Systems Analysis Group

Proving non-convertibility

Master's Thesis

A.H.J. Meurers

Supervisor:
prof. dr. Hans Zantema

Eindhoven, 1st September, 2011

Abstract

In this paper we investigate the possibilities of using SMT solving to prove that an equation $s = t$ does not equationally follow from a set E of equations. This means that there is no conversion from s to t in which every conversion step is an application of an equation in E. The main idea is to find a model in which E holds and $s = t$ does not. We focus on quasi linear interpretations which turned out to be more powerful than finite models. An implementation was made that uses the SMT solver Yices to find the required models.

Contents

Chapter 1

Introduction

Equations are formulas of the shape $s = t$. They occur frequently in mathematics, logic and computer science. Given such an equation we are often interested in whether it follows or is a consequence of, some given set of equations. Suppose we have the following set of axioms for succession and addition:

$$+(0, x) = x$$
$$+(S(x), y) = S(+(x, y))$$
$$+(x, y) = +(y, x)$$

We might wish to know whether the equation $+(S(0), S(0)) = S(S(0))$ is a consequence of the given axioms. Showing that this is true, is done by means of equational reasoning [7]. It should be obvious that the given equation can indeed be shown to follow from the set of equations only by substituting the variables for arbitrary terms and applying the equations.

To prove that that an equation follows from a set of equations is commonly referred to as the word problem. It is a problem known to be undecidable [1]. However the problem has been extensively studied, and as a result there have been developed many tools to solve instances of the word problem [6, 3].

In the case that an equation does not follow from the set of equations, i.e this equation is *non-convertible*, it would still be very valuable to be able to prove this. One of the motivating examples for this paper is the tool Streambox [8]. Streambox is a tool that attempts to prove the equality of infinite streams. During the proof process, Streambox attempts to prove equations with respect to some equation system. It turns out that a great many of these equations are non-convertible. Especially if the equation system is non-terminating, Streambox will keep searching indefinitely. After a certain time limit this process is stopped, having acquired no proof of non-convertibility.

Despite the obvious importance of proving non-convertibility, there seems to have been done little research on the subject. In this paper we therefore investigate an approach using Satisfiability Modulo Theories (SMT) [2] solving to give proofs that equations are non-convertible.

1.1 SMT solving

SMT solving is about checking the satisfiability of logical formulas containing operations from various theories such as equality reasoning, arithmetic etc. SMT is an extension of SAT

where we attempt to check the satisfiability of boolean expressions, written using only the operations $\wedge, \vee, \neg$ together with variables and parentheses. However for many problems it is often desired to have the added expressiveness of equality, arithmetic operations, functions, arrays, quantifiers and more. The problem of non-convertibility requires us to express that terms containing function symbols and variables, are either equal or unequal. This can be conveniently expressed by means of Satisfiability Modulo Theories.

The general approach of using SMT in this paper is that for each problem instance, we wish to generate a set of SMT formulas that represent the problem. These formulas should be designed in such a way that if they are satisfiable, the problem is non-convertible. We then use the SMT solver Yices to check that these formulas can be satisfied.

1.2 Reference tool

In order to investigate the performance and possibilities of our approach we use the automated theorem prover suite Prover9/Mace4 [6] as a reference. This suite consists of two tools, called Prover9 and Mace4. The main tool is Prover9 which is an automated theorem prover for first-order and equational logic problems. The tool Mace4 is used to provide counterexamples by means of finding finite models.
Mace4 takes a set of equalities and a goal for which it tries to find a finite model that satisfies the given problem. It does this by generating models of increasing domain sizes until it either finds a model or a certain limit is reached. If a model is found, the solution is given and the proof is completed that proves that the goal does not hold with respect to the equations. We use the approach of Mace4 as a starting point for our approach using SMT solving. Later we will see the limitations of using finite models and investigate possible improvements.

Chapter 2

Problem definition

In this chapter we define non-convetibility more concisely to provide a general framework for applying our solution based on SMT solving. We wish to prove that an equality cannot be derived from a given set of equations. To make this more formal we first introduce some basic definitions. Let Σ be the set of function symbols each having a fixed arity ≥ 0, and X to be the set of variables. We now define $T(\Sigma, X)$ to be the set of all possible terms over Σ and X. Using these terms we can define an equation to be a pair $(s, t) \in T(\Sigma, X) \times T(\Sigma, X)$. A more common way to describe an equation is by using the notation $s = t$. A set of such equations we denote as an equation system E.

The goal of this paper is to prove that a given equation $s = t$ cannot be derived from an equation system E. For this purpose we introduce the *equality relation* of an equation system E to be the relation $=_E$ on $T(\Sigma, X)$ that we define inductively as follows:

- $s^\sigma =_E t^\sigma$ for every $(s, t) \in E$ and every substitution σ
- $f(t_1, \ldots, t_k, \ldots, t_n) =_E f(t_1, \ldots, t_k^{'}, \ldots, t_n)$ for every $f \in F$ with arity n and all terms $t_1 \ldots t_n$ where $t_k =_E t_k^{'}$

We also define the following three properties of $=_E$ for all $s, t, u \in T(\Sigma, X)$:

- $t =_E t$
- $s =_E t \Rightarrow t =_E s$
- $s =_E t \wedge t =_E u \Rightarrow s =_E u$

which states that the relation $=_E$ is reflexive, symmetric and transitive. The relation $=_E$ now denotes all possible equations that can be derived from E. Using this relation we can now state the following definition:

Definition 2.1 A equation $s = t$ is defined to be non-convertible with respect to an equation system E, notation $\nvdash_E s = t$, if it holds that $(s, t) \notin =_E$.

By definition 2.1, to prove that an equation indeed is non-convertible with respect to E, we must show that it cannot exist within $=_E$. To do this we use the *interpretation method* [1, 4]

2.1 Interpretation method

The idea of the interpretation method is that one assigns values to each of the terms in E. These values are assigned by means of a valuation function. To be able to prove that an equation is non-convertible, the valuation function has to be chosen in such a way that it satisfies the following two criteria:

- For all equations $s =_E t$ the values of s and t are equal
- For the goal G of the shape $s_g = t_g$ the values of s_g and t_g are different

If there indeed exists such a valuation function, this means that for every equation in $s =_E t$ the values of s and t are equal. And thus given a goal $s_g = t_g$ for which the values of the terms are unequal, it should be obvious that this equation cannot exists within $s =_E t$. Hence by definition 2.1, $s_g = t_g$ is non-convertible.

However since there are potentially infinitely many equations in $=_E$, it would be convenient not to have to define the valuation function for all these equations. Therefore it would be preferable to only have to define the valuation function over the finitely many equations in E and the goal G. In the remainder of this chapter we shall prove that it is indeed sufficient to only define the valuation function over the terms in E and G, to prove non-convertibility of G.

Before we can give this proof, we must formally define the valuation function. This function must be able to take a term, substitute its variables for values in a domain A and then produce some value in this same domain A. To be able to do this, we must first define how each function $f \in \Sigma$ is *interpreted*. This means that for each function f of arity n we define an *interpretation function* $[f] : A^n \to A$ to be a function that can have any required shape. Using the set $A^X = \{\alpha : X \to A\}$ of all possible substitutions of a variable in X for a value in A, we can now define the valuation function

$$[t, \alpha] : T(\Sigma, X) \times A^X \to A$$

inductively by

$$[x, \alpha] = x^\alpha$$

$$[f(t_1 \dots t_n), \alpha] = [f]([t_1, \alpha] \dots [t_n, \alpha])$$

where $x \in X$, $\alpha : X \to A$, $f \in \Sigma$ and $t_1 \dots t_n \in T(\Sigma, X)$. Using this definition we can now formally state our goal:

Theorem 2.2 Let E be a set of equations and let G be the goal $s_g = t_g$. G is non-convertible with respect to E if the following holds, where $\alpha : X \to A$:

- $\forall_\alpha [s, \alpha] = [t, \alpha]$ for every equation $s = t \in E$
- $\exists_\alpha [s_g, \alpha] \neq [t_g, \alpha]$

To prove this theorem we must show that if the values for the terms in each equation in E are equal then then the same holds for the equations in $=_E$. Before we can prove this, we first need the following lemma:

Lemma 2.3 Let $\sigma : X \to T(F, X)$ be any substitution and let $\alpha : X \to A$. Define $\beta : X \to A$ by $\beta(x) = [x^\sigma, \alpha]$ for $x \in X$. Then

$$[t^\sigma, \alpha] = [t, \beta]$$

Proof We prove this by induction on the structure of t. We start with the case where $t = x$.

$$\begin{aligned}
& [x^\sigma, \alpha] \\
= & \quad \{ \text{ Definition of } \beta \ \} \\
& \beta(x) \\
= & \quad \{ \text{ Definition of substitution } \} \\
& x^\beta \\
= & \quad \{ \text{ Definition of } [t, \alpha] \ \} \\
& [x, \beta]
\end{aligned}$$

And next we prove the case that $t = f(t_1 \dots t_n)$ where $t_1 \dots t_n \in T(F, X)$:

$$\begin{aligned}
& [f(t_1, \dots, t_n)^\sigma, \alpha] \\
= & \quad \{ \text{ Property of substitution } \} \\
& [f(t_1^\sigma, \dots, t_n^\sigma), \alpha] \\
= & \quad \{ \text{ Definition of } [t, \alpha] \ \} \\
& [f]([t_1^\sigma, \alpha], \dots, [t_n^\sigma, \alpha]) \\
= & \quad \{ \text{ Induction hypothesis } \} \\
& [f]([t_1, \beta], \dots, [t_n, \beta]) \\
= & \quad \{ \text{ Definition of } [t, \alpha] \ \} \\
& [f(t_1, \dots, t_n), \beta])
\end{aligned}$$

With these two cases we have covered all possible terms t, hence the lemma has been proven. □

To now prove theorem 2.2 we state the following lemma:

Lemma 2.4 If $[s, \alpha] = [t, \alpha]$ for all $(s, t) \in E$ then it holds that $[s', \alpha] = [t', \alpha]$ for all $(s', t') \in =_E$.

Proof To prove this we must show that $=_E$ is closed under substitution and context. If that is the case then all possible equations derived from the equations in E will have the same weights. For $=_E$ to be closed under substitution, we have to prove that for a given equation $s =_E t$ and a substitution $\sigma : X \to T(\Sigma, X)$ it holds that $s^\sigma =_E t^\sigma$. We prove this by showing that for all $\alpha : X \to A$ it holds that $[s^\sigma, \alpha] = [t^\sigma, \alpha]$. This would imply that $s^\sigma =_E t^\sigma$ which is our goal.

$$\begin{aligned}
& [s^\sigma, \alpha] \\
= & \quad \{ \text{ Lemma 2.3 } \} \\
& [l, \beta] \\
= & \quad \{ \ l =_E r; \ \beta \text{ is not dependant on } l \ \} \\
& [r, \beta] \\
= & \quad \{ \text{ Lemma 2.3 } \} \\
& [t^\sigma, \alpha]
\end{aligned}$$

From this we can indeed conclude that $s^\sigma =_E t^\sigma$. Hence $=_E$ is closed under substitution.

Next we have to prove that $=_E$ is closed under context. For $=_E$ to be closed under context, we have to prove that for a given equation $s =_E t$ and an $f \in \Sigma$ it holds that $f(\ldots, s, \ldots) =_E f(\ldots, t, \ldots)$. Let $\alpha : X \to A$ then:

$$
\begin{aligned}
& [f(\ldots, s, \ldots), \alpha]) \\
= \quad & \{ \text{ Definition } [t, \alpha] \ \} \\
& [f](\ldots, [s, \alpha], \ldots) \\
= \quad & \{ \ [l, \alpha] = [r, \alpha] \ \} \\
& [f](\ldots, [t, \alpha], \ldots) \\
= \quad & \{ \text{ Definition } [t, \alpha] \ \} \\
& [f(\ldots, t, \ldots), \alpha]
\end{aligned}
$$

And so we have proven that $=_E$ is closed under substitution and context. □

By Lemma 2.4 we now know that if for every equation $s = t$ in E it holds that $[s, \alpha] = [t, \alpha]$ then for every possible equation $s' = t'$ that can be derived from E it also holds that $[s', \alpha] = [t', \alpha]$. Therefore a goal for which holds $[s, \alpha] \neq [t, \alpha]$ can never satisfy $(s, t) \in =_E$, hence this goal is non-convertible.

From this result we can conclude that if we wish to prove an equation non-convertible, we must be able to find interpretation functions for the functions in Σ such that the interpreted equations in E hold and the goal does not.

In this remainder of this paper we investigate the use of SMT solving to find these interpretation functions based on a given problem.

Chapter 3

Finite Models

We begin our investigation of using the interpretation method, by trying to prove non-convertibility problems for which the interpretation functions have a finite domain A. Our goal is to be able to find the right interpretation functions for a given non-convertibility problem. To do this we design SMT formulas that represent a non-convertibility problem by means of interpretation functions on a finite domain. An SMT solver will then attempt to find these interpretation functions such that the equation in E hold and the goal G does not. If the SMT formulas are satisfiable, and hence the interpretation functions can be found, the problem is proven non-convertible.

Since for a given problem it is not know what domain size is required for the interpretation functions we use the same approach as Mace4. For each problem We will generate SMT specifications using interpretation functions of increasing domain size. The domain size is increased until a satisfiable solution is found or a maximum domain size is reached. This maximum is a user defined limit, as theoretically it is possible to keep increasing the domain size indefinitely.

3.1 Theory

Since we use finite models, all the interpretation functions will have a finite domain A. Hence for each function symbol f in Σ of arity n, we use an interpretation function $[f] : A^n \to A$ where A is a finite set. To maximize the power of our approach we allow these functions to have any possible definition. And thus each interpretation function $[f]$ is defined as follows:

Definition 3.1 Let $[f] : A^n \to A$ be an interpretation function of arity n. We define $[f]$ as follows based on the value of n:

- If $n > 0$ then for every $v_n, \ldots, v_1 \in A$

$$[f](v_n, \ldots, v_1) \geq 0 \wedge [f](v_n, \ldots, v_1) < |A|$$

- If $n = 0$ then

$$[f] \geq 0 \wedge [f] < |A|$$

Example 3.1.1 For example let $[f](x, y) : A \times A \to A$ be an interpretation function over the two variables x, y and let $A = \{0, \ldots, k\}$. We then define $[f]$ as follows:

$$[f](x, y) \geq 0 \quad \wedge \quad [f](x, y) \leq k \qquad \text{for all } x, y \in A$$

And thus the output of $[f](x, y)$ can be an arbitrary value as long as it exists in A.

As was stated in section 1, to prove that an equation G is non-convertible, it must be shown that for all values in A, the interpreted values $[t, \alpha]$ of the terms in each equation are equal, and for G are not. However since a term is usually a composition of functions, the value of a term is the result of the composition of interpretation functions. Because we allow the interpretation functions to have any possible definition, there is often no formula that can express this definition. And thus it is not possible to define a formula that represents the composition of interpretation function occurring in a term.

To still be able to express that two terms are equal, we instead enumerate the values of the terms for each input value separately. In the next section we shall use this approach to describe the set of SMT formulas that are satisfiable when G is non-convertible.

3.2 Implementation in SMT

Given an equation system E with equations $s_i = t_i$ for which $s, t \in T(\Sigma, X)$ and a goal $s_g = t_g$. Let $\sigma : X \to A$ be a substitution that replaces a variable occurrence with a value from A. To prove that $s_g = t_g$ is non-convertible we must create a set S of SMT formulas such that the following holds:

$$S \Leftrightarrow (\forall_{i,\sigma}[s_i, \sigma] = [t_i, \sigma]) \wedge (\exists_{\sigma}[s_g, \sigma] \neq [t_g, \sigma])$$

Hence we intend to design, based on a non-convertibility problem, a set S of SMT formulas which are satisfiable if en only if the problem is indeed non-convertible.

The set S of SMT formulas that will express a non-convertibility problem can be divided in three parts:

- For each equation in E and every substitution σ the value of $[s, \sigma]$ must be equal to the value of $[t, \sigma]$
- For the goal G and at least one substitution σ the value of $[s_g, \sigma]$ must be unequal to the value of $[t_g, \sigma]$
- Each interpretation function $[f]$ must have an output value that lies inside A

The first two items represent the two requirements in the right hand side of definition 4.2. The third item expresses that the interpretation functions must be of type $A^n \to A$. By theorem 2.2, if S is satisfiable we have proven that $s_g = t_g$ is non-convertible.

We begin the description of the SMT implementation by giving the formulas in S that describe the first two items. For the implementation described in this paper, we define A to be a set of positive integers in the range $[0, k]$. In the previous section we stated that in order to define that two terms, which are compositions of finite interpretation functions, are equal we must define that they are equal for every input value in A. We describe this by means of the following SMT formula where $\sigma : X \to A$:

$$\bigwedge_{\alpha \in \sigma} [s, \alpha] = [t, \alpha]$$

To illustrate how this is implemented in practice we give the following example:

Example 3.2.1 Assume we have the following equation:

$$f(g(x,y)) = g(f(x),y)$$

The variables occurring in this equation are $X = \{x, y\}$. If we assume that $A = \{0, 1\}$, then the SMT formula that expresses that the equation is true, should look as follows:

$[f]([g](0,0)) = [g]([f](0),0)$
$\wedge$
$[f]([g](0,1)) = [g]([f](0),1)$
$\wedge$
$[f]([g](1,0)) = [g]([f](1),0)$
$\wedge$
$[f]([g](1,1)) = [g]([f](1),1)$

Thus the SMT solver is required to find interpretation functions $[f](x)$ and $[g](x, y)$ such that each of the previous equalities hold. Note that we make use of the fact that Yices supports function types [5]. And thus, given a function together with an input value, it will attempt to find a corresponding output value in an attempt to make S satisfiable.

For the goal we have to define that for at least one value in A the equation $s_g = t_g$ does not hold. The SMT formula for the second part then becomes:

$$\bigvee_{\alpha \in \sigma} [s_g, \alpha] \neq [t_g, \alpha]$$

We illustrate the implementation of this formula by means of the following example:

Example 3.2.2 Assume we have the following goal:

$$f(x,y) = f(g(y,x),y)$$

Then we know that $X = \{x, y\}$. If we then assume that $A = \{0, 1\}$, the SMT formula should look as follows:

$[f](0,0) \neq [f]([g](0,0),0)$
$\vee$
$[f](0,1) \neq [f]([g](1,0),1)$
$\vee$
$[f](1,0) \neq [f]([g](0,1),0)$
$\vee$
$[f](1,1) \neq [f]([g](1,1),1)$

To complete the SMT specification we have to express that the output values for each of the interpretation functions $[f] : A^n \to A$ and $[g] : A$ are bounded, to ensure that the interpretation functions are correctly defined over domain A. And thus by definition 3.1 we get the following SMT formulas for functions $[f]$ with arity $n > 0$:

$$\bigwedge_{v_n,\ldots,v_1 \in A} [f](v_n,\ldots,v_1) \geq 0 \quad \wedge \quad \bigwedge_{v_n,\ldots,v_1 \in A} [f](v_n,\ldots,v_1) < |A|$$

and for constant symbols $[g]$:

$$[g] \geq 0 \wedge [g] < |A|$$

In the next section we give a short example illustrating our implementation based on finite interpretation functions.

3.3 Practical example

We revisit the problem from chapter 1 that represented the rules for addition and successor using the following set E of equations:

$$+(0, x) = x$$

$$+(S(x), y) = S(+(x, y))$$

$$+(x, y) = +(y, x)$$

Using these equations we wish to prove that the following goal G is non-convertible:

$$+(S(0), S(0)) = S(0)$$

This goal is obviously non-convertible as $+(S(0), S(0)) = S(0)$ represents $1 + 1 = 1$ which should not follow from the the basic rules for addition and successor. To prove that this is indeed true we use a model with domain size $|A| = 2$. Using this information we then generate the following set of SMT formulas where for every $x \in \{0, 1\}$ and $y \in \{0, 1\}$ the following should hold:

$+(0, x) = x$
$+(S(x), y) = S(+(x, y))$
$+(x, y) = +(y, x)$

$+(S(0), S(0)) \neq S(0)$

$+(x, y) \geq 0 \wedge +(x, y) < 2$
$S(x) \geq 0 \wedge S(x) < 2$

$0 \geq 0 \wedge 0 < 2$

Note that we treat constant 0 as a nullary function which is given a single value. If we now provide this SMT specification to the SMT solver Yices, it will show that this model is indeed satisfiable. Yices gives the following solution:

$+(0, 0) = 0$
$+(0, 1) = 1$
$+(1, 0) = 1$
$+(1, 1) = 0$
$S(0) = 1$
$S(1) = 0$

$0 = 0$

For the non-convertibility problems for which no solution is found, the domain A might not be sufficient. We therefore increase the domain A by one element and repeat the previous process of generating the SMT specification. We keep doing this until either a solution is found, or we stop the process. If still no solution is found, there are three possible reasons:

- A larger domain is required
- The goal actually does follow from the equations in E
- No finite model can satisfy the problem

Dealing with the first item is simply a matter of increasing the maximum domain size. The second item is a direct consequence of the interpretation method: only if the goal is non-convertible does a model exist that satisfies the problem. The third item is however of great interest as it provides us with an opportunity to improve upon what has already been done. In the next section we will investigate an example for which no finite model exists.

3.4 Limitations of finite models

As was stated in the previous section, it is possible that there are non-convertibility problems for which no proofs can be found that use finite interpretation functions. It could also be that some problems require very large domain sizes to be solved. It turns out that for some non-convertibility problems no solution exists that uses finite models. Let us consider the following example where E consists of the following equation:

$$f(g(x)) = x$$

and the goal G is:

$$g(f(x)) = x$$

It should be obvious that the goal is indeed non-convertible. However when we use the approach based on finite models, the SMT solver finds no satisfiable solution for the tested model sizes. This gives an indication that no finite model exists. And indeed non does, as we will prove by means of contradiction.

Let us assume that a finite model exists with the functions $f, g : A \to A$ on the finite domain A that satisfies $\forall_{x \in A}\, f(g(x)) = x$ and $\exists_{x \in A}\, g(f(x)) \neq x$. From this assumption we can conclude that for each x there exists some a for which $f(a) = x$ where $a = g(x)$. This means that f is a surjective function as stated by the following lemma:

Lemma 3.2 Function $f : A \to A$ is surjective if the following holds:

$$(\exists a : a \in A : f(a) = x) \quad \text{for all } x \in A$$

We use the fact that f is surjective to prove that the function g is also surjective. To prove that g is surjective we use the following lemma about functions on a finite domain:

Lemma 3.3 If $g : A \to A$ is a function over the finite domain A the following holds:

$$g \text{ is surjective} \Leftrightarrow g \text{ is injective}$$

This tells us that if we can prove that g is injective, which means that $g(x) = g(y) \Rightarrow x = y$ for all $x, y \in A$, we know it is also surjective. To prove that g is injective we assume $g(x) = g(y)$ and then prove that $x = y$ as follows:

$$
\begin{array}{ll}
 & x \\
= & \{\ f(g(x)) = x\ \} \\
 & f(g(x)) \\
= & \{\ g(x) = g(y)\ \} \\
 & f(g(y)) \\
= & \{\ f(g(x)) = x\ \} \\
 & y
\end{array}
$$

This shows that function g is indeed injective. From the fact that g is an injective function on the finite domain A we can conclude by means of lemma 3.3 that g is also surjective. Function g being surjective, means that there exists an a such that $x = g(a)$. From the previous conclusions about the properties of f and g it is now possible to prove that for all $x \in A$ it holds that $x = g(f(x))$:

$$
\begin{array}{ll}
 & x \\
= & \{\ \text{Surjectivity of } g(x)\ \} \\
 & g(a) \\
= & \{\ f(g(x)) = x\ \} \\
 & g(f(g(a))) \\
= & \{\ \text{Surjectivity of } g(x)\ \} \\
 & g(f(x))
\end{array}
$$

This is clearly a contradiction since we assumed that there exists at least one $x \in A$ for which $g(f(x)) \neq x$. This result proves that no finite model can exists that satisfies both $\forall_x f(g(x)) = x$ and $\exists_x g(f(x)) \neq x$. And thus in order to find a model that can satisfy the problem given in this section we must look into the possibilities of using infinite models. In the next chapter we shall investigate the use of interpretation functions on the infinite domain of the integer numbers.

Chapter 4

Infinite models

As we have proven in the previous section, finite models can only handle a certain set of non-convertibility problems. A small counter example can already show the limitations of the method using finite models. To still be able to solve these problems we investigate the possibility of using infinite domains for the interpretation functions. When using infinite models we consider using functions that work on infinite sets like the natural numbers or even the integers. Whilst any kind of infinite function can be used as an interpretation function, we are limited by the ability of the SMT language to encode them. Therefore we look into the world of polynomial functions, in particular linear functions. We will investigate the usefulness of this type of interpretations and in later sections investigate possible improvements.

4.1 Linear functions

One possible way of defining interpretation functions over an infinite domain, is to use polynomial functions. Since we are intending our method to be implemented by SMT solving, we must take into account the limitations of the SMT solver yices. One of its limitations is that it can not handle SMT formulas that contain multiplications of variables. Hence we have to ensure that the definitions of the interpretation functions only contain multiplications of variables with constant values. Therefore we are limited to using linear polynomials over the integer numbers.

4.2 Theory

Let G be an equation which we intend to prove non-convertible with respect to a set of equations E by using linear interpretation functions. For each function symbol $f \in \Sigma$ with arity n appearing in the equations in E and G we use an interpretation function defined as follows:

Definition 4.1 Let $[f] : \mathbb{Z}^n \to \mathbb{Z}$ be an interpretation function of arity n. We define $[f]$ as follows:

$$[f](v_n, \ldots, v_1) = a_n \cdot v_n + a_{n-1} \cdot v_{n-1} + \ldots + a_1 \cdot v_1 + a_0$$

Thus for example the function $g(x, y, z)$ will be given the following interpretation:

$$[g](x, y, z) = a_3 \cdot x + a_2 \cdot y + a_1 \cdot z + a_0$$

Since our goal is to describe non-convertibility problems by means of SMT formulas, by theorem we have to define that for each equation in E and every substitution $\alpha : X \to \mathbb{Z}$ it holds that:

$$[s, \alpha] = [t, \alpha]$$

and for the goal there exists a substitution $\alpha : X \to \mathbb{Z}$ such that:

$$[s_g, \alpha] \neq [t_g, \alpha]$$

In contrast to the finite interpretation functions used in chapter 3, we now use interpretation functions which are defined by means of a formula. This means that we can express each term by means of a single formula that is the result of expanding all the interpretation functions in this term by their definitions. However since there are infinitely many output values for which the terms of both sides of an equation have to be equal, we cannot express this by enumerating each possibility as SMT formulas. Instead we wish to prove directly from the shape of the formulas for each term that they are equal. This can be done using the following obvious theorem:

Theorem 4.2 Let $f(v_n, \ldots, v_1) = a_n \cdot v_n + \ldots + a_1 \cdot v_1 + a_0$ and $g(v_n, \ldots, v_1) = b_n \cdot v_n + \ldots + b_1 \cdot v_1 + b_1$ be two linear functions whose number of variables is the same. Linear functions f and g are equal if $a_i = b_i$ for all $i \leq n$.

Using this theorem, we can now express that two linear functions are equal just by comparing their coefficients. We will use this to express that $[s, \alpha] = [t, \alpha]$ for all $\alpha : X \to \mathbb{Z}$. Since the terms of an equation are usually compositions of interpretation functions, expressing that two terms are equal first requires us to expand the interpretation functions for the terms on both sides of the equation. Once this is done $[s, \alpha] = [t, \alpha]$ will become an equation of the following shape:

$$L_n \cdot v_n + \ldots + L_1 \cdot v_1 + L_0 = R_n \cdot v_n + \ldots + R_1 \cdot v_1 + R_0$$

Here L_n and R_n are the combinations of coefficients that are multiplied by v_n resulting from the composition of the interpretation functions in s and t respectively. In the next section we use theorem to give the SMT specification of non-convertibility problems using linear interpretation funcions.

4.3 Implementation in SMT

The SMT solver Yices that was used for this paper has the limitation of not being able to solve equations that contain the multiplication of two variables. However as we have seen in the previous section we would like to express that $L_n = R_n$ for each n, where L_n and R_n are multiplications of the coefficients used in the interpretations of the functions that occur in the terms s and t. To still be able to express our problem using SMT equations we have to instantiate some of the coefficients such that L_n and R_n only contain one variable. While this approach allows us to express the problem using the SMT language, it limits the possible values the coefficients can have.

In our approach we choose to instantiate the values for the coefficients that are multiplied by actual variables while we keep the constant coefficients variable. Experimental evidence suggests that the values for the coefficients multiplied by variables usually lie within a small

range of values. Keeping the constant coefficients variable seems to offer more power to our approach. Another reason is that due to the fact that terms are usually compositions of functions, it is very likely that coefficients are multiplied by other coefficients. This requires them to be instantiated within the SMT formulas.

We instantiate the coefficients for a certain bounded range $c_{\max}$. This means that for each combination of values for the coefficients, we must define a separate set of constraints. The total solution then becomes a large disjunction of constraints where one of the possible sets of coefficients should yield a satisfiable solution.

To express the instantiation in SMT, we introduce a substitution $\psi : C \to V$, that substitutes the coefficients in C that appear in the interpretation functions, for some integer value in the finite set V where $V = \{0, \ldots, c_{\max}\}$. So for example if we have $f_0 \cdot x + f_c = g_0 \cdot x + g_c$ this yields the set $C = \{f_0, g_0\}$. If then the set of values is chosen to be $V = \{0, 1\}$ we get the following substitution $\psi = \{f_0 \to 0, f_0 \to 1, g_0 \to 0, g_0 \to 1\}$. Using this substitution we can now describe the SMT formulas that are satisfiable if the problem is non-convertible.

Assume we have an equation system E and a goal $s_g = t_g$. If the functions in each rule are interpreted by means of linear interpretation functions we must first expand the terms for each equation. For the equation $s_i = t_i$ in E this gives the following result:

$$L_{n,i} \cdot v_n + \ldots + L_{1,i} \cdot v_1 + L_{0,i} = R_{n,i} \cdot v_n + \ldots + R_{1,i} \cdot v_1 + R_{0,i}$$

where n is equal to the number of unique variables occurring in the terms s_i and t_i.

To now express this equality by means of SMT formulas, we use theorem 4.2, and express that for each equation in E, the coefficients of each term must be equal:

$$\bigwedge_{i \leq |E|} \left(\bigwedge_{t=0}^{n} L_{t,i}^{\alpha} = R_{t,i}^{\alpha} \right)$$

For the goal we need to express that only one of the combined coefficients is unequal where R_g and L_g are the combined coefficients of the left and right hand side of the goal G respectively:

$$\bigvee_{t=0}^{n} L_{t,g}^{\alpha} \neq R_{t,g}^{\alpha}$$

These formulas must now be generated for each substitution in ψ. This means that the complete set of SMT formulas consist of a large disjunction based on the number of possible values to be substituted for the coefficients. In the next chapter we give a practical example to illustrate what formulas are generated for a given problem.

4.4 Practical example

As an example we will use thee same example as was used in chapter 3 which is given as follows:

$$P(N, x) = x$$

$$P(S(x), y) = S(P(x, y))$$

$$P(x, y) = P(y, x)$$

With the goal:

$$P(S(N), S(N)) = S(N)$$

To improve readability we have replaced the + sign for the character P and the 0 for N. Form these equations we can now identify the functions $P(x, y)$, $S(x)$ and N to which we will assign the following interpretation functions:

$[P](x, y) = a_{P,2} \cdot x + a_{P,1} \cdot y + a_{P,0}$
$[S](x) = a_{S,1} \cdot x + a_{S,0}$
$[N] = a_{N,0}$

Using these interpretation functions we expand the terms of each equation and generate the following set of SMT formulas:

$a_{P,1} = 1$
$a_{P,2} \cdot a_{N,0} + a_{P,0} = 0$

$a_{P,2} \cdot a_{S,1} = a_{S,1} \cdot a_{P,2}$
$a_{P,1} = a_{S,1} \cdot a_{P,1}$
$a_{P,2} \cdot a_{S,0} + a_{P,0} = a_{S,1} \cdot a_{P,0} + a_{S,0}$

$a_{P,2} = a_{P,1}$
$a_{P,1} = a_{P,2}$
$a_{P,0} = a_{P,0}$

$a_{P,2} \cdot a_{S,1} \cdot a_{N,0} + a_{P,2} \cdot a_{S,0} + a_{P,1} \cdot a_{S,1} \cdot a_{N,0} + a_{P,1} \cdot a_{S,0} + a_{P,0} \neq a_{S,1} \cdot a_{N,0} + a_{S,0}$

The complete SMT specification is a large disjunction of the above formulas where the coefficients are replaced by the corresponding numerical value based on the substitution ψ.

The SMT solver Yices then proves that this model is indeed satisfiable. Yices gives the following solution:

$[+](x, y) = x + y$
$[S](x) = x - 1$
$[0] = 0$

4.5 Limitations of linear functions

If we apply our approach of using linear interpretation functions to the example given in section 3.3, it turns out that Yices is not able to find a solution. This means that using linear interpretation functions still is not powerful enough to solve this problem. It is is possible to give an argument using modular arithmetic that proves that every non-convertibility problem we can solve using linear interpretation functions can also be solved using finite interpretation functions. However in the next chapter we will show that limiting the linear functions to the domain of natural numbers gives us the required power to solve the problem from section 3.3.

Chapter 5

Alternative infinite functions

In the previous section we stated that the approach of using infinite models based on linear functions does not add extra power with respect to using finite models. This leads us to investigate the possibility of using alternate types of linear functions. We get a hint of what these functions should look like, by looking for a model that satisfies the problem that was introduced in section 3.1. We restate this problem here, where the equations in E are:

$$f(g(x)) = x$$

and the goal G is:

$$g(f(x)) = x$$

For this problem there was no finite model that could satisfy both the equations and the goal. And in chapter 4 we concluded that there is also no model for this problem based on linear polynomial interpretation functions.

There does however exist a simple model that satisfies the problem. For this model we use linear polynomial interpretation functions that range over the domain of natural numbers. Since these polynomials range over the natural numbers it is possible that for certain input values, the polynomials which have a negative constant coefficient, have a negative output value. This is however not allowed since the domain of these functions is the natural numbers.

As an example assume we have the following interpretation function $[f] : \mathbb{N} \to \mathbb{N}$ where:

$[f](x) = x - 2$

It should be obvious that for the values $x = 0$ and $x = 1$ this function would be negative. However since the output domain of $[f]$ is the natural numbers, the actual output values for $x = 0$ and $x = 1$ cannot be negative. This means that the output values for $[f](0)$ and $[f](1)$ are not defined by the algebraic definition $x - 2$. Hence to complete the definition of $[f]$ these output values have to be defined separately. So for example a correct definition of the interpretation function $[f]$, could be the following:

$[f](0) = 1$
$[f](1) = 0$
$[f](x) = x - 2 \quad$ for all $x > 1$

This new way of defining an interpretation function now gives us the necessary power to

solve the example from section 3.1. We define the following interpretation functions:

$[f](0) = 0$
$[f](x) = x - 1 \quad$ for all $x > 1$

$[g](x) = x + 1 \quad$ for all $x \in \mathbb{N}$

Using these interpretation functions, the equations in E and G are now satisfiable. First we show that the equation in E holds for all $x \in \mathbb{N}$:

$$[f]([g](x)) = x$$

To show that this equation holds for the input values for which the algebraic definition is negative, we use case distinction. For the case where $x = 0$:

$[f]([g](0)) = [f](1) = 0$

And for the case $x > 0$:

$[f]([g](x)) = [f](x + 1) = (x + 1) - 1 = x$

This shows that both $[f]([g](0)) = 0$ and $[f]([g](x)) = x$ hold. For the goal $g(f(x)) = x$ we must prove that for at least one of the values of x the goal does not hold. In this case this is the value $x = 0$:

$[g]([f](0)) = [g](0) = 1 \neq 0$

And this indeed shows that $[g]([f](0)) \neq 0$, which completes the proof.

This example shows that our method becomes more powerful if we limit the linear interpretation functions to the domain of the natural numbers. In the next chapter we shall described the required SMT formulas to solve non-convertibility problems by means of this new class of functions.

Chapter 6

Quasi linear functions

The result from the previous section shows that using interpretation functions which are linear functions restricted to the domain of natural numbers, give us the ability to solve more non-convertibility problems than was previously possible by means of finite models. We shall from now on denote the functions used in the previous section as *quasi linear functions* of level k, where the first k input values have a separately defined output value. In the coming sections we will describe how we can define SMT models of non-convertibility problems using this new type of functions. We shall first describe how we can use these functions on non-convertibility problems for which all functions have a single input argument. Later we shall extend our method to deal with functions of any arity

We start by giving a formal definition of the quasi linear functions, which we denote by $\mathcal{Q}_k$. The class $\mathcal{Q}_k$ of quasi linear functions with level k is defined as follows where $\mathbb{N}_{<k}$ denotes all positive numbers smaller then k:

$$\begin{aligned}\mathcal{Q}_k = \{f : \mathbb{N} \to \mathbb{Z} \mid \exists\, a_1 \in \mathbb{N}, c : \mathbb{N}_{<k} \to \mathbb{N}, a_0 \in \mathbb{Z} \bullet \\ \forall_{x<k}\ f(x) = c(x), \\ \forall_{x\geq k}\ f(x) = a_1 \cdot x + a_0 \wedge \\ (a_1 = 0) \Rightarrow a_0 \geq 0 \wedge \\ (a_1 > 0) \Rightarrow a_0 \geq -k\}\end{aligned}$$

The value k introduced in the definition of $\mathcal{Q}_k$ is used to define for what values of x the function is defined by its algebraic definition, and for what values of x the function has the value of $c(x)$. The function $c(x)$ is used to denote the output values of f in the range $0 \leq x < k$ for which it should hold that $c(x) \geq 0$. The reason that we split the definition in two parts is that we wish to allow the value of a_0 to be negative. However since f is only defined over the domain of natural numbers, having a negative a_0 could lead to possible negative output values for f. That is why we restrict the value of a_0 to being larger then or equal to $-k$. We prove that under these conditions f is always positive:

Lemma 6.1 For all functions $f \in \mathcal{Q}_k$ it holds that $f(x) \in \mathbb{N}$ for all $x \in \mathbb{N}$.

Proof By the definition of quasi linear functions we know that for all values of $x < k$ it holds that $f(x) = c(x)$. Since $c(x) \in \mathbb{N}$ we know that f is always positive for the range

$0 \leq x < k$. For the values $x \geq k$ we have to consider two cases based on the value of a_1. For the case that $a_1 = 0$, f becomes a constant function who's value is equal to a_0 which is in this case defined as $a_0 \geq 0$, and thus always positive. For the case that $a_1 > 0$ we prove that for the values $x \geq k$ it holds that $f(x) = a_1 \cdot x + a_0 \geq 0$. Because the value of $a_1 > 0$ the function is a monotonically increasing function. And thus the smallest input value for f always yields the smallest output value since by the definition of a monotonic function we have that $x \leq y \Rightarrow f(x) \leq f(y)$. Since $x \geq k$ the smallest input value is k. To prove that for all $x \geq k$ it holds that $f(x) \geq 0$, it suffices to prove $f(k) \geq 0$:

$$
\begin{array}{ll}
 & f(k) \\
= & \{ \text{ Definition of } f \ \} \\
 & a_1 \cdot k + a_0 \\
\geq & \{ \ a_0 \geq -k \ \} \\
 & a_1 \cdot k - k \\
\geq & \{ \ a_1 > 0 \ \} \\
 & k - k \\
= & \{ \text{ Mathematics } \} \\
 & 0
\end{array}
$$

□

6.1 Composition

Just as in the chapters about finite and linear interpretation functions, we have to be able to deal with the fact that terms are usually compositions of functions. Since we defined a new class of functions, we must prove that these functions are closed under composition. It is also important to know the properties of such a composition since we have to be able to express compositions of quasi linear functions by means of SMT formulas.

We first show that it is not immediately obvious that the composition of two quasi linear functions also yields a quasi linear function. We show this by means of the following example:

Example 6.1.1 Let $f \in \mathcal{Q}_k$ and $g \in \mathcal{Q}_{k'}$ be two functions whom are defined as follows for which $k = 1$ and $k' = 1$:

$f(0) = 0$
$f(x) = x - 1 \quad$ for all $x \geq k$

$g(0) = 0$
$g(x) = x - 1 \quad$ for all $x \geq k'$

The question is now whether the composition of f and g satisfies that $f \circ g \in \mathcal{Q}_{k''}$. The main issue with answering this question is that we do not know what the level of the composition is. The following example shows that only for a specific value of k'' it holds that $f \circ g \in \mathcal{Q}_{k''}$. Let us for example assume that $k'' = k$, the composition then becomes:

$(f \circ g)(0) = f(g(0)) = 0$
$(f \circ g)(x) = (x - 1) - 1 = x - 2 \quad$ for all $x \geq k$

It should be obvious that for this example it does not hold that $f \circ g \in \mathcal{Q}_k$ since $(f \circ g)(1) = -1$. If we instead choose $k'' = k + k'$ we get the following composition:

$(f \circ g)(0) = f(g(0)) = 0$
$(f \circ g)(1) = f(g(1)) = f(0) = 0$
$(f \circ g)(x) = (x - 1) - 1 = x - 2 \quad$ for all $x \geq k + k'$

For this composition it indeed holds that $f \circ g \in \mathcal{Q}_{k+k'}$ since $(f \circ g)(1) = 0$ and $(f \circ g)(2) = 0$.

We shall now prove that quasi linear functions are indeed closed under composition where for the composition it holds that $k'' = k + k'$

Lemma 6.2 For two quasi linear functions $f \in \mathcal{Q}_k$ and $g \in \mathcal{Q}_{k'}$ it holds that $f \circ g \in \mathcal{Q}_{k+k'}$ where $f \circ g$ satisfies:

$$c_{f \circ g}(x) = f(g(x)) \qquad \text{for all } x < k + k'$$
$$a_{f \circ g,1} = a_{f,1} \cdot a_{g,1}$$
$$a_{f \circ g,0} = a_{f,1} \cdot a_{g,0} + a_{f,0}$$

Proof To prove this we must show that $f \circ g : \mathbb{N} \to \mathbb{N}$. For all values $x < k + k'$ we know that $c_{f \circ g}(x) = f(g(x))$. By the definitions of f and g we know that they are always positive for all $x \in \mathbb{N}$. Therefore since $g(x) \in \mathbb{N}$ it also holds that $f(g(x)) \in \mathbb{N}$ and thus $c_{f \circ g}(x) \geq 0$ for all $x < k + k'$. For the values $x \geq k + k'$ there are two cases dependant on the value of $a_{f \circ g,1}$:

- $a_{f \circ g,1} = 0$

  In this case since $a_{f \circ g,1} = a_{f,1} \cdot a_{g,1} = 0$ we know by the definition of $\mathcal{Q}_k$ that $a_{f \circ g,0} = a_{f,1} \cdot a_{g,0} + a_{f,0} \geq 0$. We now have to prove that for all $x \geq k_f + k_g$ the following holds:

  $$a_{f,1} \cdot a_{g,1} \cdot x + a_{f,1} \cdot a_{g,0} + a_{f,0} \geq 0$$

  By the fact that $a_{f,1} \cdot a_{g,1} = 0$ the inequality becomes $a_{f,1} \cdot a_{g,0} + a_{f,0} \geq 0$ which is true.

- $a_{f \circ g,1} > 0$

  In this case since $a_{f \circ g,1} = a_{f,1} \cdot a_{g,1} > 0$ we know that $a_{f \circ g,0} = a_{f,1} \cdot a_{g,0} + a_{f,0} \geq -k'' = -(k + k')$. We now have to prove that for all $x \geq k + k'$ the following holds:

  $$a_{f,1} \cdot a_{g,1} \cdot x + a_{f,1} \cdot a_{g,0} + a_{f,0} \geq 0$$

  which we prove as follows:

  $a_{f,1} \cdot a_{g,1} \cdot x + a_{f,1} \cdot a_{g,0} + a_{f,0} \geq 0$
  $\Rightarrow$ { $a_{f,1} \cdot a_{g,0} + a_{f,0} \geq -(k + k')$ }
  $a_{f,1} \cdot a_{g,1} \cdot x - (k + k') \geq 0$
  $\Rightarrow$ { $x \geq k + k'$ }
  $a_{f,1} \cdot a_{g,1} \cdot (k + k') - (k + k') \geq 0$
  $\Rightarrow$ { $a_{f,1} \cdot a_{g,1} > 0$ }
  $(k + k') - (k + k') \geq 0$
  $\Rightarrow$ { Tautology }
  True

□

This proves that the output value of $f \circ g$ will always be positive and thus a natural number. By proving this lemma we have now shown that for each possible composition of two quasi linear functions the composition is also quasi linear function albeit with a different level.

6.2 SMT specification using quasi linear functions

Now that we have the new class of interpretation functions $\mathcal{Q}_k$, and have proven that they are closed under composition, we can use them with the interpretation method to prove non-convertibility. In this chapter we will describe the required SMT formulas that express a non-convertibility problem by means of quasi linear interpretation functions.

In chapter 4 where we used linear interpretation functions over the integers, we could express that the terms of an equation are equal, by means of comparing the coefficients of the result of composing all the functions in each term. This method can however not be directly applied for functions of type $\mathcal{Q}_k$. We have to take into account that the first k values of these functions are not defined by the algebraic definition of their function. We illustrate the issues involved when expressing the composition of quasi linear functions by means of the following example:

Example 6.2.1 Let $s = h(f(g(x)))$ and $t = f(h(x))$ be the terms of the equation $s = t$ and let $\alpha = \{x \to 0\}$. We now wish to express the following:

$$[h(f(g(x))), \alpha] = [f(g(x)), \alpha]$$

which becomes:

$$[h]([f]([g](0))) = [f]([h](0))$$

The question that now arises is: how do we express $[h]([f]([g](0)))$ such that we can equate it to $[f]([h](0))$? The answer is that this is entirely dependant on the definitions of the interpretation functions $[f]$, $[h]$ and $[g]$. For example if $[g](0) = 1$ then:

$[h]([f]([g](0))) = [h]([f](1))$

If $[f]$ is defined such that $k_f = 1$ we know that $[f](1) = f_1 \cdot 1 + f_0$ which means:

$[h]([f]([g](0))) = [h]([f](1)) = [h](f_1 \cdot 1 + f_0)$

It could be that $f_1 \cdot 1 + f_0 = 0$ and $[h](0) = 9$, by which finally we get:

$[h]([f]([g](0))) = [h]([f](1)) = [h](f_1 \cdot 1 + f_0) = [h](0) = 9$

The previous example is just a single possible instance of the outcome of $[h]([f]([g](0)))$. However for the general case we must express that for an equation $s = t$ in E and every substitution $\alpha = \{X \to \mathbb{N}\}$ the following holds:

$$[s, \alpha] = [t, \alpha]$$

To express this equality by means of SMT formulas, we would have to create a formula for every possible outcome of $[s, \alpha]$ and $[t, \alpha]$ for the substitutions α up to a value that is equal to the level of the compositions representing the term s and t. Expanding each term as we did in example 6.2 for the required substitutions α is possible, but not very attractive. This method creates unnecessarily large SMT specifications with a lot of redundancy. For example assume we have two terms $h(f(g(x)))$ and $l(f(g(x)))$. Here the composition $f(g(x))$ appears twice. If we just expand both terms in a direct way we would have to express all choices for $f(g(x))$ twice. However if we would define a new composition $R(x) = f(g(x))$ separately, and then state both terms in terms of this composition as $h(R(x))$ and $l(R(x))$, we then have to state the definition of $R(x)$ only once.

Defining compositions separately gives us a more convenient way of defining the equality of terms. Let t be a term consisting of the composition of several functions. If we now wish to express $[t, \alpha]$ by means of quasi linear interpretation functions we do this as follows where $[f_i]$ denotes the i'th interpretation function occurring in t and each composition of two functions is denoted by $[R_i]$:

- $[R_0](x) = [f_1]([f_0](x))$
- $[R_i](x) = [f_{i+1}]([R_{i-1}](x))$

The advantage of using this way of defining a term t is that each term is represented by a single interpretation function, namely the interpretation function that is equal to the composition of all functions in t. if we now wish to express that for a certain substitution α it has to hold that:

$$[s, \alpha] = [t, \alpha]$$

we can now express this as follows:

$$R_s^\alpha = R_t^\alpha$$

This the brings us to stating the general shape of the SMT equations. Let E be a set of equations of the shape $s_n = t_n$ where n denotes the n'th equation and let G be the goal $s_g = t_g$. We now define the SMT formulas that will be satisfiable if G is non-convertible with respect to E as follows:

Theorem 6.3 A goal G equal to $s_g = t_g$ is non-convertible with respect to a set of equations E if the set S of SMT formulas is satisfiable, where S is given as follows:

- For each composition $[R]$ that occur in the terms in E and G we calculate what the values of $c(x)$ and its coefficients will be, based on the definitions of the composed functions.
- For each equation $s = t$ in E and every substitution σ the value of $[s, \sigma]$ must be equal to the value of $[t, \sigma]$.
- For the goal G and at least one substitution σ the value of $[s_g, \sigma]$ must be unequal to the value of $[t_g, \sigma]$
- We bound the values of the symbols used in the interpretation functions, such that each function are of type $\mathbb{N} \to \mathbb{N}$

Note that in item 4 we did not add the requirement that the symbols used for the compositions are bound as well. By lemma 6.2 we proved that the composition of two quasi linear functions is also a quasi linear function. Hence all symbols used in this composition are correctly bounded by default. If by item 1 each term is now expressed by means of compositions, we can now express items 2 and 3 and by theorem 2.2 this means that the given problem is non-convertible if the SMT formulas are satisfiable.

In the next section we describe how to express the compositions that occur in the terms in E and G by means of SMT formulas:

6.3 Compositions in SMT

In this section, we specify the SMT formulas used to describe a composition $[R]$ of two interpretation functions $[f]$ and $[g]$ with levels k and k' respectivly. The definition of $[R]$ can be divided in two parts. We shall first describe the definition of $[R]$ for the values $0 \leq x < k''$ where as by lemma 6.2 the level k'' of the composition is equal to $k + k'$. To define the output value of $[R](x)$, we must know what the value of $[f]([g](x))$ is. We begin by expressing $[R](x)$ for the values of x which are smaller then k' as for these value $[g](x)$ is equal to $c_g(x)$:

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{j=0}^{k-1} (c_g(i) = j) \Rightarrow [R](i) = c_f(j)$$

We express here that if $c_g(i)$ is equal to some value j smaller then k, by the definition of $[f]$ we then know that the output value of $[R](i)$ is equal to $c_f(j)$. It is also possible that $c_g(i)$ has a value at least that of the level of $[f]$. For these cases we express the following SMT formula:

$$\bigwedge_{i=0}^{k'-1} (c_g(i) \geq k) \Rightarrow [R](i) = a_{f,1} \cdot c_g(i) + a_{f,0}$$

We also have to account for the fact that the level of $[R]$ is bigger then then individual levels of $[f]$ and $[g]$. So for these input values we give an additional SMT formula. In the case that the output value of $[g]$ is smaller then k:

$$\bigwedge_{i=k'}^{(k+k')-1} \bigwedge_{j=0}^{k-1} (g_1 \cdot i + g_0 = j) \Rightarrow [R](i) = c_f(j)$$

And finally we express the case where the output value of $[g]$ is greater then or equal to k:

$$\bigwedge_{i=k'}^{(k+k')-1} (a_{g,1} \cdot i + a_{g,0} \geq k) \Rightarrow [R](i) = a_{f,1} \cdot (a_{g,1} \cdot i + a_{g,0}) + a_{f,0}$$

With these four formulas we have completely defined the output values of $[R]$ for the values $0 \leq x < k + k'$. For the values $x \geq k + k'$ we can directly use lemma 6.2 which tells us that the composition of two quasi linear functions satisfies the following:

$$[R](x) = a_{f,1} \cdot a_{g,1} \cdot x + a_{f,1} \cdot a_{g,0} + a_{f,0} \qquad \text{for all } \; x \geq k + k'$$

However before we can express the SMT formula for this, we must take into account that it is possible for $[g]$ to be a constant function. If this is the case, it could happen that if $x \geq k + k'$,

$f_1 > 0$ and $g_1 = 0$, the function $[f]$ is fed an input value lower then $k + k'$. We illustrate this by means of an example: let $[f], [g] \in \mathcal{Q}_2$ be two interpretation functions who are defined as follows:

$$[f](x) = x + 1 \quad \text{for all } x$$
$$[g](x) = 4 \quad \text{for all } x$$

Assume now that we wish to express the value of $[f]([g](3))$. If we expand this composition by means of the definitions of $[f]$ and $[g]$ we get the following result:

$$[f]([g](3)) = [f](4) = a_{f,1} \cdot 4 + a_{f,0}$$

If however $[g]$ was defined as follows:

$$[g](x) = 1 \quad \text{for all } x$$

The composition $[f]([g](3))$ would become:

$$[f]([g](3)) = [f](1) = c_f(1)$$

This means that formula of $[R]$ for the case that $a_{g,1} = 0$ is different then that for when $a_{g,1} > 0$. This happens due to the fact that it is dependant on the value of $a_{g,0}$ as shown in the example. And thus we have to incorporate this into the SMT formula that describes $[R]$ for the values $x \geq k_R$:

$$a_{g,1} = 0 \Rightarrow \begin{cases} \bigwedge_{i=0}^{k-1} ((a_{g,0} = i) \Rightarrow a_{R,1} = 0 \wedge a_{R,0} = c_f(i)) \\ \\ (a_{g,0} \geq k_f) \Rightarrow (a_{R,1} = 0 \wedge a_{R,0} = a_{f,1} \cdot a_{g,0} + a_{f,0}) \end{cases}$$

$$a_{g,1} > 0 \Rightarrow \begin{cases} a_{R,1} = a_{f,1} \cdot a_{g,1} \\ a_{R,0} = a_{f,1} \cdot a_{g,0} + a_{f,0} \end{cases}$$

With these SMT formulas we have completely specified the composition of two quasi linear interpretation functions. In the next chapter we describe the complete SMT specification for specifying a non-convertibility problem by means of quasi linear functions.

6.4 Complete SMT specification

Now that we have the SMT formulas to describe a composition of quasi linear functions we can now give the complete SMT specification as was given in definition 6.3. In order to express the first item in definition 6.3 we begin by extracting the unique compositions of interpretation functions present in the terms of the equations in E and G. Each composition is then expressed by using the SMT formulas developed in the previous section.

Each term present in E or G that is a composition of some functions, can now be expressed by means of the composition that represents the whole term. Let $s = t$ be a term in E or

G, then $[R]$ and $[L]$ are the compositions of the interpretation functions present in s and t respectively. We now express items 2 and 3 of definition 6.3 by creating SMT formulas that express that for a given equation $s = t$ in E the compositions representing s and t are equal for every value of x. Note that it is possible that the levels of $[R]$ and $[L]$ are different. We therefore first determine which of the two compositions has the highest level. We denote this level by:

$$k = \max(k_R, k_L)$$

For the values $0 \leq x < k$ the SMT formulas for each equation in E given by:

$$\bigwedge_{i=0}^{k-1} [R](i) = [L](i)$$

And for the values $x \geq k$ we express that all the coefficients of the compositions representing s and t are equal:

$$a_{R,1} = a_{L,1}$$

$$a_{R,0} = a_{L,0}$$

To express that the goal G does not hold for one of the values of x, we must express that $\exists_\alpha [s_g, \alpha] \neq [t_g, \alpha]$. We again express this separately for the values $0 \leq x < k$ as follows:

$$\bigvee_{i=0}^{k-1} [R](i) \neq [L](i)$$

and for the values $x \geq k$:

$$a_{R,1} \neq a_{L,1}$$

$$a_{R,0} \neq a_{L,0}$$

However not all terms will be compositions of interpretation functions. It is also possible that a term is a single function or a variable. This yields a total of four addition possible SMT equations specifying the separate cases:

- $s = [f](x)$ and $t = x$

$$\left(\bigwedge_{i=0}^{k-1} [f](i) = i \right) \wedge a_{f,1} = 1 \wedge a_{f,0} = 0$$

- $s = [f](x)$ and $t = [g](x)$ where $k = \max(k_f, k_g)$

$$\left(\bigwedge_{i=0}^{k-1} [f](i) = [g](i) \right) \wedge a_{f,1} = a_{g,1} \wedge a_{f,0} = a_{g,0}$$

- $s = [R](x)$ and $t = x$

$$\left(\bigwedge_{i=0}^{k-1} [R](i) = i \right) \wedge a_{R,1} = 1 \wedge a_{R,0} = 0$$

- $s = [R](x)$ and $t = [g](x)$ where $k = \max(k_R, k_g)$

$$\left(\bigwedge_{i=0}^{k-1} [R](i) = [g](i)\right) \wedge a_{R,1} = a_{g,1} \wedge a_{R,0} = a_{g,0}$$

The SMT formulas for the goal G are similar but with inequalities and disjunction quantifiers since they have to hold for only one value of x.

To complete the specification we must create the SMT formulas for item 4 of definition 6.3. Item 4 requires us to add bounds to all the symbols used in the definitions of the interpretation functions. First we bound the coefficients for each interpretation function $[f]$ with level k:

$$\bigwedge_{f \in \Sigma} a_1 \geq 0 \wedge a_0 \geq -k$$

Next we express that constant functions must always be positive:

$$\bigwedge_{f \in \Sigma} a_1 = 0 \Rightarrow a_0 \geq 0$$

As stated by the definition of functions of type $\mathcal{Q}_k$ we must also define that for every interpretation function $[f]$, the value of $c(x)$ is always positive:

$$\bigwedge_{f \in \Sigma} \left(\bigwedge_{i=0}^{k-1} c(i) \geq 0\right)$$

Just as in chapter 4 we have to deal with the fact that Yices can not solve SMT formulas containing multiplications of variables. The SMT specification given in this chapter does however contain the multiplications. We solve this by instantiating the coefficients a_1 for each interpretation function $[f]$ for a values in the range $\{0, \ldots, c_{max}\}$. We then generate the SMT formulas given in this chapter for every possible combination of the values of the coefficients. The complete SMT formula generated, is then a large disjunction based on the values of the coefficients.

6.5 Practical example

We shall now use the problem given in section 3.4 to show what kind of model is generated and what solution is found by Yices. Let E be:

$$f(g(x)) = x$$

and the goal G be:

$$g(f(x)) = x$$

For all terms that are a composition of functions, we first identify all the compositions contained in these terms:

$R_1(x) = f(g(x))$
$R_2(x) = g(f(x))$

For this problem we assume that the interpretation functions of f and g are of level 1. If we also only allow the values of the coefficients to be equal to 1, Yices will show this SMT specification to be satisfiable, and provides us with the following result:

$[f](0) = 2$
$[f](x) = x - 1$

$[g](0) = 1$
$[g](x) = x + 1$

Note that this answer is not equal to the one given in section 3.4. This is due to the fact that many possible solutions exists, while Yices only gives the first solution it finds.

For this example we chose the level of the interpretation functions to be equal to one. However in general we can say that if a solution is found for level k, it also means that solutions exist for levels greater then k. We can show this for our example by generating a new set of SMT equations where each interpretation function has level 5. For this instance Yices yields the following solution:

$[f](0) = 0$
$[f](1) = 7$
$[f](2) = 0$
$[f](3) = 0$
$[f](4) = 10$
$[f](x) = x - 4$

$[g](0) = 0$
$[g](1) = 5$
$[g](2) = 6$
$[g](3) = 7$
$[g](4) = 8$
$[g](x) = x + 4$

While we can now express any problem that contains terms who's functions have arity 1, we cannot solve a simple problem like the following, where E is:

$+(0, x) = x$
$+(S(x), y) = S(+(x, y))$
$+(x, y) = +(y, x)$

with goal G given as:

$+(S(0), S(0)) = S(0)$

It is therefore desirable that we can express non-convertibility problems which contain functions of arity larger then 1. In the next chapter we extend the definition of $\mathcal{Q}_k$ to a class of quasi linear functions of arbitrary arity.

Chapter 7

Extension to n-ary quasi linear functions

So far we have described how we can solve non-convertibility problems for which the functions had at most arity 1. In this chapter we will extend our approach to be able to deal with functions of arbitrary arity. Just as in chapter 6 we intend to use quasi linear functions. However we have to extend the definition of quasi linear functions such that these functions have arbitrary arity. The class of quasi linear functions $\mathcal{Q}_{k,n}$ of level k with arity n is defined as follows:

$$\begin{aligned}\mathcal{Q}_{k,n} = \{f : \mathbb{N}^n \to \mathbb{Z} \mid \exists\, a_n, \ldots a_1 \in \mathbb{N}, c : \mathbb{N}^n_{<k} \to \mathbb{N}, a_0 \in \mathbb{Z} \bullet \\ \forall_{v_n<k \,\wedge\, \ldots \,\wedge\, v_1<k}\ f(v_n, \ldots, v_1) = c(v_n, \ldots, v_1), \\ \forall_{v_n\geq k \,\vee\, \ldots \,\vee\, v_1\geq k}\ f(v_n, \ldots, v_1) = a_n \cdot v_n + \ldots + a_1 \cdot v_1 + a_0 \,\wedge \\ (a_n = 0) \vee \ldots \vee (a_1 = 0) \Rightarrow a_0 \geq 0 \,\wedge \\ (a_n > 0) \wedge \ldots \wedge (a_1 > 0) \Rightarrow a_0 \geq -k\}\end{aligned}$$

Just as in chapter 6 we use the value of k to ensure that the output of f is always positive. The function $c(v_n, \ldots, v_1)$ is again defined to be greater then zero, for every set of input values. For the case that the values of the coefficients a_n to a_1 are chosen to be both then one, the value of $a_{f,0}$ is allowed to be negative up to $-k_f$. If at least one of these coefficients is zero, we no longer allow a_0 to be negative. Using the results from chapter 6 we prove that n-ary quasi linear functions always produce an output in the natural numbers:

Lemma 7.1 For all functions $f \in \mathcal{Q}_{k,n}$ it holds that $f(v_n, \ldots, v_1) \in \mathbb{N}$ for all $v_n, \ldots, v_1 \in \mathbb{N}$.

Proof By the definition of f we know that if $\forall_{1\leq i\leq n}\ v_i < k$ it holds that $[f](v_n, \ldots, v_1) = c_f(v_n, \ldots, v_1)$. Since $c_f(v_n, \ldots, v_1) \geq 0$ we know that f is always positive. For the case that $\exists_{1\leq i\leq n}\ v_i \geq k$ the output value of f is dependant on the values of its coefficients. If now one of the coefficients a_n to a_1 are equal to zero, we know that $a_0 \geq 0$ and thus that the output of f is always positive. If all of the coefficients a_n to a_1 are greater then zero, we know that $a_0 \geq -k$. In this case we have to prove the following:

$$a_n \cdot v_n + \ldots + a_1 \cdot v_1 + a_0 \geq 0$$

Since all the coefficients are positive, f is a monotonic function which means that the smallest input value yields the smallest output value. Therefore since there is some variable $v_s \geq k$ the smallest possible input value for f is when $\forall_{1 \leq i < s \wedge s < i \leq n}\ v_i = 0$ and thus we get:

$$a_s \cdot v_s + a_0 \geq 0$$

We can now see that this is one of the cases from the definition of $\mathcal{Q}_k$. Therefore by lemma 6.1 this is true. □

7.1 Compositions of multi-variable functions

Just as in chapter 6, in order for the quasi linear functions to be useful, we must prove that they are closed under composition and know what their properties are. Let $f : \mathbb{N}^n \to \mathbb{N}$ and let $g_i : \mathbb{N}^{n_i} \to \mathbb{N}$ where $1 \leq i \leq n$. We denote f as the outer function of a composition and g_1 to g_n as the inner functions. The composition of f and the functions g_1 to g_n is now the result of applying the outputs of the functions g_1 to g_n to the inputs of f. We can denote the result of applying the functions g_1 to g_n to a given set of input values by means of the function:

$$(g_1 \dots g_n) : \mathbb{N}^{\sum_{i=1}^{n} n_i} \to \mathbb{N}^n$$

The composition of the functions f and g_1 to g_n can now be given as follows:

$$f \circ (g_1 \dots g_n) : \mathbb{N}^{\sum_{i=1}^{n} n_i} \to \mathbb{N}$$

From this we can see that the composition of n-ary functions has an arity equal to the sum of the arities of its inner functions. We illustrate this by means of the following example:

$$(f \circ (g, h))(x, y, i, j) = f(g(x, y), h(i, j))$$

While the inner functions g and h have only arity two, the composition of f with these functions actually has an arity of four.

We shall now prove that n-ary quasi linear functions are indeed closed under composition where such a composition has a level k' given as follows:

$$k' = k + \sum_{i=1}^{n} k_i$$

where k is the level of f and k_i is the level of g_i. We shall prove this by means of the following lemma:

Lemma 7.2 Let $f \in \mathcal{Q}_{k,n}$ and $\forall_{1 \leq i \leq n}\ g_i \in \mathcal{Q}_{k_i,n_i}$ be quasi linear functions. The composition $R = f \circ (g_n, \dots, g_1)$ with arity r then satisfies:

$$R \in \mathcal{Q}_{k',r}$$

for which $k' = k + \sum_{i=1}^{n} k_i$ and $r = \sum_{i=1}^{n} n_i$.

Proof To prove this we must show that $R : \mathbb{N}^r \to \mathbb{N}$. If for all input values of R hold that they are smaller then k' we know that $c(v_r, \dots, v_1) = (f \circ (g_n, \dots, g_1))(v_r, \dots, v_1)$. Since all the functions g_n to g_1 are quasi linear functions, we know that they always produces a positive output value for any combination of input values. And since f is also a quasi linear function it therefore must hold that $c(v_r, \dots, v_1) \geq 0$. If one of the variables is at least k' there are two different cases dependant on the values of the coefficients of R:

- $(a_r = 0) \vee \ldots \vee (a_1 = 0)$

  In this case as one of the coefficients of R is equal to zero we know that $a_0 \geq 0$. Therefore since we also know that all of the coefficients have a positive value, R is always positive.

- $(a_r > 0) \wedge \ldots \wedge (a_1 > 0)$

  Since all coefficients are greater then zero we know that $a_0 \geq -k'$. We now must prove that if for least one variable v_s its value is $v_s \geq k'$, the following holds:

  $$a_r \cdot v_r + \ldots + a_1 \cdot v_1 + a_0 \geq 0$$

  which we prove as follows:

  $a_r \cdot v_r + \ldots + a_1 \cdot v_1 + a_0 \geq 0$
  $\Rightarrow$ { $\forall_{1 \leq i \leq r}\ a_i > 0$ }
  $v_r + \ldots + v_1 + a_0 \geq 0$
  $\Rightarrow$ { $a_0 \geq -k'$ }
  $v_r + \ldots + v_1 - k' \geq 0$
  $\Rightarrow$ { Since $\exists\ v_s \in X : v_s \geq k'$, R is smallest when all other variables are zero }
  $v_s - k' \geq 0$
  $\Rightarrow$ { $v_s \geq k'$ }
  $k' - k' \geq 0$
  $\Rightarrow$ { Tautology }
  True

With this proof we have now proven that the quasi linear functions of arbitrary arity are closed under composition. With this knowledge we can describe the SMT formulas required to specify non-convertibility problems that contain functions with any arity.

7.2 SMT Specification

The SMT specification of non-convertibility problems using n-ary quasi linear interpretation functions follows the same structure as was described in chapter 6. We again use theorem 6.3 as its validity has not changed due to the fact that we are using n-ary quasi linear functions. Therefore we start with item 1 from theorem 6.3 and describe the SMT formulas required for the composition of n-ary quasi linear functions. In order to generate the formulas for the composition $[R] = [f] \circ ([g_n], \ldots, [g_1])$ of n-ary quasi linear functions, we require the following information:

- The arity n of the outer function $[f]$
- The arities $ar(g_i)$ of the inner functions $[g_i]$
- The set of variables X_i appearing in each inner function $[g_i]$
- The complete set $X = \bigcup_i X_i$ containing all variables appearing in the composition.

- The level k' of $[R]$ which is equal to $k + \sum_{i=1}^{n} k_i$ where k is the level of f and k_1 to k_n are the levels of the inner functions g_1 to g_n

As in chapter 6 we divide the specification of the composition $[R]$ in two parts. In the first part we express the output values of $[R]$ for all possible combinations of the input values smaller then k'. Since we are dealing with the substitution of possibly multiple variables, for ease of notation of the SMT formulas we introduce the following substitution:

$$\sigma : X \to \mathbb{N}_{<k'}$$

We can now express that the variables of $[R]$ are substituted by values smaller then k' by using the notation $[R]^\sigma$, e.g. given the function $[R](x, y)$, if $\sigma = \{x \to 0, y \to 1\}$ then $[R]^\sigma = R(0, 1)$. Additionally we define a set of all possible combinations of the output values produced by the inner functions. This set P is defined as follows:

$$P = \{(x_n, \ldots, x_1) \in \mathbb{N}^n_{<k}\}$$

where $P_{i,j}$ is element j of entry i of P. Note that we bound P by the level of f, since in the case that one of the values of P is at least k we use a separate SMT formula. For every substitution σ the composition $[R]$ is now denoted by the following SMT formula:

$$\bigwedge_{i=1}^{|P|} \left(\bigwedge_{j=1}^{n} [g_j]^\sigma = P_{i,j} \right) \Rightarrow [R]^\sigma = [f](P_{i,n}, \ldots, P_{i,1})$$

This formula expresses that if for the input values given by σ, all of the output values of the interpretation functions $[g_n]$ to $[g_1]$ are equal to the corresponding values in P_i, then $[R]^\sigma$ is equal to the output of $[f]$ for the values in P_i. By the definition of quasi linear functions, the actual shape of the expressions $[g_j]^\sigma$ and $[f](P_{i,n}, \ldots, P_{i,1})$ is dependant on the values in σ and P_i. If all the values for which the variables in $[g_j]$ are substituted, are smaller then k_j then the shape of $[g_j]^\sigma$ becomes where $m = ar(g_j)$:

$$c(v^\sigma_m, \ldots, v^\sigma_1)$$

If at least one of substituted the values is greater then of equal to k_j the shape of $[g_j]^\sigma$ becomes:

$$a_m \cdot v^\sigma_m, \ldots, a_1 \cdot v^\sigma_1 + a_0$$

To complete the specification of $[R]$ we must also create an SMT formula for the case that at least one of the output values of the inner functions is greater then or equal to k. For every substitution σ this SMT formula is given as follows:

$$\left(\bigvee_{j=1}^{n} [g_j]^\sigma \geq k \right) \Rightarrow [R]^\sigma = a_{f,n} \cdot [g_n]^\sigma + \ldots + a_{f,1} \cdot [g_1]^\sigma + a_{f,0}$$

Just as for the SMT formula where all outputs of the inner functions were smaller then k, $[g_j]^\sigma$ has to be replaced by the correct definition depending on the values of its inputs.

Next we must describe the second part of the specification of $[R]$. If at least one of the input values for $[R]$ is greater then or equal to k', the output of $[R]$ is given by its formula. Just as in chapter 6 we must take care of the case that all the inner functions are constant functions. If that is the case, $[R]$ itself becomes a constant function who's value is dependant on the value of $a_{f,0}$. The SMT formula that describes $[R]$ for the case that at least one of its input values is greater then or equal to k' is then given as follows:

- $[g_r] = a_{g_r,0} \wedge \ldots \wedge [g_1] = a_{g_1,0}$

  $$\bigwedge_{i=0}^{|P|} \left(\bigwedge_{j=1}^{r} a_{g_j,0} = P_{i,j} \right) \Rightarrow a_{R,r} = 0 \wedge \ldots \wedge a_{R,1} = 0 \wedge a_{R,0} = c_f(P_{i,r}, \ldots, P_{i,1})$$

  $$\left(\bigvee_{j=1}^{r} a_{g_j,0} \geq k_f \right) \Rightarrow a_{R,r} = 0 \wedge \ldots \wedge a_{R,1} = 0 \wedge a_{R,0} = a_{f,r} \cdot a_{g_r,0} + \ldots + a_{f,1} \cdot a_{g_1,0} + a_{f,0}$$

- $[g_r] \neq a_{g_r,0} \vee \ldots \vee [g_1] \neq a_{g_1,0}$

  In this case, in order to get the values of the coefficients of $[R]$ we must expand the formula of $[R]$ which is given as follows:

  $$[R] = a_{f,n} \cdot [g_n] + \ldots + a_{f,1} \cdot [g_1] + a_{f,0}$$

  Once this definition is expanded we get the following result where $r = \sum_{i=1}^{n} n_i$:

  $$[R](v_r \ldots v_1) = L_r \cdot v_r + \ldots + L_1 \cdot v_1 + L_0$$

  With this result the coefficients of R then become:

  $$\bigwedge_{i=0}^{r} a_{R,i} = L_i$$

This gives us the complete definition of a composition of n-ary quasi linear interpretation functions.

Having specified the SMT specification of a composition, we can now express items 2 and 3 of theorem 6.3. To be able express the equality of an equation $s = t$ in E where R and L are the compositions representing the terms s and t, we need use the substitution σ defined as follows:

$$\sigma : X \to \mathbb{N}_{<k}$$

where $X = X_R \cup X_L$ are all the variables occurring in s and t and $k = \max(k_R, k_L)$. The equality of a term in E is then expressed as follows:

$$\bigwedge_{\alpha \in \sigma} [R]^\alpha = [L]^\alpha$$

For the case one of the variables is greater then or equal to k we specify that all coefficients are equal:

$$\bigwedge_{v \in X} a_{R,v} = a_{L,v} \qquad \text{where } v \notin X_R \Rightarrow a_{R,v} = 0 \text{ and } v \notin X_L \Rightarrow a_{L,v} = 0$$

For the goal $s_g = s_t$ we get similar SMT formulas:

$$\bigvee_{\alpha \in \sigma} [R]^\alpha \neq [L]^\alpha$$

$$\bigvee_{v \in X} a_{R,v} \neq a_{R,v} \qquad \text{where } v \notin X_R \Rightarrow a_{R,v} = 0 \text{ and } v \notin X_L \Rightarrow a_{R,v} = 0$$

As we saw in chapter 6 it is possible that a term is not a composition but either a function on some variables or a single variable. We express the equality of terms between compositions, functions and variables in a similar fashion as in chapter 6 with the difference that we have to deal with multiple variables.

Finally we express item 4 of theorem 6.3 where we must assign bounds to all the symbols used. First we bound the coefficients for each function $f \in \Sigma$ were we use the assumption that the ranges of the functions are all equal to k and where $n = ar(f)$:

$$\bigwedge_{f \in \Sigma} a_n \geq 0 \wedge, \ldots, \wedge\, a_1 \geq 0 \wedge a_0 \geq -k$$

To ensure that constant functions are always positive we require the following SMT formula:

$$\bigwedge_{f \in \Sigma} (a_n = 0 \wedge, \ldots, \wedge\, a_1 = 0) \Rightarrow a_0 \geq 0$$

And finally by the definition of $\mathcal{Q}_{k,n}$ we must also specify that for all the input values $P = \{(x_n, \ldots, x_1) \in \mathbb{N}^n_{<k}\}$ of a function $f \in \Sigma$, f must be positive:

$$\bigwedge_{i=1}^{|P|} c_f(P_{i,n}, \ldots, P_{i,1}) \geq 0$$

To prevent the SMT formulas from containing multiplications of symbols we instantiate the coefficients for the functions occurring in the terms in E and G by constant values. This means that the previous SMT specification has to be repeated for every combination of the coefficients up to a certain maximum coefficient value $c_{\max}$.

7.3 Results

Having specified the SMT formulas to express a non-convertibility problem containing n-ary functions, we should now be able to express the problem of successor and addition mentioned in previous chapters given as follows:

$+(0, x) = x$
$+(S(x), y) = S(+(x, y))$
$+(x, y) = +(y, x)$

with goal G:

$+(S(0), S(0)) = S(0)$

If we assume that all functions have level 1, and the coefficients are limited to having the value 1, Yices will find these formulas satisfiable and gives the following result:

$[+](0, 0) = 0$

$[+](x, y) = x + y$

$[0] = 0$

$[S](0) = 1$
$[S](x) = x + 1$

This is however a non-convertibility problem that could also be solved by means of finite models. Next we give an example with n-ary functions for which no finite model seems to exist. Let E be:

$f(g(x, y)) = x$

and the goal G be:

$g(f(x), y) = x$

If each function has arity 1 and the range of values for the coefficients is between zero and one, Yices will solve this SMT specification, and return the following result:

$[g](0, 0) = 1$
$[g](x, y) = x + 1$

$[f](0) = 2$
$[f](x) = x - 1$

Note that for this example it was required to choose the range of values for the coefficients to be between zero and one since for $[g]$ the coefficient a_1 is equal to zero.

To be able to automatically solve non-convertibility problems by means of the SMT specifications for finite and infinite models, we have developed the tool NCSolver that can parse any non-convertibility problem and turn it into an SMT specification. NCSolver then uses Yices to determine whether the SMT specifications are satisfiable or not, and displays the output Yices produces. In the next chapter we shall discuss NCSolver and investigate its performance with respect to Mace4.

Chapter 8

NCSolver

In order to measure the performance and possibilities of our approach using SMT solving, a Java tool called NCSolver was created that can parse any non-convertibility problem and generates SMT specifications using both finite and infinite models. NCSolver then uses the SMT solver Yices to prove satisfiability of the generated SMT formulas. If the SMT formulas are found satisfiable, NCSolver informs the user of the shape of the used interpretation functions for proving non-convertibility of the given problem. The use of finite models was implemented as a means to compare NCSolver with that of the reference tool Mace4. As the main novelty, for problems that cannot be solved using finite models, NCSolver can try to find infinite models based on quasi linear interpretation functions. For each non-convertibility problem, the user has the option to select whether the tool has to search for finite or infinite models. Depending on the kind of model, certain options are available to configure the model searching. We shall begin by discussing the option of proving non-convertibility by means of finite models.

8.1 Finite model solving

For solving non-convertibility problem using interpretation functions over a finite domain, NCSolver generates SMT specifications using finite interpretation functions of increasing domain size. The user has to supply a maximum domain size such that NCSolver knows up to what model size it has to generate SMT specifications, before it should stop. The best maximum model size is somewhat arbitrary. If the maximum model size is chosen too large it is possible that Yices takes too long to give a verdict on the satisfiability of the generated SMT formulas. It is in the nature of SAT problems, to have a certain size for which the time required to finish analysis becomes unfeasible. In practice however it appears that solving non-convertibility problems a maximum model size of 5 seems to be a good estimate, such that Yices completes analysis within a reasonable amount of time. Nevertheless for small problems it is certainly possible to use larger model sizes.

To measure the performance of our approach using finite models, we let both NCSolver and Mace4 attempt to solve sets of several hundred word problems. The problem sets were generated during proof attempts by the tool Streambox. Since these problems contain both convertible and non-convertible instances, we could use them to test the performance of NCSolver and Mace4 for the cases that there existed a solution and not.

From these tests we can conclude that the run times of both tools are comparable, usually

no more then 50 milliseconds apart. Nevertheless there were instances for which Mace4 seems to perform considerably less then NCSolver. From these tests we identified the following non-convertibility problem:

Example 8.1.1 Let E be:

$\mathrm{Tl}(f(x,y)) = y$
$\mathrm{M} = f(0,\mathrm{C})$
$\mathrm{C} = f(1,h(C))$
$h(f(0,x)) = f(0,f(1,h(x)))$
$h(f(1,x)) = f(1,f(0,h(x)))$
$\mathrm{Even}(x) = f(\mathrm{Hd}(x),\mathrm{Odd}(\mathrm{Tl}(x)))$
$\mathrm{Odd}(x) = \mathrm{Even}(\mathrm{Tl}(x))$
$\mathrm{Freeze}(h(x)) = \mathrm{Hfrozen}(\mathrm{Freeze}(x))$
$\mathrm{Freeze}(\mathrm{Even}(\mathrm{M})) = \mathrm{Freeze}(\mathrm{M})$
$\mathrm{Freeze}(\mathrm{Tl}(\mathrm{Even}(\mathrm{M}))) = \mathrm{Freeze}(\mathrm{Tl}(\mathrm{M}))$
$\mathrm{Freeze}(\mathrm{Tl}(\mathrm{Tl}(\mathrm{Even}(\mathrm{M})))) = \mathrm{Freeze}(\mathrm{Tl}(\mathrm{Tl}(\mathrm{M})))$

and the goal G be:

$\mathrm{Freeze}(\mathrm{Tl}(\mathrm{Tl}(\mathrm{Tl}(\mathrm{M})))) = \mathrm{Freeze}(\mathrm{Tl}(\mathrm{Tl}(\mathrm{Tl}(\mathrm{Even}(\mathrm{M})))))$

This problem can be solved by NCSolver within a running time of 7 seconds to find a solution with model size 5. However when solved by Mace4, it can take up to 47 seconds to reach the same conclusion.

While this result is an extreme case, we have encountered other problem instances for which Mace4 performs slower, sometimes up to several seconds. Despite investigating these instances we have not been able to identify what causes these differences in runnting times.

In conclusion we can say that NCSolver has a performance comparable to that of Mace4, with the exception of some special cases, for which the difference in performance has yet to be explained.

8.2 Infinite model solving

As the main novelty, NCSolver can also solve non-convertibility problems by trying to find infinite models that satisfy the given problem. Since the infinite model solver uses quasi linear interpretation functions, in order to generate SMT specifications, NCSolver requires the following three parameters:

- The level of the quasi linear interpretation functions.
- The minimum value for which the coefficients are instantiated.
- The maximum value for which the coefficients are instantiated.

For these parameters it holds that a larger level and a bigger range of coefficient values gives a greater chance of solving the given problem. However by increasing these value, the SMT specifications grow in size, and the process of finding a solution becomes slower. So in general

one chooses the values as small as possible, and increase them if necessary. Note that we decided that the levels for the interpretation functions are all equal. While it is certainly possible that not every interpretation interpretation function requires the same level, it is impossible to know beforehand which interpretation requires what level.

Next we shall give a few simple examples of non-convertibility problems that can only be solved by means of infinite models, and what parameters are required to solve them. We first give an example that shows that certain non-convertibility problems require a larger level then 1:

Example 8.2.1 Let E be:

$f(h(g(g(x)))) = g(f(x))$
$f(x) = f(g(f(x)))$

and the goal G be:

$f(f(g(f(x)))) = f(h(g(f(x))))$

This problem can only be solved if the level of the interpretation functions is chosen to be 3. If the coefficients are only allowed to have value 1, NCSolver gives the following solution:

$[f](0) = 1, \quad [f](1) = 1, \quad [f](2) = 0$
$[f](x) = x - 1$
$[h](0) = 10, \quad [h](1) = 11, \quad [h](2) = 2$
$[h](x) = x - 1$
$[g](0) = 2, \quad [g](1) = 0, \quad [g](2) = 3$
$[g](x) = x + 1$

In the next example, the solution of the non-convertibility problem requires that the maximum instantiated value for the coefficients is 2:

Example 8.2.2 Let E be:

$f(f(h(x))) = h(f(x))$
$g(f(x)) = x$

and the goal G be:

$f(g(x)) = x$

To solve this problem we are required to tell NCSolver that the range of coefficient values lies between 1 and 2. If the level is chosen to be 1, we get the following solution:

$[f](0) = 0$
$[f](x) = x + 1$
$[h](0) = 0$
$[h](x) = 2x + 1$
$[g](0) = 0$

$[g](x) = x - 1$

And indeed the solution shows that the interpretation function $[h]$ requires a coefficient of value 2.

The previous examples show that each problem requires the right set of parameters to be solvable. It would seem tempting to just pick a large enough level and range. However we have to take into account that the performance of NCSolver drops as the values of the parameters increase.

The main problem when discussing the performance of NCSolver for infinite models is that to our knowledge there are no other tools available that can solve these kind of non-convertibility problems. If we compare the infinite model solving to that of finite model solving, it is clear that infinite model solving is slower for all problems, which however can be expected for a method that is more powerful.

Chapter 9

Future work

While our approach enables us to solve more non-convertibility problems then was previously possible, there are still many challenges left. It should be obvious that due to the fact that the problem of non-convertibility is an undecidable problem it will not always be possible to give simple models to prove non-convertibility. However there are many other possible types of interpretation functions that could be investigated. Due to the limitations of the SMT solver Yices we did not investigate the use of polynomial functions of a higher degree then one. Neither have we investigated the use of for example the division operator or functions with fractional coefficients.

We introduced the use of quasi linear function to solve non-convertibility problems. However an issue with this approach is that for problems containing n-ary functions, the SMT formulas generated to express them, quickly grow in size. This is due to the fact that for each interpretation function, the level and the function arity determine the required size of $c(v_n, \ldots, v_1)$. Additionally we proved that the level of the composition of quasi linear functions is bigger then the levels of the functions part of the composition. Hence the bigger the terms, the more compositions are nested, the higher the levels will be. Since each of the elements of $c(v_n, \ldots, v_1)$ require individual decision variables, the complexity of the SMT specification can grow quickly. Possible future work can involve investigating a more tight bound on the size of $c(v_n, \ldots, v_1)$.

During our research we significantly reduced the size of the SMT specifications by defining the compositions found in the equations separately, and reusing them for other equations. This shows that there might be additional improvements that could improve the SMT specification. It is for example known that redundance in the SMT formulas can reduce the performance of Yices.

Chapter 10

Conclusions

We have investigated the use of SMT specifications to solve non-convertibility problems. Our approach based on the interpretation method searches for models for which a certain set of equations hold, and the equation to be proven non-convertible does not. We started by using models based on interpretation functions over a finite domain A. To measure the performance of our approach, we compared it to the tool Mace4. It turned out that the performance of our approach is in general comparable to that of Mace4, while for some special cases the performance was significantly better.

However the approach using finite models has its limitations. A very simple problem can be given which can be proven to be unsolvable for any approach based on finite models. This led us to investigate whether infinite models could be used. For this purpose we created a new class of functions called the Quasi linear functions. The quasi linear functions are linear functions over the domain of natural number for which the first k output values are defined separately from the formula of the function. This new class of functions allows us solve non-convertibility problems which can not be solved by means of finite models. In order to increase the power of our models we extended the class of Quasi linear functions to include functions of any arity.

In this paper we showed that the use of SMT solving to solve non-convertibility problems is indeed successful. We created a tool called NCSolver that can fully automatically solve non-convertibility problems for which can be proven not to be solvable by means of finite models. We were able to improve in both performance and power upon what has already been done. Future research can be done in using other types of interpretation functions. It will also be useful to investigate ways to reduce the complexity of the SMT formulas involved, such that our approach can be used on larger problem instances.

Bibliography

[1] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998. 1, 3

[2] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. *Satisfiability Modulo Theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009. 1

[3] Arnim Buch, Thomas Hillenbrand, and Roland Fettig. Waldmeister: High performance equational theorem proving. 1

[4] Ahlem Ben Cherifa and Pierre Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Sci. Comput. Program.*, 9(2):137–159, 1987. 3

[5] Bruno Dutertre and Leonardo de Moura. The yices smt solver. available at `http://yices.csl.sri.com/tool-paper.pdf`. 9

[6] W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/ mccune/prover9/`, 2005–2010. 1, 2

[7] David A. Plaisted. *Equational reasoning and term rewriting systems*, pages 274–364. Oxford University Press, Inc., New York, NY, USA, 1993. 1

[8] Hans Zantema and Joerg Endrullis. Proving Equality of Streams Automatically. In Manfred Schmidt-Schauß, editor, *22nd International Conference on Rewriting Techniques and Applications (RTA'11)*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 393–408, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 1