Termination Analysis for Partial Functions*

Jürgen Brauburger and Jürgen Giesl

FB Informatik, TH Darmstadt, Alexanderstr. 10, 64283 Darmstadt, Germany
E-mail: {brauburger|giesl}@inferenzsysteme.informatik.th-darmstadt.de

Abstract. This paper deals with automated termination analysis for *partial functional programs*, i.e. for functional programs which do not terminate for each input. We present a method to determine their *domains* (resp. non-trivial subsets of their domains) *automatically*. More precisely, for each functional program a *termination predicate* algorithm is synthesized, which only returns true for inputs where the program is terminating. To ease subsequent reasoning about the generated termination predicates we also present a procedure for their simplification.

1 Introduction

Termination of algorithms is a central problem in software development and formal methods for termination analysis are essential for program verification. While most work on the automation of termination proofs has been done in the areas of *term rewriting systems* (for surveys see e.g. [Der87, Ste95]) and of *logic programs* (e.g. [UV88, Plü90, SD94]), in this paper we focus on *functional programs.*

Up to now all methods for automated termination analysis of functional programs (e.g. [BM79, Wal88, Hol91, Wal94b, NN95, Gie95b, Gie95c]) aim to prove that a program terminates for *each* input. However, if the termination proof fails then these methods provide no means to find a (sub-)domain where termination is provable. Therefore these methods cannot be used to analyze the termination behaviour of *partial* functional programs, i.e. of programs which do not terminate for all inputs [BM88].

In this paper we automate Manna's approach for termination analysis of "partial programs" [Man74]: For every algorithm defining a function f there has to be a *termination predicate*[1] θ_{f} which specifies the "admissible input" of f (i.e. evaluation of f must terminate for each input admitted by the termination predicate). But while in [Man74] termination predicates have to be provided by the user, in this paper we present a technique to synthesize them *automatically.*

---

* Technical Report IBN 96/33, Technische Hochschule Darmstadt. This is an extended version of a paper [BG96] which appeared in the *Proceedings of the Third International Static Analysis Symposium*, Aachen, Germany, LNCS 1145, Springer-Verlag, 1996.
This work was supported by the Deutsche Forschungsgemeinschaft under grant no. Wa 652/7-1 as part of the focus program "Deduktion".

[1] Instead of "termination predicates" Manna uses the notion of "input predicates".

In Section 2 we introduce our functional programming language and sketch the basic approach for proving termination of algorithms. Then in Section 3 we show the requirements termination predicates have to satisfy and based on these requirements we present a procedure for the automated synthesis of termination predicates[2] in Section 4. The generated termination predicates can be used both for further automated and interactive program analysis. To ease the handling of these termination predicates we have developed a procedure for their simplification which is introduced in Section 5. Finally, we give a summary of our method (Section 6) and we end up with an appendix which contains a collection of examples to illustrate the power of our method.

2 Termination of Algorithms

In this paper we regard an eager first-order functional language with (free) algebraic data types. To simplify the presentation we restrict ourselves to non-parameterized types and to functions without mutual recursion (see the conclusion for a discussion of possible extensions of our method).

As an example consider the algebraic data type nat for natural numbers. Its objects are built with the *constructors* 0 and succ and we use a *selector* pred as an inverse function to succ (with $\mathsf{pred}(\mathsf{succ}(x)) = x$ and $\mathsf{pred}(0) = 0$, i.e. pred is a total function). To ease readability we often write "1" instead of "succ(0)" etc. For each data type s there must be a pre-defined equality function "=" : $s \times s \to \mathsf{bool}$. Then the following algorithm defines the subtraction function:

$$
\begin{array}{l}
\textit{function}\ \mathsf{minus}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ x = y\ \ \textit{then}\ 0 \\
\qquad\qquad\qquad \textit{else}\ \ \mathsf{succ}(\mathsf{minus}(\mathsf{pred}(x), y)).
\end{array}
$$

In our language, the body q of an algorithm "*function* $\mathsf{f}(x_1 : s_1, \ldots, x_n : s_n) : s \Leftarrow q$" is a term built from the variables $x_1, \ldots, x_n$, constructors, selectors, equality function symbols, function symbols defined by algorithms, and conditionals (where we write "*if* t_1 *then* t_2 *else* t_3" instead of "$\textit{if}(t_1, t_2, t_3)$"). These conditionals are the only functions with non-eager semantics, i.e. when evaluating "*if* t_1 *then* t_2 *else* t_3", the (boolean) term t_1 is evaluated first and depending on the result of its evaluation either t_2 or t_3 is evaluated afterwards.

To prove *termination* of an algorithm one has to show that in each recursive call a given *measure* is decreased. For that purpose a *measure function* $|.|$ is used which maps a tuple of data objects $t_1, \ldots, t_n$ to a natural number $|t_1, \ldots, t_n|$. In the following we often abbreviate tuples $t_1, \ldots t_n$ by t^*.

For example, one might attempt to prove termination of minus with the *size* measure $|.|_\#$, where the size of an object of type nat is the number it represents (i.e. the number of succ's it contains). So we have $|0|_\# = 0$, $|\mathsf{succ}(0)|_\# = 1$ etc. As minus is a binary function, for its termination proof we need a measure function

[2] Strictly speaking, we synthesize *algorithms* which compute termination predicates. For the sake of brevity sometimes we also refer to these algorithms as "termination predicates".

on pairs of data objects. Therefore we extend the size measure function to pairs by measuring a pair by the size of the first object, i.e. $|t_1, t_2|_\# = |t_1|_\#$. Hence, to prove termination of minus we now have to verify the following inequality for all instantiations of x and y where $x \neq y$ holds[3]:

$$|\mathsf{pred}(x), y|_\# < |x, y|_\#. \tag{1}$$

But the algorithm for minus does not terminate for all inputs, i.e. minus is a *partial* function (in fact, $\mathsf{minus}(x, y)$ only terminates if the number x is not smaller than the number y). For instance, the call $\mathsf{minus}(0, 2)$ leads to the recursive call $\mathsf{minus}(\mathsf{pred}(0), 2)$. As $\mathsf{pred}(0)$ is evaluated to 0, this results in calling $\mathsf{minus}(0, 2)$ again. Hence, evaluation of $\mathsf{minus}(0, 2)$ is not terminating. Consequently, our termination proof for minus must fail. For example, (1) is not satisfied if x is 0 and y is 2.

Instead of proving that an algorithm terminates for *all* inputs (*absolute* termination), in the following we are interested in finding subsets of inputs where the algorithms are terminating. Hence, for each algorithm defining a function f we want to generate a *termination predicate* algorithm θ_f where evaluation of θ_f always terminates and if θ_f returns true for some input t^* then evaluation of $\mathsf{f}(t^*)$ terminates, too.

Definition 1. Let $\mathsf{f} : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ be defined by a (possibly non-terminating) algorithm. A *total* function $\theta_\mathsf{f} : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{bool}$ is a **termination predicate** for f iff for all tuples t^* of data objects, $\theta_\mathsf{f}(t^*) = \mathsf{true}$ implies that the evaluation of $\mathsf{f}(t^*)$ is terminating.

Of course the problem of determining the *exact* domains of functions is undecidable. As we want to generate termination predicates automatically we therefore only demand that a termination predicate θ_f represents a *sufficient* criterion for the termination of f's algorithm. So in general, a function f may have an infinite number of termination predicates and false is a termination predicate for each function. But of course our aim is to synthesize weaker termination predicates, i.e. termination predicates which return true as often as possible.

3 Requirements for Termination Predicates

In this section we introduce two requirements that are sufficient for termination predicates, i.e. if a (terminating) algorithm satisfies these requirements then it defines a termination predicate for the function under consideration. A procedure for the automated synthesis of such algorithms will be presented in Section 4.

First, we consider simple partial functions like minus (Section 3.1) and subsequently we will also examine algorithms which call other partial functions (Section 3.2).

[3] We often use "$t \neq r$" as an abbreviation for $\neg(t = r)$, where the boolean function $\neg$ is defined by an (obvious) algorithm.

3.1 Termination Predicates for Simple Partial Functions

We resume our example and generate a termination predicate θ_{minus} such that evaluation of $\mathsf{minus}(x, y)$ terminates if $\theta_{\mathsf{minus}}(x, y)$ is true. Recall that for proving absolute termination one has to show that a certain measure is decreased in each recursive call. But as we illustrated, the algorithm for minus is not always terminating and therefore inequality (1) does not hold for all instantiations of x and y which lead to a recursive call. Hence, the central idea for the construction of a termination predicate θ_{minus} is to let θ_{minus} return true only for those inputs x and y where the measure of x and y is greater than the measure of the corresponding recursive call and to return false for all other inputs. So if evaluation of $\mathsf{minus}(x, y)$ leads to a recursive call (i.e. if $x \neq y$ holds), then $\theta_{\mathsf{minus}}(x, y)$ may only return true if the measure $|\mathsf{pred}(x), y|_{\#}$ is smaller than $|x, y|_{\#}$. This yields the following requirement for a termination predicate θ_{minus}:

$$\theta_{\mathsf{minus}}(x, y) \wedge x \neq y \rightarrow |\mathsf{pred}(x), y|_{\#} < |x, y|_{\#}. \tag{2}$$

For example, the function defined by the following algorithm satisfies (2):

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{minus}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = y\ \ \textit{then}\ \mathsf{true} \\ \qquad\qquad\qquad\quad \textit{else}\ \ |\mathsf{pred}(x), y|_{\#} < |x, y|_{\#}. \end{array}$$

This algorithm for θ_{minus} uses the same case analysis as minus. Since minus terminates in its non-recursive case (i.e. if $x = y$), the corresponding result of θ_{minus} is true. For the recursive case (if $x \neq y$), θ_{minus} returns true iff $|\mathsf{pred}(x), y)|_{\#} < |x, y|_{\#}$ is true. We assume that each measure function $|.|$ is defined by a (terminating) algorithm. Hence, in the result of the second case θ_{minus} calls the algorithm for the computation of the size measure $|.|_{\#}$ and it also calls a (terminating) algorithm to compute the less-than relation "$<$" on natural numbers.

So in general, given an algorithm for f we demand the following requirement for termination predicates θ_{f} (where $|.|$ is an arbitrary measure function):

If evaluation of $\mathsf{f}(t^*)$ leads to a recursive call $\mathsf{f}(r^*)$, then $\theta_{\mathsf{f}}(t^*)$ may only return true if $|r^*| < |t^*|$ holds. (Req1)

However, (Req1) is not a *sufficient* requirement for termination predicates. For instance, the function θ_{minus} defined above is not a termination predicate for minus although it satisfies requirement (Req1). The reason is that $\theta_{\mathsf{minus}}(1, 2)$ returns true (as $|\mathsf{pred}(1), 2|_{\#} < |1, 2|_{\#}$ holds). But evaluation of $\mathsf{minus}(1, 2)$ is not terminating because its evaluation leads to the (non-terminating) recursive call $\mathsf{minus}(0, 2)$.

This non-termination is not recognized by θ_{minus} because $\theta_{\mathsf{minus}}(1, 2)$ only checks if the arguments $(0, 2)$ of the *next* recursive call of minus are smaller than the input $(1, 2)$. But it is not guaranteed that *subsequent* recursive calls are also measure decreasing. For example, the next recursive call with the arguments $(0, 2)$ will lead to a subsequent recursive call of minus with the same arguments, i.e. in the subsequent recursive call the measure of the arguments remains the same. For that reason $\theta_{\mathsf{minus}}(1,2)$ evaluates to true, but application of θ_{minus} to the arguments $(0,2)$ of the following recursive call yields false.

Therefore in addition to (Req1) we must demand that a termination predicate θ_{f} remains valid for each recursive call in f's algorithm. This ensures that subsequent recursive calls are also measure decreasing:

$$\begin{array}{l}\text{If evaluation of } \mathsf{f}(t^*) \text{ leads to a recursive call } \mathsf{f}(r^*),\\ \text{then } \theta_{\mathsf{f}}(t^*) \text{ may only return true if } \theta_{\mathsf{f}}(r^*) \text{ is also true.}\end{array} \tag{3}$$

In our example, to satisfy the requirements (Req1) and (3) we modify the result of θ_{minus}'s second case by demanding that θ_{minus} also holds for the following recursive call of minus:

$$\begin{array}{l}\textit{function}\ \theta_{\mathsf{minus}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow\\ \quad \textit{if}\ \ x = y \ \ \textit{then}\ \mathsf{true}\\ \qquad\qquad\qquad\ \ \textit{else}\ \ |\mathsf{pred}(x), y|_{\#} < |x, y|_{\#} \ \wedge\ \theta_{\mathsf{minus}}(\mathsf{pred}(x), y).\end{array}$$

In this algorithm we use the boolean function symbol $\wedge$ to ease readability, where $\varphi_1 \wedge \varphi_2$ abbreviates "*if* φ_1 *then* φ_2 *else* false". Hence, the semantics of the function $\wedge$ are *not* eager. So terms in a conjunction are evaluated from left to right, i.e. given a conjunction $\varphi_1 \wedge \varphi_2$ of boolean terms (which we also refer to as "formulas"), φ_1 is evaluated first. If the value of φ_1 is false, then false is returned, otherwise φ_2 is evaluated and its value is returned. Note that we need a *lazy* conjunction function $\wedge$ to ensure termination of θ_{minus}. It guarantees that evaluation of $\theta_{\mathsf{minus}}(x, y)$ can only lead to a recursive call $\theta_{\mathsf{minus}}(\mathsf{pred}(x), y)$ if the measure of the recursive arguments $|\mathsf{pred}(x), y|_{\#}$ is smaller than the measure of the inputs $|x, y|_{\#}$.

The above algorithm really defines a termination predicate for minus, i.e. θ_{minus} is a total function and the truth of θ_{minus} is sufficient for the termination of minus. This algorithm for θ_{minus} was constructed in order to obtain an algorithm satisfying the requirements (Req1) and (3). In Section 4 we will show that this construction can easily be automated. A closer look at θ_{minus} reveals that we have synthesized an algorithm which computes the usual greater-equal relation "$\geq$" on natural numbers. As $\mathsf{minus}(x, y)$ is *only* terminating if x is greater than or equal to y, in this example we have even generated the weakest possible termination predicate, i.e. θ_{minus} returns true not only for a subset but for *all* elements of the domain of minus.

3.2 Algorithms Calling Other Partial Functions

In general (Req1) and (3) are not sufficient criteria for termination predicates. These requirements can only be used for algorithms like minus which (apart from recursive calls) only call other *total* functions (like =, succ, and pred).

In this section we will examine algorithms which call other *partial* functions. As an example consider the algorithm for $\mathsf{list\_minus}(l, y)$ which subtracts the number y from all elements of a list l. Objects of the data type list are built with the constructors empty and add, where $\mathsf{add}(x, k)$ represents the insertion of the number x into the list k. We also use the selectors head and tail, where head returns the first element of a list and tail returns a list without its first element (i.e. $\mathsf{head}(\mathsf{add}(x,k)) = x$, $\mathsf{head}(\mathsf{empty}) = 0$, $\mathsf{tail}(\mathsf{add}(x,k)) = k$, $\mathsf{tail}(\mathsf{empty}) = \mathsf{empty}$).

$$
\begin{array}{l}
\textit{function } \mathsf{list\_minus}(l : \mathsf{list}, y : \mathsf{nat}) : \mathsf{list} \Leftarrow \\
\quad \textit{if } \ l = \mathsf{empty} \quad \textit{then } \mathsf{empty} \\
\qquad\qquad\qquad\qquad \textit{else } \ \mathsf{add}(\mathsf{minus}(\mathsf{head}(l), y), \mathsf{list\_minus}(\mathsf{tail}(l), y)).
\end{array}
$$

We construct the following algorithm for $\theta_{\mathsf{list\_minus}}$ by measuring pairs $|l, y|_{\#}$ by the *size* of the first object $|l|_{\#}$ again, where the size of a list is its length.

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{list\_minus}}(l : \mathsf{list}, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \ l = \mathsf{empty} \quad \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad \textit{else } \ |\mathsf{tail}(l), y|_{\#} < |l, y|_{\#} \wedge \theta_{\mathsf{list\_minus}}(\mathsf{tail}(l), y).
\end{array}
$$

But although this algorithm defines a function which satisfies (Req1) and (3), it is not a termination predicate for list_minus. The reason is that $\theta_{\mathsf{list\_minus}}(\mathsf{add}(0, \mathsf{empty}), 2)$ evaluates to true because the size of the empty list is smaller than the size of $\mathsf{add}(0, \mathsf{empty})$. But evaluation of $\mathsf{list\_minus}(\mathsf{add}(0, \mathsf{empty}), 2)$ is not terminating as it leads to the (non-terminating) evaluation of $\mathsf{minus}(0, 2)$.

The problem is that $\theta_{\mathsf{list\_minus}}$ only checks if *recursive* calls of list_minus are measure decreasing but it does not guarantee the termination of *other* algorithms called. Therefore we have to demand that $\theta_{\mathsf{list\_minus}}$ ensures termination of the subsequent call of minus, i.e. in the second case $\theta_{\mathsf{list\_minus}}(l, y)$ must imply $\theta_{\mathsf{minus}}(\mathsf{head}(l), y)$.

So we replace (3) by a requirement that guarantees the truth of $\theta_{\mathsf{g}}(r^*)$ for *all* function calls $\mathsf{g}(r^*)$ in f's algorithm (i.e. also for functions g different from f):

$$
\begin{array}{l}
\text{If evaluation of } \mathsf{f}(t^*) \text{ leads to a function call } \mathsf{g}(r^*), \\
\text{then } \theta_{\mathsf{f}}(t^*) \text{ may only return true if } \theta_{\mathsf{g}}(r^*) \text{ is also true.}
\end{array}
\qquad \text{(Req2)}
$$

Note that (Req2) must also be demanded for non-recursive cases. The function $\theta_{\mathsf{list\_minus}}$ defined by the following algorithm satisfies (Req1) and the extended requirement (Req2):

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{list\_minus}}(l : \mathsf{list}, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \ l = \mathsf{empty} \quad \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad \textit{else } \ \theta_{\mathsf{minus}}(\mathsf{head}(l), y) \wedge |\mathsf{tail}(l), y|_{\#} < |l, y|_{\#} \wedge \theta_{\mathsf{list\_minus}}(\mathsf{tail}(l), y).
\end{array}
$$

The above algorithm in fact defines a termination predicate for list_minus. Analyzing the algorithm one notices that $\theta_{\mathsf{list\_minus}}(l, y)$ returns true iff all elements of l are greater than or equal to y. As evaluation of $\mathsf{list\_minus}(l, y)$ only terminates for such inputs, we have synthesized the weakest possible termination predicate again.

Note that algorithms may also call partial functions in their *conditions*. For example consider the algorithm for half which calls minus in its conditions:

$$
\begin{array}{l}
\textit{function}\ \mathsf{half}(x : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ \mathsf{minus}(x, 2) = 0 \quad \textit{then}\ 1 \\
\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{succ}(\mathsf{half}(\mathsf{minus}(x, 2))).
\end{array}
$$

This algorithm does not terminate for the inputs 0 or 1, since in the conditions the term $\mathsf{minus}(x, 2)$ must be evaluated. Therefore due to (Req2), θ_{half} must ensure that all calls of the partial function minus in the conditions are terminating, i.e. $\theta_{\mathsf{half}}(x)$ must imply $\theta_{\mathsf{minus}}(x, 2)$. The following algorithm for θ_{half} satisfies both requirements (Req1) and (Req2):

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{half}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{minus}}(x, 2) \ \ \wedge \ \ (\,\textit{if}\ \ \mathsf{minus}(x, 2) = 0 \\
\qquad\qquad\qquad\qquad\quad \textit{then}\ \ \mathsf{true} \\
\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \ \theta_{\mathsf{minus}}(x, 2) \wedge |\mathsf{minus}(x, 2)|_{\#} < |x|_{\#} \wedge \theta_{\mathsf{half}}(\mathsf{minus}(x, 2))\,).
\end{array}
$$

The above algorithm first checks if the call of the algorithm minus in the conditions of half is terminating. If the corresponding termination predicate $\theta_{\mathsf{minus}}(x, 2)$ is false, then θ_{half} also returns false. Otherwise, evaluation of θ_{half} continues as usual.

This algorithm really defines a termination predicate for half. Analysis of θ_{half} reveals that we have synthesized the "even"-algorithm (for numbers greater than 0) which again is the weakest possible termination predicate for half.

The following lemma states that the two requirements we have derived are in fact sufficient for termination predicates.

Lemma 2. *If a total function θ_{f} satisfies the requirements (Req1) and (Req2) then θ_{f} is a termination predicate for* f.

Proof. Suppose that there exist data objects t^* such that $\theta_{\mathsf{f}}(t^*)$ returns true but evaluation of $\mathsf{f}(t^*)$ does not terminate. Then let t^* be the smallest such data objects, i.e. for all objects r^* with a measure $|r^*|$ smaller than $|t^*|$ the truth of $\theta_{\mathsf{f}}(r^*)$ implies termination of $\mathsf{f}(r^*)$.

As we have excluded mutual recursion we may assume that for all other functions g (which are called by f) the predicate θ_{g} really is a termination predicate. Hence, requirement (Req2) ensures that evaluation of $f(t^*)$ can only lead to terminating calls of *other* functions g. Therefore the non-termination of $f(t^*)$ cannot be caused by another function g.

So evaluation of $\mathsf{f}(t^*)$ must lead to recursive calls $\mathsf{f}(r^*)$. But because of requirement (Req1), r^* has a smaller measure than t^*. Hence, due to the minimality of t^*, $\mathsf{f}(r^*)$ must be terminating (as (Req2) ensures that $\theta_{\mathsf{f}}(r^*)$ also returns true). So the recursive calls of f cannot cause non-termination either. Therefore evaluation of $\mathsf{f}(t^*)$ must also be terminating. □

4 Automated Generation of Termination Predicates

In this section we show how algorithms defining termination predicates can be synthesized automatically. Given a functional program f, we present a technique to generate a (terminating) algorithm for θ_{f} satisfying the requirements (Req1) and (Req2). Then due to Lemma 2 this algorithm defines a termination predicate for f.

Requirement (Req2) demands that θ_{f} may only return true if evaluation of all terms in the conditions and results of f is terminating. Therefore we extend the idea of termination predicates from *algorithms* to arbitrary *terms*.

Hence, for each term t we construct a boolean term $\Theta(t)$ (a *termination formula* for t) such that evaluation of $\Theta(t)$ is terminating and $\Theta(t) = \mathsf{true}$ implies that evaluation of t is also terminating[4]. For example, a termination formula for $\mathsf{half}(\mathsf{minus}(x,2))$ is $\theta_{\mathsf{minus}}(x,2) \wedge \theta_{\mathsf{half}}(\mathsf{minus}(x,2))$, because due to the eager nature of our functional language in this term minus is evaluated before evaluating half. So termination formulas have to guarantee that a subterm $\mathsf{g}(r^*)$ is only evaluated if $\theta_{\mathsf{g}}(r^*)$ holds. In general, *termination formulas* are constructed by the following rules:

$$\begin{array}{lll} \Theta(x) & :\equiv \mathsf{true}, & \text{for variables } x, \quad \text{(i)} \\ \Theta(\mathsf{g}(r_1,\ldots,r_n)) & :\equiv \Theta(r_1) \wedge \ldots \wedge \Theta(r_n) \wedge \theta_{\mathsf{g}}(r_1,\ldots,r_n), & \text{for functions } \mathsf{g}, \quad \text{(ii)} \\ \Theta(\mathit{if}\ r_1\ \mathit{then}\ r_2\ \mathit{else}\ r_3) & :\equiv \Theta(r_1) \wedge \mathit{if}\ r_1\ \mathit{then}\ \Theta(r_2)\ \mathit{else}\ \Theta(r_3). & \text{(iii)} \end{array}$$

Note that in rule (ii), if g is a constructor, a selector, or an equality function, then we have $\theta_{\mathsf{g}}(x^*) = \mathsf{true}$, because those functions are total.

To satisfy requirement (Req2) θ_{f} must ensure that evaluation of all terms in the body of an algorithm f terminates. So if f is defined by the algorithm "*function* $\mathsf{f}(x_1 : s_1, \ldots, x_n : s_n) : s \Leftarrow q$", then θ_{f} has to check whether the termination formula $\Theta(q)$ of f's body is true.

But the body of f can also contain recursive calls $\mathsf{f}(r^*)$. To satisfy requirement (Req1) we must additionally ensure that the measure $|r^*|$ of recursive calls is smaller than the measure of the inputs $|x^*|$. Therefore for recursive calls $\mathsf{f}(r^*)$ we have to change the definition of *termination formulas* as follows:

$$\Theta(\mathsf{f}(r_1,\ldots,r_n)) :\equiv \Theta(r_1) \wedge \ldots \wedge \Theta(r_n) \wedge |r_1,\ldots,r_n| < |x_1,\ldots,x_n| \wedge \theta_{\mathsf{f}}(r_1,\ldots,r_n) \quad \text{(iv)}$$

In this way we obtain the following procedure for the generation of termination predicates.

Theorem 3. *Given an algorithm "function* $\mathsf{f}(x_1 : s_1, \ldots, x_n : s_n) : s \Leftarrow q$*", we define the algorithm "function* $\theta_{\mathsf{f}}(x_1 : s_1, \ldots, x_n : s_n) : \mathsf{bool} \Leftarrow \Theta(q)$*", where the termination formula* $\Theta(q)$ *is constructed by the rules (i) - (iv). Then this algorithm defines a termination predicate* θ_{f} *for* f *(i.e. this algorithm is terminating and if* $\theta_{\mathsf{f}}(t^*)$ *returns* true*, then evaluation of* $\mathsf{f}(t^*)$ *is also terminating).*

Proof. By the definition of termination formulas, algorithms generated according to Theorem 3 are *terminating*, because evaluation of $\theta_{\mathsf{f}}(t^*)$ can only lead to a

[4] More precisely, this implication holds for each substitution σ of t's variables by data objects: For all such σ, evaluation of $\sigma(\Theta(t))$ is terminating and $\sigma(\Theta(t)) = \mathsf{true}$ implies that the evaluation of $\sigma(t)$ is also terminating.

recursive call $\theta_{\mathsf{f}}(r^*)$ if the measure $|r^*|$ is smaller than $|t^*|$ and because calls of *other* functions $\mathsf{g}(s^*)$ can only be evaluated if $\theta_{\mathsf{g}}(s^*)$ holds.

Moreover, by construction the generated algorithm defines a function θ_{f} which satisfies the requirements (Req1) and (Req2) we presented in Section 3. Due to Lemma 2 this implies that θ_{f} must be a termination predicate for f, i.e. it is total and it is sufficient for termination of f. □

The construction of algorithms for termination predicates according to Theorem 3 can be directly automated. So by this theorem we have developed a procedure for the automated generation of termination predicates. For instance, the termination predicate algorithms for minus, list_minus, and half in the last section were built according to Theorem 3 (where for the sake of brevity we omitted termination predicates for *total* functions because such predicates always return true). As demonstrated, the generated termination predicates often are as weak as possible, i.e. they often describe the *whole* domain of the partial function under consideration (instead of just a sub-domain).

5 Simplification of Termination Predicates

In the last section we presented a method for the automated generation of algorithms which define termination predicates. But sometimes the synthesized algorithms are unnecessarily complex. To ease subsequent reasoning about termination predicates in the following sections we introduce a procedure to *simplify* the generated termination predicate algorithms which consists of four steps.

5.1 Application of Induction Lemmata

First, the well-known induction lemma method by *R. S. Boyer* and *J S. Moore* [BM79] is used to eliminate (some of) the inequalities $|r^*| < |x^*|$ (which ensure that recursive calls are measure decreasing) from the termination predicate algorithms. Elimination of these inequalities simplifies the algorithms considerably and often enables the execution of subsequent simplification steps.

An *induction lemma* points out that under a certain hypothesis δ some operation drives some measure down, i.e. induction lemmata have the form

$$\delta \rightarrow |r^*| < |x^*|.$$

In the system of Boyer and Moore induction lemmata have to be provided by the *user*. However, *C. Walther* presented a method to generate a certain class of induction lemmata for the *size* measure function $|.|_\#$ *automatically* [Wal94b] and we recently generalized his approach towards measure functions based on arbitrary *polynomial norms* [Gie95b]. For instance, the induction lemma needed in the following example can be synthesized by Walther's and our method.

While Boyer and Moore use induction lemmata for absolute termination proofs, we will now illustrate their use for the simplification of termination predicate algorithms. As an example consider the following algorithm:

$$
\begin{array}{l}
\textit{function } \mathsf{quotient}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if} \;\; x < y \;\; \textit{then } 0 \\
\qquad\qquad\quad \textit{else} \;\; \mathsf{succ}(\mathsf{quotient}(\mathsf{minus}(x, y), y)).
\end{array}
$$

Using the procedure of Theorem 3 the following termination predicate algorithm is generated. In this algorithm we again neglect the call of the termination predicate $\theta_<$ as "$<$" is defined by an (absolutely) terminating algorithm and therefore $\theta_<$ always returns true.

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{quotient}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if} \;\; x < y \;\; \textit{then } \mathsf{true} \\
\qquad\qquad\quad \textit{else} \;\; \theta_{\mathsf{minus}}(x, y) \wedge |\mathsf{minus}(x, y), y|_\# < |x, y|_\# \wedge \theta_{\mathsf{quotient}}(\mathsf{minus}(x, y), y).
\end{array}
$$

We know that in the result of $\theta_{\mathsf{quotient}}$ the term $\mathsf{minus}(x, y)$ will only be evaluated if this evaluation is terminating, i.e. if $\theta_{\mathsf{minus}}(x, y)$ holds. So in order to eliminate the inequality occurring in the result of $\theta_{\mathsf{quotient}}$'s second case, we look for an induction lemma which states that provided minus is terminating the measure of $|\mathsf{minus}(x, y), y|_\#$ is smaller than $|x, y|_\#$ under some hypothesis δ. Hence, we search for an induction lemma of the form

$$\theta_{\mathsf{minus}}(x, y) \wedge \delta \rightarrow |\mathsf{minus}(x, y), y|_\# < |x, y|_\#.$$

For instance, we can use the following induction lemma which states that (provided $\mathsf{minus}(x, y)$ terminates) the result of $\mathsf{minus}(x, y)$ is smaller than its first argument x, if both x and y are not 0:

$$\theta_{\mathsf{minus}}(x, y) \wedge x \neq 0 \wedge y \neq 0 \rightarrow |\mathsf{minus}(x, y), y|_\# < |x, y|_\#.$$

As in the result of $\theta_{\mathsf{quotient}}$ the truth of $\theta_{\mathsf{minus}}(x, y)$ is guaranteed before evaluating the inequality $|\mathsf{minus}(x, y), y|_\# < |x, y|_\#$ we can now replace this inequality by $x \neq 0 \wedge y \neq 0$ which yields the following simplified algorithm:

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{quotient}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if} \;\; x < y \;\; \textit{then } \mathsf{true} \\
\qquad\qquad\quad \textit{else} \;\; \theta_{\mathsf{minus}}(x, y) \wedge x \neq 0 \wedge y \neq 0 \wedge \theta_{\mathsf{quotient}}(\mathsf{minus}(x, y), y).
\end{array}
$$

So in general, if the body of an algorithm contains an inequality $|r^*| < |x^*|$ which will only be evaluated under the condition ψ, then our simplification procedure looks for an induction lemma of the form

$$\psi \wedge \delta \rightarrow |r^*| < |x^*|.$$

If such an induction lemma is known (or can be synthesized) then the inequality $|r^*| < |x^*|$ is replaced by δ.

5.2 Subsumption Elimination

In the next simplification step *redundant terms* are *eliminated* from the termination predicate algorithms. Recall that θ_{minus} computes the greater-equal relation "$\geq$" on natural numbers. Hence the condition of $\theta_{\mathsf{quotient}}$'s second case implies the truth of $\theta_{\mathsf{minus}}(x, y)$, i.e. we can verify

$$x \not< y \ \rightarrow \ \theta_{\mathsf{minus}}(x, y). \tag{4}$$

For that reason the *subsumed term* $\theta_{\mathsf{minus}}(x, y)$ may be eliminated from the second case of $\theta_{\mathsf{quotient}}$ which yields

$$\begin{array}{lll} \textit{if} \ \ x < y & \textit{then} \ \mathsf{true} \\ & \textit{else} \ \ x \neq 0 \wedge y \neq 0 \wedge \theta_{\mathsf{quotient}}(\mathsf{minus}(x, y), y). \end{array}$$

Note that evaluation of the terms $x \neq 0$ and $y \neq 0$ is always terminating (i.e. their termination formulas $\Theta(x \neq 0)$ and $\Theta(y \neq 0)$ are both true). Hence, the order of the terms $x \neq 0$ and $y \neq 0$ can be changed without affecting the semantics of $\theta_{\mathsf{quotient}}$. Then in the result of $\theta_{\mathsf{quotient}}$'s second case the term $x \neq 0$ will only be evaluated under the condition $x \not< y \wedge y \neq 0$. But this condition again implies the truth of $x \neq 0$, i.e. we can easily verify

$$x \not< y \wedge y \neq 0 \ \rightarrow \ x \neq 0. \tag{5}$$

Hence, the *subsumed term* $x \neq 0$ can also be eliminated which results in the following algorithm for $\theta_{\mathsf{quotient}}$:

$$\begin{array}{l} \textit{function} \ \theta_{\mathsf{quotient}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if} \ \ x < y \ \ \textit{then} \ \mathsf{true} \\ \quad \qquad\qquad\quad \textit{else} \ \ y \neq 0 \wedge \theta_{\mathsf{quotient}}(\mathsf{minus}(x, y), y). \end{array}$$

According to [Wal94b] we call formulas like (4) and (5) *subsumption formulas.* So in general, if a term ψ_2 will only be evaluated under the condition ψ_1 and if the subsumption formula $\psi_1 \rightarrow \psi_2$ can be verified, then our simplification procedure replaces the term ψ_2 by true. (Subsequently of course, in a conjunction the term true may be eliminated.)

For the automated verification of subsumption formulas an *induction theorem proving system* is used (e.g. one of those described in [BM79, Bi$^+$86, Bu$^+$90, Wal94a]). For instance, the subsumption formula (4) can be verified by an easy induction proof and subsumption formula (5) can already be proved by case analysis and propositional reasoning only.

5.3 Recursion Elimination

To apply the following simplification step recall that $\varphi_1 \wedge \varphi_2$ is an abbreviation for "*if* φ_1 *then* φ_2 *else* false". Hence, the algorithm for $\theta_{\mathsf{quotient}}$ in fact reads as follows:

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{quotient}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x < y\ \ \textit{then}\ \ \mathsf{true} \\
\qquad\qquad\quad \textit{else}\ \ (\ \textit{if}\ \ y \neq 0\ \ \textit{then}\ \ \theta_{\mathsf{quotient}}(\mathsf{minus}(x, y), y) \\
\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{false}\ ).
\end{array}
$$

So this algorithm has three *cases*, where the first case has the *result* true which is only evaluated under the *condition* $x < y$, the second case has the result $\theta_{\mathsf{quotient}}(\mathsf{minus}(x, y), y)$ and the corresponding condition $x \not< y \wedge y \neq 0$, and the third case has the result false and the condition $x \not< y \wedge y = 0$.

Now we *eliminate* the *recursive call* of $\theta_{\mathsf{quotient}}$ according to the *recursion elimination* technique of Walther [Wal94b]. If we can verify that evaluation of a recursive call $\theta_{\mathsf{f}}(r^*)$ always yields the same result (i.e. it always yields true or it always yields false) then we can replace the recursive call $\theta_{\mathsf{f}}(r^*)$ by this result. In this way it is possible to replace the recursive call of $\theta_{\mathsf{quotient}}$ by the value true. The reason is that each recursive call $\theta_{\mathsf{quotient}}(\mathsf{minus}(x, y), y)$ evaluates to true.

More precisely, the parameters $(\mathsf{minus}(x, y), y)$ of the recursive call either satisfy the condition of $\theta_{\mathsf{quotient}}$'s first case (i.e. $\mathsf{minus}(x, y) < y$) or they satisfy the condition of $\theta_{\mathsf{quotient}}$'s second case (i.e. $\mathsf{minus}(x, y) \not< y \wedge y \neq 0$). This property is expressed by the following formula:

$$x \not< y \wedge y \neq 0 \ \rightarrow\ \mathsf{minus}(x, y) < y \ \vee\ (\mathsf{minus}(x, y) \not< y \ \wedge\ y \neq 0). \tag{6}$$

As the arguments of recursive calls always satisfy the condition of the first (non-recursive) or the second (recursive) case, due to the termination of $\theta_{\mathsf{quotient}}$ after a finite number of recursive calls $\theta_{\mathsf{quotient}}$ will be called with arguments that satisfy the condition of the first non-recursive case. Hence, the result of the evaluation is true. Therefore the recursive call of $\theta_{\mathsf{quotient}}$ can in fact be replaced by true which yields the following non-recursive version of $\theta_{\mathsf{quotient}}$:

$$
\begin{array}{lcl}
\textit{function}\ \theta_{\mathsf{quotient}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow & \text{resp.} & \textit{function}\ \theta_{\mathsf{quotient}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ x < y\ \textit{then}\ \mathsf{true} & & \quad \textit{if}\ x < y\ \textit{then}\ \mathsf{true} \\
\qquad\quad \textit{else}\ \ (\ \textit{if}\ y \neq 0\ \textit{then}\ \mathsf{true} & & \qquad\quad \textit{else}\ \ y \neq 0. \\
\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{false}\ ) & &
\end{array}
$$

In general, let R be a set of recursive θ_{f}-cases with *results* of the form $\theta_{\mathsf{f}}(r^*)$ and let b be a boolean value (either true or false). Our simplification procedure replaces the recursive calls in the R-cases by the boolean value b, if for each case in R evaluation of the result $\theta_{\mathsf{f}}(r^*)$ either leads to a non-recursive case with the result b or to a recursive case from R.

Let Ψ be the set of all *conditions* from non-recursive cases with the result b and of all conditions from R-cases. Then one has to show that the arguments r^* satisfy one of the conditions $\varphi \in \Psi$, i.e. $\varphi[x^*/r^*]$ must be valid (where $[x^*/r^*]$ denotes the substitution of the formal parameters x^* by the terms r^*). Hence, for each case in R with the *condition* ψ the following *recursion elimination formula* has to be verified:

$$\psi \rightarrow \bigvee_{\varphi \in \Psi} \varphi[x^*/r^*].$$

Again, for the automated verification of such formulas an (induction) theorem prover is used. For instance, formula (6) can already be verified by propositional reasoning only.

5.4 Case Elimination

In the last simplification step one tries to replace conditionals by their results. More precisely, regard a conditional of the form "*if* φ_1 *then* true *else* φ_2" which will only be evaluated under a condition ψ. Now the simplification procedure tries to replace this conditional by the result φ_2. For that purpose the procedure has to check whether φ_2 also holds in the *then*-case of the conditional, i.e. it tries to verify the *case elimination formula*

$$\psi \wedge \varphi_1 \;\rightarrow\; \varphi_2.$$

If this implication can be proved (and if the condition $\neg\varphi_1$ is not necessary to ensure termination of φ_2's evaluation, i.e. if $\psi \rightarrow \Theta(\varphi_2)$), then the conditional is replaced by φ_2. Of course, conditionals of the form "*if* φ_1 *then* φ_2 *else* true" can be simplified in a similar way.

In our example, the case elimination formula $x < y \;\rightarrow\; y \neq 0$ can be verified. Moreover, as evaluation of $y \neq 0$ is always terminating (i.e. $\Theta(y \neq 0)$ is true), the condition $x \not< y$ is not necessary to ensure termination of that evaluation. Therefore the conditional in the body of $\theta_{\mathsf{quotient}}$'s algorithm is now replaced by $y \neq 0$. In this way we obtain the final version of $\theta_{\mathsf{quotient}}$:

$$\textit{function}\ \theta_{\mathsf{quotient}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad y \neq 0.$$

Using the above techniques this simple algorithm for $\theta_{\mathsf{quotient}}$ has been constructed which states that evaluation of $\mathsf{quotient}(x, y)$ terminates if y is not 0. This example demonstrates that our simplification procedure eases further automated reasoning about termination predicates significantly and it also enhances the readability of the termination predicate algorithms.

Summing up, the procedure for simplification of termination predicate algorithms works as follows: First, induction lemmata are used to *replace inequalities* by simpler formulas. Then the procedure eliminates *subsumed terms* and *recursive calls.* Finally, *cases* are *eliminated* by replacing conditionals by their results if possible.

This simplification procedure for termination predicates works *automatically.* It is based on a method for the synthesis of induction lemmata [Wal94b, Gie95b] and it uses an induction theorem prover to verify the subsumption, recursion elimination, and case elimination formulas (which often is a simple task).

6 Conclusion

We have presented a method to determine the domains (resp. non-trivial subdomains) of partial functions automatically. For that purpose we have *automated* the approach for termination analysis suggested by Manna [Man74]. Our analysis uses *termination predicates* which represent conditions that are sufficient for the termination of the algorithm under consideration. Based on sufficient requirements for termination predicates we have developed a procedure for the automated synthesis of termination predicate algorithms. Subsequently we introduced a procedure for the simplification of these generated termination predicate algorithms which also works automatically.

The presented approach can be used for polymorphic types, too, and an extension to mutual recursion is possible in the same way as suggested in [Gie96] for absolute termination proofs. Termination analysis can also be extended to higher-order functions by inspecting the decrease of their first-order arguments, cf. [NN95]. To determine non-trivial subdomains of higher-order functions which are not always terminating, in general one does not only need a termination predicate for each function f but one also has to generate termination predicates for the (higher-order) *results* of each function.

Our method proved successful on numerous examples (see Table 1 for some examples to illustrate its power). For each function f in this table the corresponding termination predicate θ_{f} could be synthesized automatically. Moreover, for all these examples the synthesized termination predicate is not only sufficient for termination, but it even describes the *exact* domain of the functions.

These examples demonstrate that the procedure of Theorem 3 is able to synthesize sophisticated termination predicate algorithms (e.g. for a quotient algorithm it synthesizes the termination predicate "divides", for a logarithm algorithm it synthesizes a termination predicate which checks if one number is a power of another number, for an algorithm which deletes an element from a list a termination predicate for list membership is synthesized etc.). By subsequent application of our simplification procedure one usually obtains very simple formulations of the synthesized termination predicate algorithms.

Termination of those algorithms marked with * can be proved by methods for absolute termination proofs, too. But the termination behaviour of all other algorithms in Table 1 could not be analyzed with any other automatic method. Although those functions without * which have the termination predicate true are also total, their totality cannot be verified by the existing methods for absolute termination proofs. The reason is that their algorithms call other non-terminating algorithms. A detailed description of our experiments can be found in the appendix.

The presented procedure for the generation of termination predicates works for any given measure function $|.|$. Therefore the procedure can also be combined with methods for the *automated* generation of suitable measure functions (e.g. the one we presented in [Gie95a, Gie95c]). For example, by using the measures suggested by this method, for all[5] 82 algorithms from the database of [BM79] our procedure synthesizes termination predicates which always return true (i.e. in this way (absolute) termination of all these algorithms is proved automatically).

Furthermore, with our approach it is also possible to perform termination

[5] As mentioned in [Wal94b] one algorithm (greatest.factor) must be slightly modified.

analysis for *imperative programs*: When translating an imperative program into a functional one, usually each *while*-loop is transformed into a partial function, cf. [Hen80]. Now the termination predicates for these partial "loop functions" can be used to prove termination of the whole imperative program.

Acknowledgements. We would like to thank Christoph Walther and the referees for helpful comments.

No	**Function** f	θ_f
1	minus(x, y)	$x \geq y$
2	half1(x)	even(x)
3	half2(x)	even$(x) \wedge x \neq 0$
4	half3*(x)	true
5	double*(x)	true
6	even*(x)	true
7	plus*(x, y)	true
8	times(x, y)	true
9	exp(x, y)	true
10	lt*(x, y)	true
11	quotient1(x, y)	$y \neq 0$
12	mod(x, y)	$y \neq 0$
13	quotient2(x, y)	$y \mid x$
14	gcd(x, y)	$x = 0 \wedge y = 0 \vee x \neq 0 \wedge y \neq 0$
15	lcm(x, y)	$x \neq 0 \wedge y \neq 0$
16	dual_log1(x)	$x \neq 0$
17	dual_log2(x)	$x = 2^n$
18	log1(x, y)	$x = 1 \vee x \neq 0 \wedge y \neq 0 \wedge y \neq 1$
19	log2(x, y)	$x = 1 \vee x = y^n \wedge x \neq 0 \wedge y \neq 1$
20	list_minus(l, y)	$\bigwedge_i l_i \geq y$
21	last(l)	$l \neq$ empty
22	but_last(l)	$l \neq$ empty
23	reverse(l)	true
24	minimum*(x, y)	true
25	list_min(l)	$l \neq$ empty
26	length*(l)	true
27	last_x(l, x)	length$(l) \geq x$
28	index(x, l)	$x = 0 \vee$ member(x, l)
29	delete(x, l)	$x = 0 \vee$ member(x, l)
30	sum_lists(l, k)	length$(l) =$ length(k)
31	nat_to_bin(x, y)	$y = 2^n$
32	bin_vec(x)	$x \neq 0$

Table 1. Termination predicates synthesized by our method.

A Examples

This appendix contains 32 examples to illustrate the power of our method (cf. Table 1). Algorithms marked with * are (absolutely) terminating and only call terminating algorithms (they are required as auxiliary algorithms for the other examples). Hence their termination can also be proved with known techniques for absolute termination proofs. For all other algorithms in this appendix an automated termination analysis is not possible with methods for absolute termination proofs. Note that in all following examples the termination predicates generated by our method are the weakest possible ones (i.e. they return true *iff* the algorithm under consideration terminates).

For each algorithm we first describe its intended semantics. Then we mention the termination predicate algorithm synthesized by our procedure (and the used measure function). Subsequently we show the results of applying the simplification procedure to the termination predicate. This procedure always consists of the four steps:

(a) Induction Lemma,
(b) Subsumption Elimination,
(c) Recursion Elimination,
(d) Case Elimination.

After each of these four steps, we mention the intermediate version of the termination predicate algorithm, where we omit steps that are not applicable in the particular example (and hence, do not change the termination predicate algorithm). In the end we describe the semantics of the resulting termination predicate.

The data structures used in the examples are nat (with the constructors 0 and succ and the selector pred) and list (with the constructors empty and add and the selectors head and tail). We omit termination predicates for constructors, selectors, and equality, because these predicates always return true.

1 minus

function $\mathsf{minus}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow$
  if $x = y$ *then* $\mathsf{0}$
    else $\mathsf{succ}(\mathsf{minus}(\mathsf{pred}(x), y))$

Intended Semantics: $x - y$

Synthesis

Measure: $m(x, y) = |x|_{\#}$ (i.e. the number of succ-applications in the first argument)

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{minus}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if} \;\; x = y \;\; \textit{then} \; \mathsf{true} \\
\qquad\qquad\quad \textit{else} \;\; |\mathsf{pred}(x)|_{\#} < |x|_{\#} \wedge \theta_{\mathsf{minus}}(\mathsf{pred}(x), y)
\end{array}
$$

Simplification

(a) Induction Lemma
$x \neq 0 \rightarrow |\mathsf{pred}(x)|_{\#} < |x|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{minus}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if} \;\; x = y \;\; \textit{then} \; \mathsf{true} \\
\qquad\qquad\quad \textit{else} \;\; x \neq 0 \wedge \theta_{\mathsf{minus}}(\mathsf{pred}(x), y)
\end{array}
$$

Semantics: $x \geq y$

2 half1

$$
\begin{array}{l}
\textit{function } \mathsf{half1}(x : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if} \;\; x = 0 \;\; \textit{then} \; 0 \\
\qquad\qquad\quad \textit{else} \;\; \mathsf{succ}(\mathsf{half1}(\mathsf{minus}(x, 2)))
\end{array}
$$

Intended Semantics: $x/2$

Synthesis

Measure: $m(x) = |x|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{half1}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if} \;\; x = 0 \;\; \textit{then} \; \mathsf{true} \\
\qquad\qquad\quad \textit{else} \;\; \theta_{\mathsf{minus}}(x, 2) \wedge |\mathsf{minus}(x, 2)|_{\#} < |x|_{\#} \wedge \theta_{\mathsf{half1}}(\mathsf{minus}(x, 2))
\end{array}
$$

Simplification

(a) Induction Lemma
$\theta_{\mathsf{minus}}(x, 2) \rightarrow |\mathsf{minus}(x, 2)|_{\#} < |x|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{half1}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if} \;\; x = 0 \;\; \textit{then} \; \mathsf{true} \\
\qquad\qquad\quad \textit{else} \;\; \theta_{\mathsf{minus}}(x, 2) \wedge \theta_{\mathsf{half1}}(\mathsf{minus}(x, 2))
\end{array}
$$

Semantics: true iff x is even

3 half2

$$
\begin{array}{l}
\textit{function}\ \mathsf{half2}(x : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ \mathsf{minus}(x, 2) = 0\ \ \textit{then}\ 1 \\
\qquad\qquad\qquad\qquad\quad\ \textit{else}\ \ \mathsf{succ}(\mathsf{half2}(\mathsf{minus}(x, 2)))
\end{array}
$$

Intended Semantics: $x/2$

Synthesis

Measure: $m(x) = |x|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{half2}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{minus}}(x, 2)\ \ \wedge\ \ (\ \textit{if}\ \ \mathsf{minus}(x, 2) = 0 \\
\qquad\qquad\qquad\qquad\quad \textit{then}\ \ \mathsf{true} \\
\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \ \theta_{\mathsf{minus}}(x, 2) \wedge |\mathsf{minus}(x, 2)|_\# < |x|_\# \wedge \theta_{\mathsf{half2}}(\mathsf{minus}(x, 2))\ )
\end{array}
$$

Simplification

(a) Induction Lemma

$\theta_{\mathsf{minus}}(x, 2) \to |\mathsf{minus}(x, 2)|_\# < |x|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{half2}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{minus}}(x, 2)\ \ \wedge\ \ (\ \textit{if}\ \ \mathsf{minus}(x, 2) = 0 \\
\qquad\qquad\qquad\qquad\quad \textit{then}\ \ \mathsf{true} \\
\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \ \theta_{\mathsf{minus}}(x, 2) \wedge \theta_{\mathsf{half2}}(\mathsf{minus}(x, 2))\ )
\end{array}
$$

(b) Subsumption Elimination

$\theta_{\mathsf{minus}}(x, 2) \to \theta_{\mathsf{minus}}(x, 2)$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{half2}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{minus}}(x, 2)\ \ \wedge\ \ (\ \textit{if}\ \ \mathsf{minus}(x, 2) = 0 \\
\qquad\qquad\qquad\qquad\quad \textit{then}\ \ \mathsf{true} \\
\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \ \theta_{\mathsf{half2}}(\mathsf{minus}(x, 2))\ )
\end{array}
$$

Semantics: true iff $x \neq 0$ and x is even

4 half3*

$$
\begin{array}{l}
\textit{function}\ \mathsf{half3}(x : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ x \neq 0 \wedge x \neq \mathsf{succ}(0)\ \ \textit{then}\ \mathsf{succ}(\mathsf{half3}(\mathsf{pred}(\mathsf{pred}(x)))) \\
\qquad\qquad\qquad\qquad\qquad\quad\ \textit{else}\ \ 0
\end{array}
$$

Intended Semantics: $x/2$

Synthesis

Measure: $m(x) = |x|_{\#}$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{half3}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x \neq 0 \wedge x \neq \mathsf{succ}(0)\ \ \textit{then}\ \ |\mathsf{pred}(\mathsf{pred}(x))|_{\#} < |x|_{\#} \wedge \theta_{\mathsf{half3}}(\mathsf{pred}(\mathsf{pred}(x))) \\ \qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{true} \end{array}$$

Simplification

(a) Induction Lemma

$x \neq 0 \wedge x \neq \mathsf{succ}(0) \to |\mathsf{pred}(\mathsf{pred}(x))|_{\#} < |x|_{\#}$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{half3}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x \neq 0 \wedge x \neq \mathsf{succ}(0)\ \ \textit{then}\ \ \theta_{\mathsf{half3}}(\mathsf{pred}(\mathsf{pred}(x))) \\ \qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{true} \end{array}$$

(c) Recursion Elimination

$$\begin{array}{l} x \neq 0 \wedge x \neq \mathsf{succ}(0) \to \mathsf{pred}(\mathsf{pred}(x)) \neq 0 \wedge \mathsf{pred}(\mathsf{pred}(x)) \neq \mathsf{succ}(0) \vee \\ \qquad\qquad\qquad\qquad\quad \neg(\mathsf{pred}(\mathsf{pred}(x)) \neq 0 \wedge \mathsf{pred}(\mathsf{pred}(x)) \neq \mathsf{succ}(0)) \end{array}$$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{half3}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x \neq 0 \wedge x \neq \mathsf{succ}(0)\ \ \textit{then}\ \ \mathsf{true} \\ \qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{true} \end{array}$$

(d) Case Elimination

$\ldots \to \mathsf{true}$

$$\textit{function}\ \theta_{\mathsf{half3}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad \mathsf{true}$$

Semantics: true

5 double*

$$\begin{array}{l} \textit{function}\ \mathsf{double}(x : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\ \quad \textit{if}\ \ x = 0\ \ \textit{then}\ \ 0 \\ \qquad\qquad\quad\ \textit{else}\ \ \mathsf{succ}(\mathsf{succ}(\mathsf{double}(\mathsf{pred}(x)))) \end{array}$$

Intended Semantics: $2x$

Synthesis

Measure: $m(x) = |x|_\#$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{double}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = 0 \quad \textit{then}\ \mathsf{true} \\ \qquad\qquad\quad \textit{else}\ \ |\mathsf{pred}(x)|_\# < |x|_\# \wedge \theta_{\mathsf{double}}(\mathsf{pred}(x)) \end{array}$$

Simplification

(a) Induction Lemma
$x \neq 0 \rightarrow |\mathsf{pred}(x)| < |x|_\#$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{double}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = 0 \quad \textit{then}\ \mathsf{true} \\ \qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{double}}(\mathsf{pred}(x)) \end{array}$$

(c) Recursion Elimination
$x \neq 0 \rightarrow \mathsf{pred}(x) = 0 \vee \mathsf{pred}(x) \neq 0$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{double}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = 0 \quad \textit{then}\ \mathsf{true} \\ \qquad\qquad\quad \textit{else}\ \ \mathsf{true} \end{array}$$

(d) Case Elimination
$\ldots \rightarrow \mathsf{true}$

$$\textit{function}\ \theta_{\mathsf{double}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad \mathsf{true}$$

Semantics: true

6 even*

$$\begin{array}{l} \textit{function}\ \mathsf{even}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = 0 \quad \textit{then}\ \mathsf{true} \\ \qquad\qquad\quad \textit{else}\ \ (\textit{if}\ \ x = \mathsf{succ}(0) \quad \textit{then}\ \mathsf{false} \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{even}(\mathsf{pred}(\mathsf{pred}(x)))\,) \end{array}$$

Intended Semantics: true iff x is even

Synthesis

Measure: $m(x) = |x|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{even}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\quad \textit{else}\ \ (\ \textit{if}\ \ x = \mathsf{succ}(0)\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ |\mathsf{pred}(\mathsf{pred}(x))|_\# < |x|_\# \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{even}}(\mathsf{pred}(\mathsf{pred}(x)))\ )
\end{array}
$$

Simplification

Analogously to θ_{half3}

Semantics: true

7 plus*

$$
\begin{array}{l}
\textit{function}\ \mathsf{plus}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ x = 0\ \ \textit{then}\ y \\
\qquad\qquad\quad \textit{else}\ \ \mathsf{succ}(\mathsf{plus}(\mathsf{pred}(x), y))
\end{array}
$$

Intended Semantics: $x + y$

Synthesis

Measure: $m(x, y) = |x|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{plus}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\quad \textit{else}\ \ |\mathsf{pred}(x)|_\# < |x|_\# \wedge \theta_{\mathsf{plus}}(\mathsf{pred}(x), y)
\end{array}
$$

Simplification

Analogously to θ_{double}

Semantics: true

8 times

$$
\begin{array}{l}
\textit{function}\ \mathsf{times}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ x = 0\ \ \textit{then}\ 0 \\
\qquad\qquad\quad \textit{else}\ \ (\ \textit{if}\ \ \mathsf{even}(x) \\
\qquad\qquad\qquad\qquad\quad \textit{then}\ \ \mathsf{times}(\mathsf{half1}(x), \mathsf{double}(y)) \\
\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \ \mathsf{plus}(y, \mathsf{times}(\mathsf{half1}(\mathsf{pred}(x)), \mathsf{double}(y)))\ )
\end{array}
$$

Intended Semantics: $x * y$

Synthesis

Measure: $m(x,y) = |x|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{times}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad \textit{else}\ \ \theta_{\mathsf{even}}(x) \ \ \wedge\ \ (\ \textit{if}\ \ \mathsf{even}(x) \\
\qquad\qquad\qquad\qquad \textit{then}\ \ \theta_{\mathsf{half1}}(x) \wedge \theta_{\mathsf{double}}(y) \wedge \\
\qquad\qquad\qquad\qquad\qquad |\mathsf{half1}(x)|_\# < |x|_\# \wedge \\
\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{times}}(\mathsf{half1}(x), \mathsf{double}(y)) \\
\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{half1}}(\mathsf{pred}(x)) \wedge \theta_{\mathsf{double}}(y) \wedge \\
\qquad\qquad\qquad\qquad\qquad |\mathsf{half1}(\mathsf{pred}(x))|_\# < |x|_\# \wedge \\
\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{times}}(\mathsf{half1}(\mathsf{pred}(x)), \mathsf{double}(y)) \wedge \\
\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{plus}}(y, \mathsf{times}(\mathsf{half1}(\mathsf{pred}(x)), \mathsf{double}(y)))\ )
\end{array}
$$

Simplification

(a) Induction Lemmata

$\theta_{\mathsf{half1}}(x) \wedge x \neq 0 \to |\mathsf{half1}(x)|_\# < |x|_\#$

$\theta_{\mathsf{half1}}(\mathsf{pred}(x)) \wedge x \neq 0 \to |\mathsf{half1}(\mathsf{pred}(x))|_\# < |x|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{times}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad \textit{else}\ \ \theta_{\mathsf{even}}(x) \ \ \wedge\ \ (\ \textit{if}\ \ \mathsf{even}(x) \\
\qquad\qquad\qquad\qquad \textit{then}\ \ \theta_{\mathsf{half1}}(x) \wedge \theta_{\mathsf{double}}(y) \wedge x \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{times}}(\mathsf{half1}(x), \mathsf{double}(y)) \\
\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{half1}}(\mathsf{pred}(x)) \wedge \theta_{\mathsf{double}}(y) \wedge x \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{times}}(\mathsf{half1}(\mathsf{pred}(x)), \mathsf{double}(y)) \wedge \\
\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{plus}}(y, \mathsf{times}(\mathsf{half1}(\mathsf{pred}(x)), \mathsf{double}(y)))\ )
\end{array}
$$

(b) Subsumption Elimination

$\ldots \to \theta_{\mathsf{even}}(x)$

$x \neq 0 \wedge \ \ \mathsf{even}(x) \to \theta_{\mathsf{half1}}(x) \wedge \theta_{\mathsf{double}}(y) \wedge x \neq 0$

$x \neq 0 \wedge \neg\mathsf{even}(x) \to \theta_{\mathsf{half1}}(\mathsf{pred}(x)) \wedge \theta_{\mathsf{double}}(y) \wedge x \neq 0$

$\ldots \to \theta_{\mathsf{plus}}(y, \mathsf{times}(\mathsf{half1}(\mathsf{pred}(x)), \mathsf{double}(y)))$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{times}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad \textit{else}\ \ (\ \textit{if}\ \ \mathsf{even}(x)\ \ \textit{then}\ \theta_{\mathsf{times}}(\mathsf{half1}(x), \mathsf{double}(y)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ \textit{else}\ \ \theta_{\mathsf{times}}(\mathsf{half1}(\mathsf{pred}(x)), \mathsf{double}(y))\ )
\end{array}
$$

(c) Recursion Elimination

$$\begin{array}{rl} x \neq 0 \wedge \ \ \mathsf{even}(x) \rightarrow & \mathsf{half1}(x) = 0 \vee \mathsf{half1}(x) \neq 0 \wedge \mathsf{even}(\mathsf{half1}(x)) \vee \\ & \mathsf{half1}(x) \neq 0 \wedge \neg\mathsf{even}(\mathsf{half1}(x)) \\ x \neq 0 \wedge \neg\mathsf{even}(x) \rightarrow & \mathsf{half1}(\mathsf{pred}(x)) = 0 \vee \\ & \mathsf{half1}(\mathsf{pred}(x)) \neq 0 \wedge \ \ \mathsf{even}(\mathsf{half1}(\mathsf{pred}(x))) \vee \\ & \mathsf{half1}(\mathsf{pred}(x)) \neq 0 \wedge \neg\mathsf{even}(\mathsf{half1}(\mathsf{pred}(x))) \end{array}$$

function $\theta_{\mathsf{times}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow$
  if $x = 0$ *then* true
    else (*if* even(x) *then* true
        else true)

(d) Case Elimination

$\ldots \rightarrow \mathsf{true}$

function $\theta_{\mathsf{times}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow$ true

Semantics: true

9 exp

function $\mathsf{exp}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow$
  if $y = 0$ *then* succ(0)
    else (*if* even(y)
      then exp(times(x, x), half1(y))
      else times(x, exp(times(x, x), half1(pred(y)))))

Intended Semantics: x^y

Synthesis

Measure: $m(x, y) = |y|_{\#}$

function $\theta_{\mathsf{exp}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow$
  if $y = 0$ *then* true
    else $\theta_{\mathsf{even}}(y)$ $\wedge$ (*if* even(y)
      then $\theta_{\mathsf{times}}(x, x) \wedge \theta_{\mathsf{half1}}(y) \wedge$
        $|\mathsf{half1}(y)|_{\#} < |y|_{\#} \wedge$
        $\theta_{\mathsf{exp}}(\mathsf{times}(x, x), \mathsf{half1}(y))$
      else $\theta_{\mathsf{times}}(x, x) \wedge \theta_{\mathsf{half1}}(\mathsf{pred}(y)) \wedge$
        $|\mathsf{half1}(\mathsf{pred}(y))|_{\#} < |y|_{\#} \wedge$
        $\theta_{\mathsf{exp}}(\mathsf{times}(x, x), \mathsf{half1}(\mathsf{pred}(y))) \wedge$
        $\theta_{\mathsf{times}}(x, \mathsf{exp}(\mathsf{times}(x, x), \mathsf{half1}(\mathsf{pred}(y)))))$

Simplification

Analogously to θ_{times}

Semantics: true

10 lt*

$$
\begin{array}{l}
\textit{function}\ \mathsf{lt}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ y = 0\ \ \textit{then}\ \mathsf{false} \\
\qquad\qquad\quad \textit{else}\ \ (\ \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\quad\ \textit{else}\ \ \mathsf{lt}(\mathsf{pred}(x), \mathsf{pred}(y))\ )
\end{array}
$$

Intended Semantics: $x < y$

Synthesis

Measure: $m(x, y) = |x|_{\#}$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{lt}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ y = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\quad \textit{else}\ \ (\ \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\quad\ \textit{else}\ \ |\mathsf{pred}(x)|_{\#} < |x|_{\#} \wedge \theta_{\mathsf{lt}}(\mathsf{pred}(x), \mathsf{pred}(y))\ )
\end{array}
$$

Simplification

Analogously to θ_{double}

Semantics: true

11 quotient1

$$
\begin{array}{l}
\textit{function}\ \mathsf{quotient1}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \ 0 \\
\qquad\qquad\qquad \textit{else}\ \ \mathsf{succ}(\mathsf{quotient1}(\mathsf{minus}(x, y), y))
\end{array}
$$

Intended Semantics: $\lfloor x/y \rfloor$

Synthesis

Measure: $m(x, y) = |x|_{\#}$

$$\begin{array}{l}
\textit{function}\ \theta_{\mathsf{quotient1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{lt}}(x, y) \quad \wedge \quad (\ \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{minus}}(x, y) \wedge |\mathsf{minus}(x, y)|_{\#} < |x|_{\#} \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{quotient1}}(\mathsf{minus}(x, y), y)\ )
\end{array}$$

Simplification

(a) Induction Lemma

$\theta_{\mathsf{minus}}(x, y) \wedge x \neq 0 \wedge y \neq 0 \rightarrow |\mathsf{minus}(x, y)|_{\#} < |x|_{\#}$

$$\begin{array}{l}
\textit{function}\ \theta_{\mathsf{quotient1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{lt}}(x, y) \quad \wedge \quad (\ \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{minus}}(x, y) \wedge x \neq 0 \wedge y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{quotient1}}(\mathsf{minus}(x, y), y)\ )
\end{array}$$

(b) Subsumption Elimination

$\ldots \rightarrow \theta_{\mathsf{lt}}(x, y)$

$\neg\mathsf{lt}(x, y) \rightarrow \theta_{\mathsf{minus}}(x, y)$

$\neg\mathsf{lt}(x, y) \wedge y \neq 0 \rightarrow x \neq 0$

$$\begin{array}{l}
\textit{function}\ \theta_{\mathsf{quotient1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad \textit{else}\ \ y \neq 0 \wedge \theta_{\mathsf{quotient1}}(\mathsf{minus}(x, y), y)
\end{array}$$

(c) Recursion Elimination

$\neg\mathsf{lt}(x, y) \wedge y \neq 0 \rightarrow \mathsf{lt}(\mathsf{minus}(x, y), y) \vee \neg\mathsf{lt}(\mathsf{minus}(x, y), y) \wedge y \neq 0$

$$\begin{array}{l}
\textit{function}\ \theta_{\mathsf{quotient1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad \textit{else}\ \ y \neq 0
\end{array}$$

(d) Case Elimination

$\mathsf{lt}(x, y) \rightarrow y \neq 0$

$$\textit{function}\ \theta_{\mathsf{quotient1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad y \neq 0$$

Semantics: $y \neq 0$

12 mod

$$\begin{array}{l}
\textit{function}\ \mathsf{mod}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ x \\
\qquad\qquad\qquad \textit{else}\ \ \mathsf{mod}(\mathsf{minus}(x, y), y)
\end{array}$$

Intended Semantics: Remainder of x w.r.t. y

Synthesis

Measure: $m(x, y) = |x|_{\#}$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{mod}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{lt}}(x, y) \quad \wedge \quad (\ \textit{if}\ \ \mathsf{lt}(x, y) \ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{minus}}(x, y) \wedge |\mathsf{minus}(x, y)|_{\#} < |x|_{\#} \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{mod}}(\mathsf{minus}(x, y), y)\ )
\end{array}
$$

Simplification

Analogously to $\theta_{\mathsf{quotient1}}$

Semantics: $y \neq 0$

13 quotient2

$$
\begin{array}{l}
\textit{function}\ \mathsf{quotient2}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ x = 0 \ \ \textit{then}\ \ 0 \\
\qquad\qquad\quad \textit{else}\ \ \mathsf{succ}(\mathsf{quotient2}(\mathsf{minus}(x, y), y))
\end{array}
$$

Intended Semantics: $\lfloor x/y \rfloor$

Synthesis

Measure: $m(x, y) = |x|_{\#}$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{quotient2}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = 0 \ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{minus}}(x, y) \wedge |\mathsf{minus}(x, y)|_{\#} < |x|_{\#} \wedge \\
\qquad\qquad\qquad\quad\ \ \theta_{\mathsf{quotient2}}(\mathsf{minus}(x, y), y)
\end{array}
$$

Simplification

(a) Induction Lemma
$\theta_{\mathsf{minus}}(x, y) \wedge x \neq 0 \wedge y \neq 0 \rightarrow |\mathsf{minus}(x, y)|_{\#} < |x|_{\#}$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{quotient2}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = 0 \ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{minus}}(x, y) \wedge y \neq 0 \wedge \theta_{\mathsf{quotient2}}(\mathsf{minus}(x, y), y)
\end{array}
$$

Semantics: true iff y divides x

14 gcd

$$
\begin{array}{l}
\textit{function}\ \mathsf{gcd}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ x = y\ \ \textit{then}\ x \\
\qquad\qquad\ \textit{else}\ \ (\ \textit{if}\ \ \mathsf{lt}(x,y)\ \ \textit{then}\ \mathsf{gcd}(x, \mathsf{minus}(y,x)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \textit{else}\ \ \mathsf{gcd}(\mathsf{minus}(x,y), y)\ )
\end{array}
$$

Intended Semantics: Greatest common divisor of x and y (this algorithm is from [Manna74])

Synthesis

Measure: $m(x, y) = |x|_\# + |y|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{gcd}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = y\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\ \textit{else}\ \ \theta_{\mathsf{lt}}(x,y)\ \ \wedge\ \ (\ \textit{if}\ \ \mathsf{lt}(x,y) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{then}\ \ \theta_{\mathsf{minus}}(y,x) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad |x|_\# + |\mathsf{minus}(y,x)|_\# < |x|_\# + |y|_\# \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{gcd}}(x, \mathsf{minus}(y,x)) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{minus}}(x,y) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad |\mathsf{minus}(x,y)|_\# + |y|_\# < |x|_\# + |y|_\# \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{gcd}}(\mathsf{minus}(x,y), y)\ )
\end{array}
$$

Simplification

(a) Induction Lemma

$\theta_{\mathsf{minus}}(v, w) \wedge v \neq 0 \wedge w \neq 0 \rightarrow |\mathsf{minus}(v,w)|_\# + |w|_\# < |v|_\# + |w|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{gcd}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = y\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\ \textit{else}\ \ \theta_{\mathsf{lt}}(x,y)\ \ \wedge\ \ (\ \textit{if}\ \ \mathsf{lt}(x,y) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{then}\ \ \theta_{\mathsf{minus}}(y,x) \wedge y \neq 0 \wedge x \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{gcd}}(x, \mathsf{minus}(y,x)) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{minus}}(x,y) \wedge x \neq 0 \wedge y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{gcd}}(\mathsf{minus}(x,y), y)\ )
\end{array}
$$

(b) Subsumption Elimination

$\ldots \rightarrow \theta_{\mathsf{lt}}(x, y)$

$x \neq y \wedge\ \ \mathsf{lt}(x,y) \rightarrow \theta_{\mathsf{minus}}(y,x) \wedge y \neq 0$

$x \neq y \wedge \neg\mathsf{lt}(x,y) \rightarrow \theta_{\mathsf{minus}}(x,y) \wedge x \neq 0$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{gcd}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = y\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\ \textit{else}\ \ (\ \textit{if}\ \ \mathsf{lt}(x,y)\ \ \textit{then}\ \ x \neq 0 \wedge \theta_{\mathsf{gcd}}(x, \mathsf{minus}(y,x)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \textit{else}\ \ y \neq 0 \wedge \theta_{\mathsf{gcd}}(\mathsf{minus}(x,y), y)\ )
\end{array}
$$

(c) Recursion Elimination

$$\begin{array}{ll} x \neq y \wedge \ \mathsf{lt}(x,y) \wedge x \neq 0 \rightarrow & x = \mathsf{minus}(y,x) \vee \\ & x \neq \mathsf{minus}(y,x) \wedge \ \mathsf{lt}(x, \mathsf{minus}(y,x)) \wedge x \neq 0 \vee \\ & x \neq \mathsf{minus}(y,x) \wedge \neg\mathsf{lt}(x, \mathsf{minus}(y,x)) \wedge \mathsf{minus}(y,x) \neq 0 \\ x \neq y \wedge \neg\mathsf{lt}(x,y) \wedge y \neq 0 \rightarrow & \mathsf{minus}(x,y) = y \vee \\ & \mathsf{minus}(x,y) \neq y \wedge \ \mathsf{lt}(\mathsf{minus}(x,y), y) \wedge \mathsf{minus}(x,y) \neq 0 \vee \\ & \mathsf{minus}(x,y) \neq y \wedge \neg\mathsf{lt}(\mathsf{minus}(x,y), y) \wedge y \neq 0 \end{array}$$

function $\theta_{\mathsf{gcd}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow$
  if $x = y$ *then* true
  else (*if* $\mathsf{lt}(x,y)$ *then* $x \neq 0$
  else $y \neq 0$)

Semantics: Either both, x and y, are zero or both are non-zero

15 lcm

function $\mathsf{lcm}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \quad \mathsf{times}(x, \mathsf{quotient1}(y, \mathsf{gcd}(x,y)))$

Intended Semantics: The least common multiple of x and y

Synthesis

function $\theta_{\mathsf{lcm}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad \theta_{\mathsf{gcd}}(x,y) \wedge \theta_{\mathsf{quotient1}}(y, \mathsf{gcd}(x,y)) \wedge$
$\theta_{\mathsf{times}}(x, \mathsf{quotient1}(y, \mathsf{gcd}(x,y)))$

Simplification

(b) Subsumption Elimination

$$\theta_{\mathsf{gcd}}(x,y) \wedge \theta_{\mathsf{quotient1}}(y, \mathsf{gcd}(x,y)) \rightarrow \theta_{\mathsf{times}}(x, \mathsf{quotient1}(y, \mathsf{gcd}(x,y)))$$

function $\theta_{\mathsf{lcm}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad \theta_{\mathsf{gcd}}(x,y) \wedge \theta_{\mathsf{quotient1}}(y, \mathsf{gcd}(x,y))$

Semantics: Both, x and y, are non-zero (as $\mathsf{gcd}(0,0) = 0$)

16 dual_log1

function $\mathsf{dual\_log1}(x : \mathsf{nat}) : \mathsf{nat} \Leftarrow$
  if $x = \mathsf{succ}(0)$ *then* 0
  else $\mathsf{succ}(\mathsf{dual\_log1}(\mathsf{half3}(x)))$

Intended Semantics: The dual logarithm of x

Synthesis

Measure: $m(x) = |x|_\#$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{dual\_log1}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = \mathsf{succ}(0)\ \ \textit{then}\ \mathsf{true} \\ \qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{half3}}(x) \wedge |\mathsf{half3}(x)|_\# < |x|_\# \wedge \theta_{\mathsf{dual\_log1}}(\mathsf{half3}(x)) \end{array}$$

Simplification

(a) Induction Lemma
$\theta_{\mathsf{half3}}(x) \wedge x \neq 0 \rightarrow |\mathsf{half3}(x)|_\# < |x|_\#$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{dual\_log1}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = \mathsf{succ}(0)\ \ \textit{then}\ \mathsf{true} \\ \qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{half3}}(x) \wedge x \neq 0 \wedge \theta_{\mathsf{dual\_log1}}(\mathsf{half3}(x)) \end{array}$$

(b) Subsumption Elimination
$\ldots \rightarrow \theta_{\mathsf{half3}}(x)$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{dual\_log1}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = \mathsf{succ}(0)\ \ \textit{then}\ \mathsf{true} \\ \qquad\qquad\qquad\qquad \textit{else}\ \ x \neq 0 \wedge \theta_{\mathsf{dual\_log1}}(\mathsf{half3}(x)) \end{array}$$

(c) Recursion Elimination
$$\begin{array}{l} x \neq \mathsf{succ}(0) \wedge x \neq 0 \rightarrow \\ \qquad\qquad \mathsf{half3}(x) = \mathsf{succ}(0) \vee \mathsf{half3}(x) \neq \mathsf{succ}(0) \wedge \mathsf{half3}(x) \neq 0 \end{array}$$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{dual\_log1}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ x = \mathsf{succ}(0)\ \ \textit{then}\ \mathsf{true} \\ \qquad\qquad\qquad\qquad \textit{else}\ \ x \neq 0 \end{array}$$

(d) Case Elimination
$x = \mathsf{succ}(0) \rightarrow x \neq 0$

$$\textit{function}\ \theta_{\mathsf{dual\_log1}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad x \neq 0$$

Semantics: $x \neq 0$

17 dual_log2

$$\begin{array}{l} \textit{function}\ \mathsf{dual\_log2}(x : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\ \quad \textit{if}\ \ x = \mathsf{succ}(0)\ \ \textit{then}\ 0 \\ \qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{succ}(\mathsf{dual\_log2}(\mathsf{half1}(x))) \end{array}$$

Intended Semantics: The dual logarithm of x

Synthesis

Measure: $m(x) = |x|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{dual\_log2}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; x = \mathsf{succ}(0) \;\; \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad\; \textit{else } \; \theta_{\mathsf{half1}}(x) \wedge |\mathsf{half1}(x)|_{\#} < |x|_{\#} \wedge \theta_{\mathsf{dual\_log2}}(\mathsf{half1}(x))
\end{array}
$$

Simplification

(a) Induction Lemma

$\theta_{\mathsf{half1}}(x) \wedge x \neq 0 \rightarrow |\mathsf{half1}(x)|_{\#} < |x|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{dual\_log2}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; x = \mathsf{succ}(0) \;\; \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad\; \textit{else } \; \theta_{\mathsf{half1}}(x) \wedge x \neq 0 \wedge \theta_{\mathsf{dual\_log2}}(\mathsf{half1}(x))
\end{array}
$$

Semantics: $x = 2^n$ (for some $n \in \mathbb{N}$)

18 log1

$$
\begin{array}{l}
\textit{function } \mathsf{log1}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if } \; x = \mathsf{succ}(0) \;\; \textit{then } 0 \\
\qquad\qquad\qquad\qquad\; \textit{else } \; \mathsf{succ}(\mathsf{log1}(\mathsf{quotient1}(x, y), y))
\end{array}
$$

Intended Semantics: The logarithm of x w.r.t. y

Synthesis

Measure: $m(x, y) = |x|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{log1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; x = \mathsf{succ}(0) \;\; \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad\; \textit{else } \; \theta_{\mathsf{quotient1}}(x, y) \wedge |\mathsf{quotient1}(x, y)|_{\#} < |x|_{\#} \wedge \\
\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{log1}}(\mathsf{quotient1}(x, y), y)
\end{array}
$$

Simplification

(a) Induction Lemma

$\theta_{\mathsf{quotient1}}(x, y) \wedge x \neq 0 \wedge y \neq \mathsf{succ}(0) \rightarrow |\mathsf{quotient1}(x, y)|_{\#} < |x|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{log1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; x = \mathsf{succ}(0) \;\; \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad\; \textit{else } \; \theta_{\mathsf{quotient1}}(x, y) \wedge x \neq 0 \wedge y \neq \mathsf{succ}(0) \wedge \\
\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{log1}}(\mathsf{quotient1}(x, y), y)
\end{array}
$$

(c) Recursion Elimination

$$\begin{aligned} x \neq \mathsf{succ}(0) \wedge \theta_{\mathsf{quotient1}}(x, y) \wedge x \neq 0 \wedge y \neq \mathsf{succ}(0) \rightarrow \\ & \mathsf{quotient1}(x, y) = \mathsf{succ}(0) \vee \\ & \mathsf{quotient1}(x, y) \neq \mathsf{succ}(0) \wedge \\ & \theta_{\mathsf{quotient1}}(\mathsf{quotient1}(x, y), y) \wedge \\ & \mathsf{quotient1}(x, y) \neq 0 \wedge y \neq \mathsf{succ}(0) \end{aligned}$$

$$\begin{array}{l} \textit{function } \theta_{\mathsf{log1}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if } \ x = \mathsf{succ}(0) \ \ \textit{then } \ \mathsf{true} \\ \qquad\qquad\qquad\qquad\quad \textit{else } \ \theta_{\mathsf{quotient1}}(x, y) \wedge x \neq 0 \wedge y \neq \mathsf{succ}(0) \end{array}$$

Semantics: $x = 1 \vee x \neq 0 \wedge y \neq 0 \wedge y \neq 1$

19 log2

$$\begin{array}{l} \textit{function } \mathsf{log2}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\ \quad \textit{if } \ x = \mathsf{succ}(0) \ \ \textit{then } \ 0 \\ \qquad\qquad\qquad\qquad\quad \textit{else } \ \mathsf{succ}(\mathsf{log2}(\mathsf{quotient2}(x, y), y)) \end{array}$$

Intended Semantics: The logarithm of x w.r.t. y

Synthesis

Measure: $m(x, y) = |x|_{\#}$

$$\begin{array}{l} \textit{function } \theta_{\mathsf{log2}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if } \ x = \mathsf{succ}(0) \ \ \textit{then } \ \mathsf{true} \\ \qquad\qquad\qquad\qquad\quad \textit{else } \ \theta_{\mathsf{quotient2}}(x, y) \wedge |\mathsf{quotient2}(x, y)|_{\#} < |x|_{\#} \wedge \\ \qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{log2}}(\mathsf{quotient2}(x, y), y) \end{array}$$

Simplification

(a) Induction Lemma

$$\theta_{\mathsf{quotient2}}(x, y) \wedge x \neq 0 \wedge y \neq \mathsf{succ}(0) \rightarrow |\mathsf{quotient2}(x, y)|_{\#} < |x|_{\#}$$

$$\begin{array}{l} \textit{function } \theta_{\mathsf{log2}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if } \ x = \mathsf{succ}(0) \ \ \textit{then } \ \mathsf{true} \\ \qquad\qquad\qquad\qquad\quad \textit{else } \ \theta_{\mathsf{quotient2}}(x, y) \wedge x \neq 0 \wedge y \neq \mathsf{succ}(0) \wedge \\ \qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{log2}}(\mathsf{quotient2}(x, y), y) \end{array}$$

Semantics: $x = 1 \vee x = y^n \wedge x \neq 0 \wedge y \neq 1$ (for some $n \in \mathbb{N}$)

20 list_minus

function $\mathsf{list\_minus}(l : \mathsf{list}, y : \mathsf{nat}) : \mathsf{list} \Leftarrow$
$\quad$ *if* $l = \mathsf{empty}$ *then* empty
$\quad\quad\quad\quad$ *else* $\mathsf{add}(\mathsf{minus}(\mathsf{head}(l), y), \mathsf{list\_minus}(\mathsf{tail}(l), y))$

Intended Semantics: Subtracts y from each element of l

Synthesis

Measure: $m(l, y) = |l|_\#$ (i.e. the length of the list l)

function $\theta_{\mathsf{list\_minus}}(l : \mathsf{list}, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow$
$\quad$ *if* $l = \mathsf{empty}$ *then* true
$\quad\quad\quad\quad$ *else* $\theta_{\mathsf{minus}}(\mathsf{head}(l), y) \wedge |\mathsf{tail}(l)|_\# < |l|_\# \wedge \theta_{\mathsf{list\_minus}}(\mathsf{tail}(l), y)$

Simplification

(a) Induction Lemma
$l \neq \mathsf{empty} \rightarrow |\mathsf{tail}(l)|_\# < |l|_\#$

function $\theta_{\mathsf{list\_minus}}(l : \mathsf{list}, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow$
$\quad$ *if* $l = \mathsf{empty}$ *then* true
$\quad\quad\quad\quad$ *else* $\theta_{\mathsf{minus}}(\mathsf{head}(l), y) \wedge \theta_{\mathsf{list\_minus}}(\mathsf{tail}(l), y)$

Semantics: Each element of l is greater than or equal to y

21 last

function $\mathsf{last}(l : \mathsf{list}) : \mathsf{nat} \Leftarrow$
$\quad$ *if* $l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty})$ *then* $\mathsf{head}(l)$
$\quad\quad\quad\quad\quad\quad$ *else* $\mathsf{last}(\mathsf{tail}(l))$

Intended Semantics: The last element of l

Synthesis

Measure: $m(l) = |l|_\#$

function $\theta_{\mathsf{last}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow$
$\quad$ *if* $l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty})$ *then* true
$\quad\quad\quad\quad\quad\quad$ *else* $|\mathsf{tail}(l)|_\# < |l|_\# \wedge \theta_{\mathsf{last}}(\mathsf{tail}(l))$

Simplification

(a) Induction Lemma

$l \neq \mathsf{empty} \rightarrow |\mathsf{tail}(l)|_{\#} < |l|_{\#}$

$$\begin{array}{l} \textit{function } \theta_{\mathsf{last}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if } \ l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \quad \textit{then } \mathsf{true} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{else } \ l \neq \mathsf{empty} \wedge \theta_{\mathsf{last}}(\mathsf{tail}(l)) \end{array}$$

(c) Recursion Elimination

$$\begin{array}{l} l \neq \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \wedge l \neq \mathsf{empty} \rightarrow \\ \qquad\qquad \mathsf{tail}(l) = \mathsf{add}(\mathsf{head}(\mathsf{tail}(l)), \mathsf{empty}) \vee \\ \qquad\qquad \mathsf{tail}(l) \neq \mathsf{add}(\mathsf{head}(\mathsf{tail}(l)), \mathsf{empty}) \wedge \mathsf{tail}(l) \neq \mathsf{empty} \end{array}$$

$$\begin{array}{l} \textit{function } \theta_{\mathsf{last}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if } \ l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \quad \textit{then } \mathsf{true} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{else } \ l \neq \mathsf{empty} \end{array}$$

(d) Case Elimination

$l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \rightarrow l \neq \mathsf{empty}$

$$\textit{function } \theta_{\mathsf{last}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \quad l \neq \mathsf{empty}$$

Semantics: $l \neq \mathsf{empty}$

22 but_last

$$\begin{array}{l} \textit{function } \mathsf{but\_last}(l : \mathsf{list}) : \mathsf{list} \Leftarrow \\ \quad \textit{if } \ l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \quad \textit{then } \mathsf{empty} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{else } \ \mathsf{add}(\mathsf{head}(l), \mathsf{but\_last}(\mathsf{tail}(l))) \end{array}$$

Intended Semantics: A copy of l with all elements but the last

Synthesis

Measure: $m(l) = |l|_{\#}$

$$\begin{array}{l} \textit{function } \theta_{\mathsf{but\_last}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if } \ l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \quad \textit{then } \mathsf{true} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{else } \ |\mathsf{tail}(l)|_{\#} < |l|_{\#} \wedge \theta_{\mathsf{but\_last}}(\mathsf{tail}(l)) \end{array}$$

Simplification

Analogously to θ_{last}

Semantics: $l \neq \mathsf{empty}$

23 reverse

$$
\begin{array}{l}
\textit{function}\ \mathsf{reverse}(l : \mathsf{list}) : \mathsf{list} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ \mathsf{empty} \\
\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{add}(\mathsf{last}(l), \mathsf{reverse}(\mathsf{but\_last}(l)))
\end{array}
$$

Intended Semantics: Reverses l

Synthesis

Measure: $m(l) = |l|_{\#}$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{reverse}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{last}}(l) \wedge \theta_{\mathsf{but\_last}}(l) \wedge |\mathsf{but\_last}(l)|_{\#} < |l|_{\#} \wedge \\
\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{reverse}}(\mathsf{but\_last}(l))
\end{array}
$$

Simplification

(a) Induction Lemma
$\theta_{\mathsf{but\_last}}(l) \rightarrow |\mathsf{but\_last}(l)|_{\#} < |l|_{\#}$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{reverse}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{last}}(l) \wedge \theta_{\mathsf{but\_last}}(l) \wedge \theta_{\mathsf{reverse}}(\mathsf{but\_last}(l))
\end{array}
$$

(b) Subsumption Elimination
$l \neq \mathsf{empty} \rightarrow \theta_{\mathsf{last}}(l) \wedge \theta_{\mathsf{but\_last}}(l)$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{reverse}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{reverse}}(\mathsf{but\_last}(l))
\end{array}
$$

(c) Recursion Elimination
$l \neq \mathsf{empty} \rightarrow \mathsf{but\_last}(l) = \mathsf{empty} \vee \mathsf{but\_last}(l) \neq \mathsf{empty}$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{reverse}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{true}
\end{array}
$$

(d) Case Elimination

$\ldots \rightarrow \mathsf{true}$

$$\textit{function}\ \theta_{\mathsf{reverse}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \quad \mathsf{true}$$

Semantics: true

24 minimum*

$$\begin{array}{l} \textit{function}\ \mathsf{minimum}(x, y : \mathsf{nat}) : \mathsf{nat} \Leftarrow \\ \quad \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ x \\ \qquad\qquad\qquad\ \ \textit{else}\ \ y \end{array}$$

Intended Semantics: The minimum of x and y

Synthesis

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{minimum}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \theta_{\mathsf{lt}}(x, y) \quad \wedge \quad (\ \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \mathsf{true} \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{true}\ ) \end{array}$$

Simplification

(b) Subsumption Elimination

$\ldots \rightarrow \theta_{\mathsf{lt}}(x, y)$

$$\begin{array}{l} \textit{function}\ \theta_{\mathsf{minimum}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\ \quad \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \mathsf{true} \\ \qquad\qquad\qquad\ \ \textit{else}\ \ \mathsf{true} \end{array}$$

(d) Case Elimination

$\ldots \rightarrow \mathsf{true}$

$$\textit{function}\ \theta_{\mathsf{minimum}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad \mathsf{true}$$

Semantics: true

25 list_min

$$\begin{array}{l} \textit{function}\ \mathsf{list\_min}(l : \mathsf{list}) : \mathsf{nat} \Leftarrow \\ \quad \textit{if}\ \ l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty})\ \ \textit{then}\ \mathsf{head}(l) \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{minimum}(\mathsf{head}(l), \mathsf{list\_min}(\mathsf{tail}(l))) \end{array}$$

Intended Semantics: The minimum among the elements of l

Synthesis

Measure: $m(l) = |l|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{list\_min}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty})\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else}\ \ |\mathsf{tail}(l)|_\# < |l|_\# \wedge \theta_{\mathsf{list\_min}}(\mathsf{tail}(l)) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{minimum}}(\mathsf{head}(l), \mathsf{list\_min}(\mathsf{tail}(l)))
\end{array}
$$

Simplification

Analogously to θ_{last}

Semantics: $l \neq \mathsf{empty}$

26 length*

$$
\begin{array}{l}
\textit{function}\ \mathsf{length}(l : \mathsf{list}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ 0 \\
\qquad\qquad\qquad\quad \textit{else}\ \ \mathsf{succ}(\mathsf{length}(\mathsf{tail}(l)))
\end{array}
$$

Intended Semantics: Length of l

Synthesis

Measure: $m(l) = |l|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{length}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\quad \textit{else}\ \ |\mathsf{tail}(l)|_\# < |l|_\# \wedge \theta_{\mathsf{length}}(\mathsf{tail}(l))
\end{array}
$$

Simplification

(a) Induction Lemma
$l \neq \mathsf{empty} \rightarrow |\mathsf{tail}(l)|_\# < |l|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{length}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{empty}\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\quad \textit{else}\ \ \theta_{\mathsf{length}}(\mathsf{tail}(l))
\end{array}
$$

(c) Recursion Elimination
$l \neq \mathsf{empty} \rightarrow \mathsf{tail}(l) = \mathsf{empty} \vee \mathsf{tail}(l) \neq \mathsf{empty}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{length}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; l = \mathsf{empty} \quad \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\quad\; \textit{else } \; \mathsf{true}
\end{array}
$$

(d) Case Elimination

$\ldots \rightarrow \mathsf{true}$

$$\textit{function } \theta_{\mathsf{length}}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow \quad \mathsf{true}$$

Semantics: true

27 last_x

$$
\begin{array}{l}
\textit{function } \mathsf{last\_x}(l : \mathsf{list}, x : \mathsf{nat}) : \mathsf{list} \Leftarrow \\
\quad \textit{if } \; \mathsf{length}(l) = x \quad \textit{then } l \\
\qquad\qquad\qquad\qquad\;\; \textit{else } \; \mathsf{last\_x}(\mathsf{tail}(l), x)
\end{array}
$$

Intended Semantics: The list of the last x elements of l

Synthesis

Measure: $m(l, x) = |l|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{last\_x}}(l : \mathsf{list}, x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{length}}(l) \quad \wedge \quad (\, \textit{if } \; \mathsf{length}(l) = x \quad \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else } \; |\mathsf{tail}(l)|_{\#} < |l|_{\#} \wedge \theta_{\mathsf{last\_x}}(\mathsf{tail}(l), x)\,)
\end{array}
$$

Simplification

(a) Induction Lemma

$l \neq \mathsf{empty} \rightarrow |\mathsf{tail}(l)|_{\#} < |l|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{last\_x}}(l : \mathsf{list}, x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \theta_{\mathsf{length}}(l) \quad \wedge \quad (\, \textit{if } \; \mathsf{length}(l) = x \quad \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else } \; l \neq \mathsf{empty} \wedge \theta_{\mathsf{last\_x}}(\mathsf{tail}(l), x)\,)
\end{array}
$$

(b) Subsumption Elimination

$\ldots \rightarrow \theta_{\mathsf{length}}(l)$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{last\_x}}(l : \mathsf{list}, x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; \mathsf{length}(l) = x \quad \textit{then } \mathsf{true} \\
\qquad\qquad\qquad\qquad\;\; \textit{else } \; l \neq \mathsf{empty} \wedge \theta_{\mathsf{last\_x}}(\mathsf{tail}(l), x)
\end{array}
$$

Semantics: $\mathsf{length}(l) \geq x$

28 index

$$
\begin{array}{l}
\textit{function}\ \mathsf{index}(x : \mathsf{nat}, l : \mathsf{list}) : \mathsf{nat} \Leftarrow \\
\quad \textit{if}\ \ x = \mathsf{head}(l) \ \ \textit{then}\ \mathsf{succ}(0) \\
\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{succ}(\mathsf{index}(x, \mathsf{tail}(l)))
\end{array}
$$

Intended Semantics: The position of the first occurrence of x in l (beginning with 1)

Synthesis

Measure: $m(x, l) = |l|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{index}}(x : \mathsf{nat}, l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = \mathsf{head}(l) \ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad \textit{else}\ \ |\mathsf{tail}(l)|_\# < |l|_\# \wedge \theta_{\mathsf{index}}(x, \mathsf{tail}(l))
\end{array}
$$

Simplification

(a) Induction Lemma
$l \neq \mathsf{empty} \rightarrow |\mathsf{tail}(l)|_\# < |l|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{index}}(x : \mathsf{nat}, l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = \mathsf{head}(l) \ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad \textit{else}\ \ l \neq \mathsf{empty} \wedge \theta_{\mathsf{index}}(x, \mathsf{tail}(l))
\end{array}
$$

Semantics: $x = 0$ or x occurs in l

29 delete

$$
\begin{array}{l}
\textit{function}\ \mathsf{delete}(x : \mathsf{nat}, l : \mathsf{list}) : \mathsf{list} \Leftarrow \\
\quad \textit{if}\ \ x = \mathsf{head}(l) \ \ \textit{then}\ \mathsf{tail}(l) \\
\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{add}(\mathsf{head}(l), \mathsf{delete}(x, \mathsf{tail}(l)))
\end{array}
$$

Intended Semantics: Removes the first occurrence of x from l

Synthesis

Measure: $m(x, l) = |l|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{delete}}(x : \mathsf{nat}, l : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ x = \mathsf{head}(l) \ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad \textit{else}\ \ |\mathsf{tail}(l)|_\# < |l|_\# \wedge \theta_{\mathsf{delete}}(x, \mathsf{tail}(l))
\end{array}
$$

Simplification

Analogously to θ_{index}

Semantics: $x = 0$ or x occurs in l

30 sum_lists

$$
\begin{array}{l}
\textit{function } \mathsf{sum\_lists}(l, k : \mathsf{list}) : \mathsf{list} \Leftarrow \\
\quad \textit{if } \; l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \wedge \\
\qquad k = \mathsf{add}(\mathsf{head}(k), \mathsf{empty}) \quad \textit{then } \; \mathsf{add}(\mathsf{plus}(\mathsf{head}(l), \mathsf{head}(k)), \mathsf{empty}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else } \; \mathsf{add}(\mathsf{plus}(\mathsf{head}(l), \mathsf{head}(k)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{sum\_lists}(\mathsf{tail}(l), \mathsf{tail}(k)))
\end{array}
$$

Intended Semantics: Computes the list whose elements are the sums of the corresponding elements of l and k

Synthesis

Measure: $m(l, k) = |l|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{sum\_lists}}(l, k : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \wedge \\
\qquad k = \mathsf{add}(\mathsf{head}(k), \mathsf{empty}) \quad \textit{then } \; \theta_{\mathsf{plus}}(\mathsf{head}(l), \mathsf{head}(k)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else } \; \theta_{\mathsf{plus}}(\mathsf{head}(l), \mathsf{head}(k)) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad |\mathsf{tail}(l)|_{\#} < |l|_{\#} \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{sum\_lists}}(\mathsf{tail}(l), \mathsf{tail}(k))
\end{array}
$$

Simplification

(a) Induction Lemma
$l \neq \mathsf{empty} \rightarrow |\mathsf{tail}(l)|_{\#} < |l|_{\#}$

$$
\begin{array}{l}
\textit{function } \theta_{\mathsf{sum\_lists}}(l, k : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if } \; l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \wedge \\
\qquad k = \mathsf{add}(\mathsf{head}(k), \mathsf{empty}) \quad \textit{then } \; \theta_{\mathsf{plus}}(\mathsf{head}(l), \mathsf{head}(k)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{else } \; \theta_{\mathsf{plus}}(\mathsf{head}(l), \mathsf{head}(k)) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad l \neq \mathsf{empty} \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{sum\_lists}}(\mathsf{tail}(l), \mathsf{tail}(k))
\end{array}
$$

(b) Subsumption Elimination
$\ldots \rightarrow \theta_{\mathsf{plus}}(\mathsf{head}(l), \mathsf{head}(k))$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{sum\_lists}}(l, k : \mathsf{list}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ l = \mathsf{add}(\mathsf{head}(l), \mathsf{empty}) \wedge \\
\qquad k = \mathsf{add}(\mathsf{head}(k), \mathsf{empty})\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ l \neq \mathsf{empty} \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{sum\_lists}}(\mathsf{tail}(l), \mathsf{tail}(k))
\end{array}
$$

Semantics: l and k are not empty and have the same length

31 nat_to_bin

$$
\begin{array}{l}
\textit{function}\ \mathsf{nat\_to\_bin}(x, y : \mathsf{nat}) : \mathsf{list} \Leftarrow \\
\quad \textit{if}\ \ y = 1\ \ \textit{then}\ (\ \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{add}(0, \mathsf{empty}) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{add}(1, \mathsf{empty})\ ) \\
\qquad\qquad\qquad \textit{else}\ \ (\ \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \mathsf{add}(0, \mathsf{nat\_to\_bin}(x, \mathsf{half1}(y))) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{add}(1, \mathsf{nat\_to\_bin}(\mathsf{minus}(x, y), \mathsf{half1}(y)))\ )
\end{array}
$$

Intended Semantics: The binary representation of x is computed, if $y = 2^n \leq x$, n maximal (or if $y = 1$ and $x = 0$).

Synthesis

Measure: $m(x, y) = |y|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{nat\_to\_bin}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ y = 1\ \ \textit{then}\ (\ \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{true}\ ) \\
\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{lt}}(x, y)\ \wedge\ (\ \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \theta_{\mathsf{half1}}(y) \wedge |\mathsf{half1}(y)|_\# < |y|_\# \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{nat\_to\_bin}}(x, \mathsf{half1}(y)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{minus}}(x, y) \wedge \theta_{\mathsf{half1}}(y) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad |\mathsf{half1}(y)|_\# < |y|_\# \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{nat\_to\_bin}}(\mathsf{minus}(x, y), \mathsf{half1}(y))\ )
\end{array}
$$

Simplification

(a) Induction Lemma
$\theta_{\mathsf{half1}}(y) \wedge y \neq 0 \rightarrow |\mathsf{half1}(y)|_\# < |y|_\#$

$$
\begin{array}{l}
\textit{function}\ \theta_{\mathsf{nat\_to\_bin}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ y = 1\ \ \textit{then}\ (\ \textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{true}\ ) \\
\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{lt}}(x, y)\ \wedge\ (\ \textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \theta_{\mathsf{half1}}(y) \wedge y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{nat\_to\_bin}}(x, \mathsf{half1}(y)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{minus}}(x, y) \wedge \theta_{\mathsf{half1}}(y) \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta_{\mathsf{nat\_to\_bin}}(\mathsf{minus}(x, y), \mathsf{half1}(y))\ )
\end{array}
$$

(b) Subsumption Elimination

$\ldots \rightarrow \theta_{\mathsf{lt}}(x, y)$

$\neg \mathsf{lt}(x, y) \rightarrow \theta_{\mathsf{minus}}(x, y)$

$$\begin{array}{l}
\textit{function}\ \theta_{\mathsf{nat_to_bin}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ y = 1\ \ \textit{then}\ (\textit{if}\ \ x = 0\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \mathsf{true}\,) \\
\qquad\qquad\quad \textit{else}\ \ (\textit{if}\ \ \mathsf{lt}(x, y)\ \textit{then}\ \theta_{\mathsf{half1}}(y) \wedge y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{nat_to_bin}}(x, \mathsf{half1}(y)) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{half1}}(y) \wedge y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{nat_to_bin}}(\mathsf{minus}(x, y), \mathsf{half1}(y))\,)
\end{array}$$

(d) Case Elimination

$\ldots \rightarrow \mathsf{true}$

$$\begin{array}{l}
\textit{function}\ \theta_{\mathsf{nat_to_bin}}(x, y : \mathsf{nat}) : \mathsf{bool} \Leftarrow \\
\quad \textit{if}\ \ y = 1\ \ \textit{then}\ \mathsf{true} \\
\qquad\qquad\quad \textit{else}\ \ (\textit{if}\ \ \mathsf{lt}(x, y)\ \ \textit{then}\ \theta_{\mathsf{half1}}(y) \wedge y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{nat_to_bin}}(x, \mathsf{half1}(y)) \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{else}\ \ \theta_{\mathsf{half1}}(y) \wedge y \neq 0 \wedge \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \theta_{\mathsf{nat_to_bin}}(\mathsf{minus}(x, y), \mathsf{half1}(y))\,)
\end{array}$$

Semantics: $y = 2^n$ (for some $n \in \mathbb{N}$)

32 bin_vec

$\textit{function}\ \mathsf{bin\_vec}(x : \mathsf{nat}) : \mathsf{list} \Leftarrow \quad \mathsf{nat\_to\_bin}(x, \mathsf{exp}(2, \mathsf{dual\_log1}(x)))$

Intended Semantics: The binary representation of x

Synthesis

$$\begin{array}{ll}
\textit{function}\ \theta_{\mathsf{bin_vec}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow & \theta_{\mathsf{dual_log1}}(x) \wedge \theta_{\mathsf{exp}}(2, \mathsf{dual_log1}(x)) \wedge \\
& \theta_{\mathsf{nat_to_bin}}(x, \mathsf{exp}(2, \mathsf{dual_log1}(x)))
\end{array}$$

Simplification

(b) Subsumption Elimination

$\theta_{\mathsf{dual\_log1}}(x) \rightarrow \theta_{\mathsf{exp}}(2, \mathsf{dual\_log1}(x)) \wedge \theta_{\mathsf{nat\_to\_bin}}(x, \mathsf{exp}(2, \mathsf{dual\_log1}(x)))$

$\textit{function}\ \theta_{\mathsf{bin\_vec}}(x : \mathsf{nat}) : \mathsf{bool} \Leftarrow \quad \theta_{\mathsf{dual\_log1}}(x)$

Semantics: $x \neq 0$

References

[BG96] J. Brauburger & J. Giesl. Termination Analysis for Partial Functions. In *Proceedings of the Third International Static Analysis Symposium, LNCS 1145*, Aachen, Germany, 1996.

[Bi+86] S. Biundo, B. Hummel, D. Hutter & C. Walther. The Karlsruhe Induction Theorem Proving System. *Pr. 8th CADE, LNCS 230*, Oxford, England, 1986.

[BM79] R. S. Boyer & J S. Moore. *A Computational Logic.* Academic Press, 1979.

[BM88] R. S. Boyer & J S. Moore. The Addition of Bounded Quantification and Partial Functions to A Computational Logic and Its Theorem Prover. *Journal of Automated Reasoning*, 4:117-172, 1988.

[Bu+90] A. Bundy, F. van Harmelen, C. Horn & A. Smaill. The OYSTER-CLAM System. In *Proc. 10th CADE, LNAI 449*, Kaiserslautern, Germany, 1990.

[Der87] N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1, 2):69-115, 1987.

[Gie95a] J. Giesl. Generating Polynomial Orderings for Termination Proofs. *Pr. 6th Int. Conf. Rewr. Tech. & App., LNCS 914*, Kaiserslautern, Germany, 1995.

[Gie95b] J. Giesl. Automated Termination Proofs with Measure Functions. In *Proc. 19th Annual German Conf. on AI, LNAI 981*, Bielefeld, Germany, 1995.

[Gie95c] J. Giesl. Termination Analysis for Functional Programs using Term Orderings. *Pr. 2nd Int. Stat. Analysis Symp., LNCS 983*, Glasgow, Scotland, 1995.

[Gie96] J. Giesl. Termination of Nested and Mutually Recursive Algorithms. *Journal of Automated Reasoning*, 1996. To appear.

[Hen80] P. Henderson. *Functional Programming.* Prentice-Hall, London, 1980.

[Hol91] C. K. Holst. Finiteness Analysis. In *Proc. 5th ACM Conf. Functional Prog. Languages & Comp. Architecture*, LNCS 523, Cambridge, MA, USA, 1991.

[Man74] Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill, 1974.

[NN95] F. Nielson & H. R. Nielson. Termination Analysis based on Operational Semantics. Technical Report, Aarhus University, Denmark, 1995.

[Plü90] L. Plümer. *Termination Proofs for Logic Programs. LNAI 446*, Springer–Verlag, 1990.

[SD94] D. De Schreye & S. Decorte. Termination of Logic Programs: The Never-Ending Story. *Journal of Logic Programming* 19,20:199-260, 1994.

[Ste95] J. Steinbach. Simplification Orderings: History of Results. *Fundamenta Informaticae* 24:47-87, 1995.

[UV88] J. D. Ullman & A. van Gelder. Efficient Tests for Top-Down Termination of Logical Rules. *Journal of the ACM*, 35(2):345-373, 1988.

[Wal88] C. Walther. Argument-Bounded Algorithms as a Basis for Automated Termination Proofs. In *Proc. 9th CADE, LNCS 310*, Argonne, IL, 1988.

[Wal94a] C. Walther. Mathematical Induction. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. 2, Oxford University Press, 1994.

[Wal94b] C. Walther. On Proving the Termination of Algorithms by Machine. *Artificial Intelligence*, 71(1):101-157, 1994.