Technical Report 95-28, Mathematical Sciences Institute, Cornell University (1995)

A Theory of Stabilizing Algebraic Algorithms

Kiyoshi Shirayanagi*
NTT Communication Science Laboratories

Moss Sweedler
ACSyAM, Mathematical Sciences Institute, Cornell University

Abstract

The paper is about stabilizing algorithms. To capture the flavor in this abstract, we describe key results and techniques without the accompanying hypotheses. The first result is that given an algorithm $\mathcal{A}$ one can automatically generate a sequence of algorithms $\{\mathcal{A}_i\}_i$ which converges to $\mathcal{A}$. Moreover $\{\mathcal{A}_i\}_i$ stabilizes $\mathcal{A}$ in the sense that if the input I is approximated by a sequence of inputs $\{I_j\}_j$ then $lim_i\ lim_j\ \mathcal{A}_i(I_j) = \mathcal{A}(I)$. The key underlying technique is to enlarge the data set on which $\mathcal{A}$ operates. This is done by exchanging coefficients in the original data set with *pairs* consisting of a coefficient and an error bound. There are preselected coefficient values which include the singularities of predicates in the algorithm. When a coefficient is within the error bound of a preselected value, it is rewritten to the preselected value. This *rewriting* preserves convergence and moderates the effect of the singularities of the predicates in $\mathcal{A}$.

*Part of this research was carried out while the author stayed at the ACSyAM branch of the Mathematical Sciences Institute of Cornell University.

Contents

1 Introduction

This paper is about stabilizing algorithms. An ambitious activity. And as with every elaborate enterprise, there are various viewpoints from which to explain the ensuing endeavor. This introduction provides the opportunity to present both the problem and the payoff from several of the various viewpoints.

The easiest[1] viewpoint from which to describe stability is the mathematical viewpoint.

Mathematical Overview

From the mathematical view, think of an algorithm as being like a mathematical function. An algorithm is stable at a point if it is continuous at that point. Some functions or algorithms which are not continuous are nevertheless the limit of a sequence of functions or algorithms which **are** continuous. This is a satisfactory notion of stabilization but not always achievable. For example, *IS_ZERO?* is the one-line algorithm or function defined on the real numbers and returns "TRUE" for the input 0 and "FALSE" when the input is any other real number. *IS_ZERO?* cannot be the limit of continuous functions.[2] One approach to stabilizing *IS_ZERO?* is to embed {"TRUE","FALSE"} in the unit interval where "TRUE" is 1 and "FALSE" is 0. Then *IS_ZERO?* can be approximated by continuous functions to the unit interval.[3] Our methods avoid this approach because we seek to develop techniques which are more universally and automatically applicable and will handle many other algebraic structures and topologies. Hence, in general, we do not change the range. By our method of stabilization *IS_ZERO?* could[4] be approximated by the sequence of algorithms/functions $\{g_n\}_n$ where $g_n(x) =$ "FALSE" when $|x| > 1/n$ and $g_n(x) =$ "TRUE" when $|x| \leq 1/n$. The sequence of functions $\{g_n\}_n$ converges pointwise to *IS_ZERO?*. While the functions g_n are not continuous, the sequence $\{g_n\}_n$ has the following stability property: for **any** convergent sequence $\{r_k\}_k$ of real numbers converging to r, and r need not be 0:

$$lim_i \ (\ lim_j \ g_i(r_j) \ ) \ = g(r)$$

This suggests that $g(r)$ might be approximated as a limit of a sequence like $\{g_k(r_k)\}_k$.[5] If rather than dealing with **any** convergent sequence $\{r_k\}_k$, we are allowed to pick a sequence of r_n's compatible with the specific sequence $\{g_n\}_n$, then in fact $\{g_k(r_k)\}_k$ converges to

[1] The mathematical viewpoint is easiest to describe because it is so formal.

[2] *IS_ZERO?* is not continuous if the set {"TRUE","FALSE"} has the discrete topology. In this case only the (two) constant functions from the reals to {"TRUE","FALSE"} are continuous. *IS_ZERO?* is not the limit of constant functions.

[3] For example let $f_n(x) = 0$ when $|x| > 1/n$ and $f_n(x) = 1 - n|x|$ when $|x| \leq 1/n$. The sequence of functions $\{f_n\}_n$ converges pointwise to *IS_ZERO?*.

[4] Our methods do not necessarily yield a unique sequence of algorithms/functions approximating the original algorithm.

[5] If the sequence $\{r_k\}_k$ converges *too slowly*, it may be necessary to use $\{g_k(r_{k^2})\}_k$ or $\{g_k(r_{\iota(k)})\}_k$ instead of $\{g_k(r_k)\}_k$ for a suitable function $\iota : \boldsymbol{Z} \to \boldsymbol{Z}$.

$g(r)$. This in fact is what is done in Theorem (5.4.11), where of course we are treating more general algorithms than just *IS_ZERO?*. The *win* here is that even when the original algorithm involves exact computation, approximating algorithms may be selected which only require fixed precision approximate computation. g_k requires higher precision approximate arithmetic as k increases, of course. A key example is the general use of floating point computation to compute algorithms involving exact computation. The paper [6] uses floating point computation to compute Gröbner bases. Many key ideas in this paper are the continued development of ideas in [6].

Practical Overview

Since the paper is about stabilizing algorithms, we begin by giving our definition[6] of an algorithm. Basically, an algorithm has an initialization line which accepts input data and then continues with numbered instruction steps. There are only four kinds of instructions: "local variable assignment based on the result of computation", "conditional goto based on the result of predicate evaluation", "absolute goto", and "stop with the dissemination of output data". While this is an adequate formal structure to express algorithms, it is rather spare. On the other hand, this simplicity is what enables us to effectively reason about algorithms.

Another part of our specification of algorithms concerns data sets. I.e. the data which may be input, output and manipulated by an algorithm. Roughly speaking our data sets are based upon an algebraic structure R – typically a ring – and a set of variables X. Data sets themselves consist of certain amalgams based on R and X. For a data element, the entries from R are the *coefficients*. If the entries from R in a data element are all removed, what remains is the *structure* of the data element. The treatment of data is particularly important because it directly ties in with the formulation of stability. Namely, stability as treated in this paper, pertains to stability with respect to variance of the coefficients of data. Along this line we introduce structured functions (2.3.2). These are functions which vary as a function of the coefficients of data. Polynomial functions are structured.

As we said, stability is with respect to variance of coefficients. But what does it mean for coefficients to vary smoothly or for a sequence of coefficients to approach a given coefficient? To deal with this we introduce a quasi metric structure in Section (3.1) and give many examples to show that this encompasses many of the types of continuity typically considered in algebra. For example, this includes the reals, general valuations, and more. This leads naturally to the definition of convergence and bounds in Section (3.2). An important kind of bound is that of "convergent bound" which appears in Section (4.2). A convergent bound is a kind of oracle or measure of continuity for a

[6]Here and throughout this "Practical Overview" we freely omit technical details when they would slow-down the story telling. I.e. we try to cover many of the most important ideas and results and give an indication how they fit together. But we don't go into the detailed hypotheses of the results. That is done in the body of the paper.

function. Functions with convergent bounds are automatically continuous (4.6.4) and having a convergent bound might be considered the *constructive* way to specify continuity. Polynomial functions have convergent bounds. Having convergent bounds is the main requirement on functions appearing in algorithms we can stabilize.

Bracket coefficients are one of the key constructions we introduce to stabilize algorithms. Bracket coefficients might be thought of as a kind of algebraicization of interval arithmetic. They consist of a pair, the first component being an element from R or R^M or the general data set. The second component element is an error measure. Functions with convergent bounds may be extended to functions on the bracket coefficients. This, together with appropriate treatment for predicates allows us to extend algorithms which manipulate data from our original data set to algorithms which manipulate bracket coefficients in the original data set.[7] Here it is easier to obtain stabilization. In fact, for suitable non-stable algorithms the extension to bracket coefficients is stable (5.1.2). Typically, if the original algorithm is denoted "$\mathcal{A}$" then the extended algorithm will be denoted "$\mathcal{BC}(\mathcal{A})$".

Stabilizing algorithms entails handling functions and predicates. Up to here we have primarily discussed functions. Predicates map the data set to the set of two elements: {"TRUE","FALSE"}. Hence most predicates will have points where they are discontinuous. The main restriction on predicates in algorithms we can stabilize is that the discontinuity points of predicates lie in discrete sets. The way in which we stabilize algorithms by using bracket coefficients is to do what we call rewriting immediately preceding evaluating predicates. As we said, the first element of a bracket coefficient – call it E – is from the original data set and the second element is an error bound – call it ϵ. If E lies within distance ϵ of a point of discontinuity of the predicate, then E is replaced by the point of discontinuity and ϵ is replaced by $\hat{0}$, which means error bound zero. This is rewriting and the way to handle predicates to achieve stability. Once rewriting is done, the predicate is then evaluated on the first component of the bracket coefficient. The continuity and stabilization properties of rewriting are treated at some length. The key property regarding rewriting is that it preserves convergence and converts some convergence into *fast* convergence. If a sequence is approaching one of the discontinuities of the predicate, rewriting causes the sequence to *get there fast*, i.e. after a finite number of steps. Hence – after a finite number of steps – the predicate will actually be evaluated at the discontinuity point. Without rewriting, the predicate may never have been evaluated at the discontinuity point and since it is discontinuous there, the algorithm may never have the same behavior. Although we have not spelled it out in great detail, *IS_ZERO?* exhibits this problem.

As mentioned above, $\mathcal{BC}(\mathcal{A})$ denotes the extension of the algorithm $\mathcal{A}$ to an algorithm

[7]This does not change the algorithm. In this sense we give the original algorithm a kind of polymorphism and so the original algorithm applies to this other data set/structure. Thanks to Dexter Kozen for pointing out this interpretation.

which deals with bracket coefficients and (5.1.2) is the statement of $\mathcal{BC}(\mathcal{A})$ stability. (5.1.2) has important corollaries. The first (5.1.4) says that if you coerce the output of $\mathcal{BC}(\mathcal{A})$ from bracket coefficients back to the original data set, then you still have stability. $\mathcal{BC}(\mathcal{A})_R$ is used to denote $\mathcal{BC}(\mathcal{A})$ with its output coerced back to the original data set. The second corollary (5.1.6) utilizes a method of coercing data from the original data set to bracket coefficient data. If e lies in the original data set then for positive integers k we get E_k lying in the bracket coefficient set. The E_k's form a convergent sequence. Assuming this gives appropriate input data, we apply $\mathcal{BC}(\mathcal{A})_R$ to the E_k's. (5.1.6) says that the resulting sequence converges to the output obtained by invoking $\mathcal{A}$ on input data e. In fact, one can achieve finite convergence at selected output coefficient values. For example, one could insure that terms in a polynomial whose coefficients are converging to zero, reach zero in a finite number of steps.[8] This was done for the Buchberger algorithm in [6] and so the *shape*[9] of the resulting Gröbner basis is determined after a finite number of steps. Theorem (5.4.11) shows that the preceding can also be accomplished with approximate computation.

Quantified Convergence

Let us mention a meta lesson learned in the course of writing this paper. The lesson is that when convergence is required of $function_1$ or of a sequence, the convergence should be quantified or bounded by $function_2$ which itself is guaranteed or assumed to have a convergence property. Three examples of this are: convergent approximation (3.1.10), convergent bound (4.2.1) and approximate computation (5.4.1). What is interesting about this is that the process stops after one step and it is not necessary to have $function_3$ which bounds the convergence of $function_2$, where $function_3$ is guaranteed or assumed to have a convergence property, etc.

Open Questions and Further Work

Before the study of complexity, scientists simply considered whether a problem had an algorithmic solution. It was only later that we began asking for more quantitative information, namely to quantify the complexity of an algorithm and the minimal complexity of a problem. (5.1.4), (5.1.6) and (5.4.10) all refer to S_{OV}-convergence. This is a type of convergence where if the convergence is to an element in the set S_{OV}, then the sequence reaches the element in a finite number of steps. Suppose we are working with polynomials and that zero lies in S_{OV}. Further, suppose that we have *output polynomials* which S_{OV}-converge to a specific polynomial of interest. Then after a finite number of steps we reach output polynomials which have the same shape as the desired polynomial. I.e. we reach a stage where the output polynomials have precisely the same non-zero terms as

[8]This is a strong result. It is also frustrating, because we have no bound on when 0 must be achieved. On the other hand, at first scientists are happy to be able to tell if a problem has an algorithmic solution and then they begin to study the complexity of the problem.

[9]I.e. which monomials are present in the various terms of the resulting Gröbner basis.

the polynomial of interest, although the coefficients will not necessarily be the same. At this point, it might be feasible to determine the correct coefficients by another method. Perhaps by a method using linear algebra. The problem is that we have no bound for how many steps it takes for the output polynomials to become the correct shape. It is a qualitative result that this happens but we cannot quantify the number of steps required. Of course, the number of steps will depend upon the algorithm under study. Also – as with complexity – there is likely to be average case behaviour, worst case behaviour, etc. We hope others begin to consider this issue and prove some bounds.

(5.1.2) insures that under certain circumstances convergence is preserved. It is natural to wonder whether spurious convergence may occur. For example, might an algorithm fail to terminate normally for some specific input but applied to a sequence converging to the specific input the algorithm converges normally and yields a convergent sequence of outputs. (5.6.1) shows that spurious convergence may indeed occur. It is natural to ask what additional hypotheses on an algorithm or the converging input sequences or the convergent bounds used to stabilize the algorithm will insure that spurious convergence cannot occur. The same applies for (5.1.6) and (5.1.4).

The work in this paper shows that it would be feasible to develop a computer algebra system which gives the user the option to (automatically) stably approximating algorithms. Such a system would *be aware* of the singularities of its predicates and would be able to automatically assemble the appropriate S-sets for rewriting a user's program.

Concluding Remarks

We freely mention [6], the parent of this paper. However, one of the main ideas behind this paper is much older and we do not know an original reference. Namely the general idea which in over simplified form is:
1. An algorithm involves data with coefficients from a set A.
2. There is a way to switch to data with coefficients from a set B and it is easier to compute with these new coefficients.
3. After the computation is done, switch back to data with coefficients from A.
In this spirit let us mention Winkler's *lucky primes*, ([8]). It is related to our modular examples.

2 Fundamentals

2.1 Algorithms

This paper is about stabilizing algorithms which can be expressed in the following general form.

Initialize:	*Local assignment of input_data*
step_1:	*instruction*
step_2:	*instruction*
.	.
.	.
.	.
step_n:	*instruction*

input_data will be drawn from a set $\mathcal{S}(R; X)$ to be described later. The sets $\mathcal{S}(R; X)$ will be specific to the area of the algorithm. For example an algorithm to compute Gröbner bases would generally have a different $\mathcal{S}(R; X)$ from an algorithm to perform fast matrix multiplication. Our algorithms have internal, local variables and **Initialize** should be thought of as assigning the *input_data* to the internal, local variables. Here are three samples of an **Initialize** line.

- Initialize: $(StartingYear, StartingMonth, StartingDay)$
- Initialize: (X, Y, Z_1, Z_2)
- Initialize: ()

In the last case, no initialization is performed. The variables whose names happen to appear on the initialize line are assigned the *input_data* at the time an algorithm is invoked. In this sense an algorithm is somewhat like a function in that an algorithm *NumberOfMondaysSince* with an **Initialize** line of

- Initialize: $(StartingYear, StartingMonth, StartingDay)$

would be invoked

$$NumberOfMondaysSince\ (1823, 5, 17)$$

And if it were so invoked, it would begin with $StartingYear = 1823$, $StartingMonth = 5$, $StartingDay = 17$.

instructions come in four flavors:

- "stop with *Assignment of output_data*"
- "goto"
- "conditional goto"
- "local assignment from computation"

A "stop" instruction line looks much an **Initialize** line. Here are three samples how "stop" instructions might look, omitting the step number of the "stop" instruction line itself:

- "stop $(NumberOfMondays)$"
- "stop $(X, Z, 17, X)$"
- "stop ()"

What appears in the parentheses, if anything, are called "arguments" and are internal, local variables or elements of $\mathcal{S}(R; X)$. Below, we describe how a "stop" instruction and the other three types of instructions are executed.

Here is a sample how "goto" instructions might look, omitting the step number of the "goto" instruction line itself:

- "goto step_7"

Here is a sample how "conditional goto" instructions might look, omitting the step number of the "conditional goto" instruction line itself:

- "goto step_7 if GreaterThan(X,Y)"

What appears in the parentheses of the predicate[10], if anything, are called "arguments" and are internal, local variables or elements of $\mathcal{S}(R; X)$.

Here are samples how "local assignment from computation" instructions might look, omitting the step number of the "local assignment from computation" instruction line itself:

- "$X = Square(Y)$"
- "$X = Function(X, Y, Z)$"
- "$X = Y$"

Here $Square$ and $Function$ are functions. Later we describe the criteria for functions to be admissible. As usual for functions, what appears in the parentheses, if anything, are called "arguments". Just as for predicates, arguments here are internal, local variables or elements of $\mathcal{S}(R; X)$. For consistency, in a simple assignment, such as the example

[10]In this example the "predicate" is **GreaterThan()** and the arguments are **X** and **Y**.

$X = Y$, Y is called an "argument" and must be an internal, local variable or elements of $\mathcal{S}(R; X)$.

An algorithm is executed as shown in the following table. We are assuming that it began with the assignment of the initial data to any internal, local variables on the **Initialize** line and continued at step_1. Suppose now that we are at step_i. Also, assume that any internal, local variables which are arguments have been assigned values from $\mathcal{S}(R; X)$.

at step_i: "stop $(X_1, \ldots, X_n)$"	Execution halts. The algorithm outputs the sequence consisting of the values of the X_i's.
at step_i: "goto step_j"	Local variable assignments are unchanged. Continue at step_j.
at step_i: "goto step_j if $Predicate(X_1, \ldots, X_n)$"	Local variable assignments are unchanged. "$Predicate(X_1, \ldots, X_n)$" is evaluated. If: TRUE continue at step_j. FALSE continue at step_$(i+1)$.
at step_i: "$X = Function(Y_1, \ldots, Y_n)$"	$Function(Y_1, \ldots, Y_n)$ is evaluated. The result is assigned to X. Continue at step_$(i+1)$.

The algorithm together with the *input_data* is said to *terminate normally* if a "stop $(X_1, \ldots, X_n)$" instruction is reached. In this case, the value(s) of $(X_1, \ldots, X_n)$ is said to be the *output* of invoking the algorithm with the given *input_data*. An algorithm together with the *input_data* is said to *run forever* if it keeps performing steps and never reaches a stop instruction. A so-called *infinite loop*, runs forever. An algorithm together with the *input_data* is said to *crash* if it neither terminates normally nor runs forever. This could be caused by a "Function" or "Predicate" evaluating on unsuitable data or if a "goto" or "conditional goto" specifies a step number which does not exist, or if a "stop", "conditional goto" or "local assignment from computation" step is reached, where some of the arguments have not been assigned values, etc.

When we speak of an algorithm's internal, local variables we also mean that these variables names are distinct from the values of data in $\mathcal{S}(R; X)$ so there can be no ambiguity whether an argument is meant to be a value assigned to a variable or a specified element of $\mathcal{S}(R; X)$.

Let us write the output of invoking an algorithm $\mathcal{A}$ with the given *input_data* I, as $\mathcal{A}(I)$.[11] The common meaning of deterministic algorithm refers to those which can be expressed in the specified form. Looping structures such as the "While", "For", and "Repeat" loops, and branching structures such as the "If-then-else" statement can be expressed with our *instructions*.

2.2 Data Sets, $\mathcal{S}(R; X)$

Now we build an admissible data set from which *input_data* and other data values are drawn. To begin let R and X be disjoint sets of formal symbols. The elements of R will be used for the purpose of approximation, a key ingredient in our stabilization technique. The elements of X will be used to account for structural elements which do not vary in approximations. For example, X might be a specific, fixed basis for a vector space and R might be (a copy of) the coefficient field, perhaps the rationals, reals or complex numbers. This would be used for stabilizing algorithms by way of approximation in the coefficient field. The elements of R are referred to as *coefficients* and the elements of X are referred to as *atoms*.

In many examples R will arise from a ring, or other algebraic structure, where there is one symbol of R for each element of the ring. In such cases the ring element may be identified with the R symbol, i.e. for a ring element r we may simply write r instead of $Symbol_r$, when considering it as an element of R. In this case R is not considered to be a ring. It is still just a set of formal symbols. However, the ring operations would give certain maps among R and its Cartesian products. For example, negation gives the map from R to itself sending r to $-r$. Addition and multiplication give maps from $R \times R$ to R etc.

Consider the disjoint union: $R \uplus X$. The elements of $R \uplus X$ are the *indecomposable data*. The final step in specifying admissible values for *input_data* and *transition_data* is to construct $\mathcal{S}(R; X)$, the set of sequences of indecomposable data. More precisely:

1. The empty sequence: "()" is in $\mathcal{S}(R; X)$.
2. If $d_1, \ldots, d_n \in R \uplus X$, then $(d_1, \ldots, d_n) \in \mathcal{S}(R; X)$. The d_i's are not assumed to be distinct.

By a sequence "$(s_1, \ldots, s_n)$" we mean the ordered tuple of $s_1, \ldots, s_n$. When writing a sequence of indecomposable elements we write the elements surrounded by outer parentheses. At times it may be convenient to also have internal parentheses. For example,

[11] $\mathcal{A}(I)$ is like functional notion. It is like $\mathcal{A}$ being invoked or evaluated on I. If $\mathcal{A}$ does not terminate normally with the input data I, then $\mathcal{A}(I)$ is similar to functional notation where one may write the expression: "$f(a)$" although a may not be in the domain of f. In neither case is a value returned.

when discussing a realization of a free module, we talk about sequences of pairs, such as: $((r'_1, x'_1), \ldots, (r'_m, x'_m))$. In this case consider X to also contain symbols: "(", and ")" and that we have a convention of omitting commas around the internal parentheses. When we discuss sequences of sequences in $\mathcal{S}(R; X)$, please just assume that we have "(", and ")" in X, even if we have not explicitly mentioned the fact. If "(","$)$"$\in X$ then $\mathcal{S}(R; X)$ contains sequences with unbalanced internal parentheses. However, independently of "(", and ")", $\mathcal{S}(R; X)$ contains a lot of garbage that we do not care about. Part of *the game* is to ignore the garbage by selecting useful subsets of $\mathcal{S}(R; X)$ from which *input_data* and *transition_data* are drawn. The set $T \subset \mathcal{S}(R; X)$, in the example below of a free module, is such a useful subset.

The definition of $\mathcal{S}(R; X)$ only depends on $R \uplus X$ as a set. I.e. we could have defined $\mathcal{S}(D)$ where D is any set. However, R and X play distinctly different roles in what follows and they induce specific functions on $\mathcal{S}(R; X)$. That is why the notation "$\mathcal{S}(R; X)$" preserves the individuality of R and X.

There are concepts of *structure* and *coefficients* for elements of $\mathcal{S}(R; X)$. These concepts will allow us to vary *coefficient* entries of data while preserving the *structural integrity*, which is one of the fundamental techniques in this paper.

For $s \in \mathcal{S}(R; X)$ let $\sharp s$ be the number of entries from R which appear in s. Duplicates must be counted with their multiplicity. For example, $\sharp(r, x, r, y, r) = 3 = \sharp(r, (x, (r, (y, r))))$ if $r \in R$ and $x, y \in X$. (Of course "(","$)$"$\in X$ also, but we generally do not explicitly mention this.) Another example, $\sharp(x, (y, z)) = 0$. Let $*$ be an element which is not in X and form $\mathcal{S}(\{*\}; X)$. There is a natural map, also denoted $*$, from $\mathcal{S}(R; X)$ to $\mathcal{S}(\{*\}; X)$, namely, replace each element of R by $*$. For example, $*(p, x, q, y, r) = (*, x, *, y, *)$ and $*(p, (x, (q, (y, r)))) = (*, (x, (*, (y, *))))$ if $p, q, r \in R$ and $x, y \in X$. This map satisfies: for any $s \in \mathcal{S}(R; X)$: $\sharp s = \sharp * s$. The map $*$ should be thought of as *isolating* the *structure* of an element by means of removing the specific *coefficients*, i.e. elements of R.

Coefficients are captured by maps to R^i for various i.[12] Let $\mathcal{S}(R; X)_i$ be the elements s of $\mathcal{S}(R; X)$ with $\sharp s$ equal to i. $\kappa : \mathcal{S}(R; X)_i \to R^i$ is now defined. For $s \in \mathcal{S}(R; X)_i$, $\kappa(s)$ is obtained by deleting all elements of X (including any internal parentheses) and preserving the order of what remains. For example, $\kappa(p, x, q, y, r) = (p, q, r) = \kappa(p, (y, (q, (x, r))))$ and $\kappa(x, y, z) = () = \kappa(z, (x, y))$. Elements of $\mathcal{S}(R; X)$ may be recovered from their *structure* and *coefficients*. For each natural number i there is a map $\langle , \rangle : \mathcal{S}(\{*\}; X)_i \times R^i \to \mathcal{S}(R; X)_i$. $\langle u, (r_1, \ldots, r_i) \rangle$ is defined by replacing the first, from left to right, $*$ in u by r_1, then the next $*$ in u by r_2, etc. After i steps one has replaced all the $*$'s in u by $r_1, \ldots, r_i$. For example, $\langle (*, (x, (*, (y, *)))), (p, q, r) \rangle = (p, (x, (q, (y, r))))$ and $\langle (x, (y, z)), () \rangle = (x, (y, z))$. Although obvious, it is worth noting that: $*\langle u, (r_1, \ldots, r_i) \rangle =$

[12]For positive integers i, $R^i = R \times \cdots \times R$, i-times. R^0 is identified with the set of one element $\{()\}$ where that element is an empty sequence. So, R^0 has cardinality one, not zero.

u and $\kappa\langle u, (r_1, \ldots, r_i)\rangle = (r_1, \ldots, r_i)$.

The precise statement that elements of $\mathcal{S}(R; X)$ may be recovered from their *structure* and *coefficients* is the equation: $s = \langle *s, \kappa(s)\rangle$ which holds for all $s \in \mathcal{S}(R; X)$. The equation is most easily understood in the terms: "if one strips the elements of R out of s and then puts them back in the same positions, one recovers s."

We have shown that $\mathcal{S}(R; X)$ is in one-to-one correspondence with the disjoint union of $\mathcal{S}(\{*\}; X)_i \times R^i$ for $i = 0, 1, 2, \ldots$. This underlies all that follows. For example, the definition a bit later of *structured function* depends upon this decomposition and could be formulated for other data structures than $\mathcal{S}(R; X)$ which are the disjoint union of copies of R^i *times* other sets, where *times* means Cartesian product.

The last bit of notation in this section is a second interpretation of $\mathcal{S}(\{*\}; X)$ or $\mathcal{S}(\{*\}; X)_i$. Namely, one may view elements of $\mathcal{S}(\{*\}; X)_i$ as functions from R^i to $\mathcal{S}(R; X)_i$. More specifically:

2.2.1 ***NOTATION for $\mathcal{S}(\{*\}; X)$ as Functions:*** *For $u \in \mathcal{S}(\{*\}; X)_i$ we consider:*

$$\overline{u} : R^i \to \mathcal{S}(R; X)_i, \quad \underline{r} \to \langle u, \underline{r}\rangle$$

For $u \in \mathcal{S}(\{*\}; X)_i$, we typically overbar u when considering u as a function from R^i to $\mathcal{S}(R; X)_i$.

Eliminating "$\langle, \rangle$" from the notation considerably simplifies later equations.

2.3 Structured Functions and Algorithms

Functions between subsets of $\mathcal{S}(R; X)$ are among the main objects of study in this paper because they are among the main things we must stabilize. An important concept is that of a function being *structured*. In words, this means that the structure of the image of an element varies as a function of the structure of the element. This implies that on elements of the same structure, the coefficients of the images of elements vary as a function of the coefficients of the elements. Structured functions – and related concepts – help distinguish the algorithms of interest.

2.3.1 ***DEFINITION Structured Algorithms:***

An algorithm is called ***structured*** *if it satisfies the following:*

($\mathcal{A}$1) *input_data* $\in \mathcal{S}(R; X)$.

($\mathcal{A}$2) *Every "function" is structured.*

(𝓐3) *Every "predicate" is a function from a subset of $\mathcal{S}(R;X)$ to the set with two elements: $\{TRUE, FALSE\}$.*

2.3.2 *DEFINITION Structured Function:* *Let $S, T \subseteq \mathcal{S}(R;X)$ and $\varphi : S \to T$. φ is a structured function if for all $s_1, s_2 \in S$:*

$$*\varphi(s_1) = *\varphi(s_2) \text{ if } *s_1 = *s_2$$

Note that in the definition of *structured function*, either or both of S and T may be empty or may be equal to $\mathcal{S}(R;X)$ itself.[13] Let $*S \subset \mathcal{S}(\{*\};X)$ be defined as: $\{b \in \mathcal{S}(\{*\};X) |$ there exists $s \in S$ with $b = *s\}$. In other words, $*S$ is the image of S under the map: $* : \mathcal{S}(R;X) \to \mathcal{S}(\{*\};X)$. If $\{\}$ is the empty set then $\mathcal{S}(R;\{\})$ is simply the disjoint union: $R^0 \uplus R^1 \uplus R^2 \uplus \cdots$. We use the simpler notation $R^\star$ for this set. Note that $\sharp$ on $R^\star$ is simply the constant n on R^n.

Structured functions are characterized as follows:

2.3.3 *LEMMA:* *φ is structured if and only if there are functions $Struc : *S \to *T$ and $\gamma : S \to R^\star$ where for all $s \in S$: $\sharp Struc(*s) = \sharp\gamma(s)$ and*

2.3.4 $$\varphi(s) = \langle Struc(*s), \gamma(s)\rangle$$

In this case: Struc and γ are uniquely determined by φ. γ is simply the composite $\kappa\varphi$.

PROOF: Assume *Struc* and γ are maps with the stated properties. Apply $*$ to both sides of (2.3.4) giving: $*\varphi(s) = Struc(*s)$. Hence $*\varphi(s_1) = *\varphi(s_2)$ if $*s_1 = *s_2$ and φ is *structured*.

Conversely suppose that φ is structured. Then $s \to *\varphi(s)$ only depends on $*s$ and so there is a unique function: $Struc : *S \to \mathcal{S}(\{*\};X)$ where $*\varphi(s) = Struc(*s)$. Then (2.3.4) follows, with the composite $\kappa\varphi$ used for γ, from the fact that $\varphi(s) = \langle *\varphi(s), \kappa\varphi(s)\rangle$ after substituting $Struc(*s)$ for $*\varphi(s)$.

The remaining details are left to the reader. ∎

While γ is simply the composite $\kappa\varphi$, *Struc* does not generally have such a simple expression.

2.3.5 *NOTATION for STRUCTURE MAP of φ:* *Let "$Struc_\varphi$" indicate the "Struc" map corresponding to the structured function φ. When the meaning is clear from the context, we simply use "φ" itself to stand for "$Struc_\varphi$".*

[13] There are not many structured functions from S to T when S or T are empty. However, allowing these degenerate cases keeps the definition simple and obviates special cases in later arguments.

The meaning generally **is** clear from the context because the domain of the original structured function φ is a subset of $\mathcal{S}(R;X)$ and the domain of "φ" standing for "$Struc_\varphi$" is a subset of $\mathcal{S}(\{*\};X)$. Thus the *argument* "arg" in "$\varphi(arg)$" indicates which meaning of "φ" is relevant. (2.4.2) illustrates the two usages of "φ" in close proximity. Where "$Struc_\varphi$" occurs without an argument, we typically must specify "$Struc_\varphi$" and cannot simply use "φ". This is illustrated in (2.4.3).

As mentioned preceding Definition (2.3.2), if a structured function $\varphi : S \to T$ is restricted to elements of the same structure, the coefficients of the images of elements vary as a function of the coefficients of the elements. These maps are defined in (2.4.2.d) below. It is precisely these functions[14] which must be controlled to achieve stabilization. These considerations get rather technical and are quarantined to the next section which we recommend skipping until the results are called upon.

2.4 Technical Results about Structured Functions

We shall see how structured functions are themselves structured. I.e. how they are put together from component parts.

2.4.1 ***DEFINITION:*** *Let $S \subseteq \mathcal{S}(R;X)$ and let $z \in \mathcal{S}(\{*\};X)$.*

a. S_z denotes $\{s \in S | *s = z\}$. In other words, S_z is the set of elements of S which have the structure z.

b. κS_z denotes $\kappa(S_z)$. In other words, κS_z is the set of coefficients of the elements in S_z.

S_z or κS_z may be empty. They are **not** empty if and only if $z \in *S$. Frequently, $z = *s_0$ for $s_0 \in S$ and we write: S_{*s_0} and κS_{*s_0}. S_{*s_0} and κS_{*s_0} are never empty; specifically, $s_0 \in S_{*s_0}$ and $\kappa(s_0) \in \kappa S_{*s_0}$. Assertion (a) in the following should be thought of as saying that S is partitioned into *slices* S_z.[15] Assertion (b) says that elements lying in the same slice are uniquely determined by their coefficients. Assertion (g) shows how a structured function restricted to a *slice* may be expressed in terms of the coefficients of elements of the slice.

2.4.2 ***LEMMA-DEFINITION:*** *Let $S, T \subseteq \mathcal{S}(R;X)$, $\varphi : S \to T$ a structured function and $z \in *S$.*

a. $S = \uplus_{z \in *S} S_z$, where $\uplus$ denotes "disjoint union".

[14] While these are the functions which must be controlled, predicates must be handled as well.

[15] These are the *slices* in (4.5.10) which is said to proceed *slice-by-slice*.

b. The map $\overline{z}$ arising from z, as defined at (2.2.1), is inverse to κ_z defined:

$$\kappa_z : S_z \to \kappa S_z, s \to \kappa(s)$$

c. $\varphi(S_z) \subset T_{\varphi(z)}$.

d. ***DEFINE:*** $\kappa\varphi\overline{z} : \kappa S_z \to \kappa T_{\varphi(z)}$ *as the composite of the three maps:*

$$\kappa S_z \overset{\overline{z}}{\to} S_z \overset{\varphi|_{S_z}}{\to} T_{\varphi(z)} \overset{\kappa_{\varphi(z)}}{\to} \kappa T_{\varphi(z)}$$

e. If $u \in \kappa S_z$ and $q \in S_z$ with $\kappa(q) = u$ then q is the unique element $p \in S_z$ with $\kappa(p) = u$. In addition, $\kappa\varphi\overline{z}(u) = \kappa\varphi(q)$.

f. The following two composites are equal:

$$\kappa S_z \overset{\overline{z}}{\to} S_z \overset{\varphi|_{S_z}}{\to} T_{\varphi(z)} \text{ and } \kappa S_z \overset{\kappa\varphi\overline{z}}{\to} \kappa T_{\varphi(z)} \overset{\overline{\varphi(z)}}{\to} T_{\varphi(z)}$$

or equivalently $\varphi|_{S_z} : S_z \to T_{\varphi(z)}$ equals the composite:

$$S_z \overset{\kappa_z}{\to} \kappa S_z \overset{\kappa\varphi\overline{z}}{\to} \kappa T_{\varphi(z)} \overset{\overline{\varphi(z)}}{\to} T_{\varphi(z)}$$

g. For $s \in S_z$,

$$\varphi(s) = \langle \varphi(z), \kappa\varphi\overline{z}(\kappa s) \rangle$$

and hence for any $t \in S$:

$$\varphi(t) = \langle \varphi(*t), \kappa\varphi\overline{*t}(\kappa t) \rangle$$

PROOF: The details are left to the reader. ∎

2.4.3 ***DEFINITION:*** *$Struc_\varphi$ (2.3.5) is called the structure function of φ and the $\kappa\varphi\overline{z}$'s are the coefficient functions of φ. We continue to use "φ" for "$Struc_\varphi$" when the meaning is clear from context.*

(2.4.2,g) shows how φ is *put together* or *structured* from $Struc_\varphi$ and the $\kappa\varphi\overline{z}$'s.

2.4.4 ***PROPOSITION Structure of Structured Functions:*** *Let $S, T \subseteq \mathcal{S}(R; X)$. Suppose that $Struc : *S \to *T$ and for each $z \in *S$ there is a map $Coef(z) : \kappa S_z \to \kappa T_{Struc(z)}$. Define the map $\varphi : S \to T$ by defining for each $z \in *S$ the restriction $\varphi|_{S_z} : S_z \to T_{Struc(z)}$ as the composite:*

$$S_z \overset{\kappa_z}{\to} \kappa S_z \overset{Coef(z)}{\to} \kappa T_{Struc(z)} \overset{\overline{Struc(z)}}{\to} T_{Struc(z)}$$

Then φ is a structured function, $Struc_\varphi = Struc$ and for each $z \in *S$, $\kappa\varphi\overline{z} = Coef(z)$.

PROOF: Again, the details are left to the reader. ∎

Stabilization of algorithms depends upon stabilization of predicates and structured functions. Stabilization of structured functions depends upon stabilization of their *coefficient functions*. Later we specify conditions which *coefficient functions* must satisfy for the purpose of stabilization. But first we tell how structured functions and their coefficient functions behave with respect to basic set theoretic constructions.

2.4.5 *LEMMA:* *Suppose that $A, B, C, D \subseteq \mathcal{S}(R; X)$ and that $\alpha : A \to C$, $\beta : B \to D$ are structured functions. Suppose that $a_0 \in A$ and $b_0 \in B$ so that $*a_0 \in *A$ and $*b_0 \in *B$.*

a. $\alpha\times\beta : A\times B \to C\times D$, $(a, b) \to (\alpha(a), \beta(b))$ is a structured function. $\kappa(A\times B)_{*(a_0,b_0)}$ coincides with $\kappa A_{*a_0} \times \kappa B_{*b_0}$ and $\kappa(\alpha \times \beta)\overline{*(a_0, b_0)} = \kappa\alpha\overline{*a_0} \times \kappa\beta\overline{*b_0}$.

b. If $D \subseteq A$ then the composite map: $\alpha\beta$ is structured and if $\beta(b_0) = a_0$ then $\kappa\alpha\beta\overline{*b_0}$ is the composite map: $\kappa\alpha\overline{*a_0} \circ \kappa\beta\overline{*b_0}$.

The following general set theoretic functions are structured functions.

c. Constant functions from A to C. If γ is the constant function sending A to $c \in C$, then $\kappa\gamma\overline{*a_0}$ is the constant function sending κA_{*a_0} to $\kappa(c)$.

d. Diagonal functions, $\Delta_\ell : A \to A^\ell$, for each non-negative integer ℓ, sending elements $a \in A$ to $(a, \ldots, a)$, ℓ-times. For $\ell = 0$ this is just the constant function to a one point set. For $\ell = 1$ this is just the identity function from A to itself. $\kappa{A^\ell}_{*(a_0,\ldots,a_0)}$ coincides with $\kappa A_{*a_0} \times \cdots \times \kappa A_{*a_0} = \kappa{A_{*a_0}}^\ell$ and $\kappa\Delta_\ell\overline{*a_0}$ is just the diagonal function: $\kappa A_{*a_0} \to \kappa{A_{*a_0}}^\ell$.

e. Projection functions such as: $\Pi_A : A\times B \to A$, $(a, b) \to a$. $\kappa(A\times B)_{*(a_0,b_0)}$ coincides with $\kappa A_{*a_0} \times \kappa B_{*b_0}$ and $\kappa\Pi_A\overline{*(a_0, b_0)}$ is the projection function

$$\Pi_{\kappa A_{*a_0}} : \kappa A_{*a_0} \times \kappa B_{*b_0} \to \kappa A_{*a_0}$$

Similarly for projection onto B.

f. Permutation functions such as $A^\ell \to A^\ell$, such as $(a_1, \ldots, a_\ell) \to (a_{e_1}, \ldots, a_{e_\ell})$ where $(e_1, \ldots, e_\ell)$ is a permutation of $(1, \ldots, \ell)$. Suppose that γ denotes this function and $a'_1, \ldots, a'_\ell \in A$. $\kappa{A^\ell}_{*(a'_1,\ldots,a'_\ell)}$ coincides with $\kappa A_{*a'_1} \times \cdots \times \kappa A_{*a'_\ell}$ and $\kappa\gamma\overline{*(a'_1, \ldots, a'_\ell)}$ is the map:

$$\kappa A_{*a'_1} \times \cdots \times \kappa A_{*a'_\ell} \to \kappa A_{*a'_{e_1}} \times \cdots \times \kappa A_{*a'_{e_\ell}}$$
$$(v_1, \ldots, v_\ell) \to (v_{e_1}, \ldots, v_{e_\ell})$$

g. For $c \in \mathcal{S}(R; X)$ the function $A \to A \times \{c\}$, $a \to (a, c)$. Suppose that γ denotes this function. $\kappa(A \times \{c\})_{*(a_0,c)}$ coincides with $\kappa A_{*a_0} \times \{\kappa(c)\}$ and

$$\kappa\gamma\overline{*a_0} : \kappa A_{*a_0} \to \kappa A_{*a_0} \times \{\kappa(c)\},\ v \to (v, \kappa(c))$$

PROOF: We outline the major ideas and leave most of the details to the reader.

First we verify the *structuredness*. (a), (e), (f) and (g) follow from the fact that $*(u,v) = (*u, *v)$ and (d) simply uses: $*(a,\ldots,a) = (*a,\ldots,*a)$. Alternatively for (g), it is (Identity $\times$ Constant) composed with Diagonal and by parts (a–d) this implies that (g) gives structured maps.

(b) Since β is *structured* if $b, b' \in B$ with $*b = *b'$ then $*\beta(b) = *\beta(b')$ and so $*\alpha\beta(b) = *\alpha\beta(b')$ because α is *structured*.

(c) If γ is a constant function then $\gamma(a) = \gamma(a')$ for $a, a' \in A$ so $*\gamma(a) = *\gamma(a')$.

Next we partly verify the claimed *coefficient functions*, and leave much to the reader.

Claims such as:

$$\kappa(A \times B)_{*(a_0,b_0)} \text{ coincides with } \kappa A_{*a_0} \times \kappa B_{*b_0}$$

found in (a), follow from the fact that $\kappa(u,v) = (\kappa(u), \kappa(v))$. (d), (e), (f) and (g) have similar claims which follow for similar reasons. Let us verify the rest of (a) and also (b).

Suppose that $v \in \kappa A_{*a_0}$. Let $a \in A_{*a_0}$ be the unique element where $\kappa(a) = v$. Then $\kappa\alpha\overline{*a_0}(v) = \kappa\alpha(a)$. Similarly if $w \in \kappa B_{*b_0}$ and $b \in B_{*b_0}$ is the unique element where $\kappa(b) = w$, then $\kappa\beta\overline{*b_0}(w) = \kappa\beta(b)$. Since (a,b) is the unique element $x \in A_{*a_0} \times B_{*b_0}$ where $\kappa(x) = (v,w)$ it follows that: $\kappa(\alpha \times \beta)\overline{*(a_0,b_0)}(v,w) = \kappa(\alpha \times \beta)(a,b) = (\kappa\alpha(a), \kappa\beta(b)) = (\kappa\alpha\overline{*a_0}(v), \kappa\beta\overline{*b_0}(w))$. In other words: $\kappa(\alpha \times \beta)\overline{*(a_0,b_0)} = \kappa\alpha\overline{*a_0} \times \kappa\beta\overline{*b_0}$ as asserted.

Here is (b). Suppose that $w \in \kappa B_{*b_0}$. Let $b \in B_{*b_0}$ be the unique element where $\kappa(b) = w$ so that $\kappa\beta\overline{*b_0}(w) = \kappa\beta(b)$ and $\kappa\alpha\beta\overline{*b_0}(w) = \kappa\alpha\beta(b)$. Since $b \in B_{*b_0}$ and β is structured, it follows that $\beta(b) \in D_{*\beta(b_0)}$ and $D_{*\beta(b_0)} \subseteq A_{*a_0}$ since $\beta(b_0) = a_0$. Hence, the image of $\kappa\beta\overline{*b_0}$ lies in the domain of $\kappa\alpha\overline{*a_0}$ and the composite is defined. Also, since $\kappa\beta(b) = \kappa\beta\overline{*b_0}(w)$, $\beta(b)$ must be the unique element $x \in A_{*a_0}$ where $\kappa(x) = \kappa\beta\overline{*b_0}(w)$. Thus $\kappa\alpha\overline{*a_0} \circ \kappa\beta\overline{*b_0}(w) = \kappa\alpha\beta(b)$ which we already observed equals $\kappa\alpha\beta\overline{*b_0}(w)$.

The rest is left to the reader. ∎

2.4.6 *DEFINITION Closed under Composition and Cartesian Construction:*

Let $\mathcal{E}$ be a set of structured functions mapping some subsets of $\mathcal{S}(R;X)$ to other subsets of $\mathcal{S}(R;X)$. $\mathcal{E}$ is called ***C4*** *if properties (a–g) hold. Here, $A, B, C, D \subseteq \mathcal{S}(R;X)$ and $\alpha : A \to C$, $\beta : B \to D$ are maps in $\mathcal{E}$.*

Let $\mathcal{F}$ be a set of functions mapping some subsets of R^m to R^n for various m's and n's. $\mathcal{F}$ is called ***C4*** *if properties (A–G) hold. Here, $A \subseteq R^m$, $C \subseteq R^n$, $B \subseteq R^{m'}$, $D \subseteq R^{n'}$ and $\alpha : A \to C$, $\beta : B \to D$ are maps in $\mathcal{F}$.*

a. The product map $\alpha \times \beta : A \times B \to C \times D$, $(a,b) \to (\alpha(a), \beta(b))$ is in $\mathcal{E}$.

b. If $D \subseteq A$ then the composite map: $\alpha\beta$ is in $\mathcal{E}$.

The following general set theoretic functions are in $\mathcal{E}$.

c. Constant functions from A to C.

d. Diagonal functions, $\Delta_\ell : A \to A^\ell$, for each non-negative integer ℓ, sending elements $a \in A$ to $(a, \ldots, a)$, ℓ-times.

e. Projection functions such as: $\Pi_A : A \times B \to A$, $(a, b) \to a$. And similarly for projection onto B.

f. Permutation functions such as $A^\ell \to A^\ell$, such as $(a_1, \ldots, a_\ell) \to (a_{e_1}, \ldots, a_{e_\ell})$ where $(e_1, \ldots, e_\ell)$ is a permutation of $(1, \ldots, \ell)$.

g. For $c \in \mathcal{S}(R; X)$ the function $A \to A \times \{c\}$, $a \to (a, c)$.

A. The product map $\alpha \times \beta : A \times B \to C \times D$, $(a, b) \to (\alpha(a), \beta(b))$ is in $\mathcal{F}$.

B. If $D \subseteq A$ then the composite map: $\alpha\beta$ is in $\mathcal{F}$.

The following general set theoretic functions are in $\mathcal{F}$:

C. Constant functions from A to C.

D. Diagonal functions, $\delta_\ell : A \to A^\ell$, for each non-negative integer ℓ, sending elements $a \in A$ to $(a, \ldots, a)$, ℓ-times.

E. Projection functions such as: $\pi_A : A \times B \to A$, $(a, b) \to a$. And similarly for projection onto B.

F. Permutation functions such as $A^\ell \to A^\ell$, such as $(a_1, \ldots, a_\ell) \to (a_{e_1}, \ldots, a_{e_\ell})$ where $(e_1, \ldots, e_\ell)$ is a permutation of $(1, \ldots, \ell)$.

G. For $c \in R^n$ the function $A \to A \times \{c\}$, $a \to (a, c)$.[16]

In both the $\mathcal{E}$ and $\mathcal{F}$ cases, (g) or (G) is given by (Identity $\times$ Constant) composed with Diagonal. By parts (a–d) it follows that (g) is in $\mathcal{E}$ or $\mathcal{F}$.

Notice that given a set $\mathcal{U}$ of structured functions mapping some subsets of $\mathcal{S}(R; X)$ to other subsets of $\mathcal{S}(R; X)$, one can build up or generate a smallest "C4" set containing $\mathcal{U}$. This set is denoted $C4(\mathcal{U})$. One begins with the set $\mathcal{U}$, adds in all maps of type (c–g) which are not there already, and then adds in all maps which can be expressed as iterated products and composites, so that (a) and (b) will be satisfied. Similarly, given a set $\mathcal{V}$ of functions mapping some subsets of R^m to R^n for various m's and n's one can build up or generate a smallest "C4" set containing $\mathcal{V}$. This set is denoted $C4(\mathcal{V})$. The process is similar to the construction of $C4(\mathcal{U})$ and in both cases the precise details are left to the reader.

[16] Consider $A \times \{c\} \subseteq A \times R^n$.

2.4.7 *PROPOSITION Closed under Composition and Cartesian Construction:* *Let $\mathcal{U}$ be a set of structured functions mapping some subsets of $\mathcal{S}(R;X)$ to other subsets of $\mathcal{S}(R;X)$. Let $\mathcal{V}$ be a set of functions mapping some subsets of R^m to R^n for various m's and n's. Assume that for each $A, C \subseteq \mathcal{S}(R;X)$ and map $\alpha : A \to C$ in $\mathcal{U}$ and element $z \in {}^*A$, the map $\kappa\alpha\overline{z}$ lies in $\mathcal{V}$. Then if $B, D \subseteq \mathcal{S}(R;X)$ and the map $\beta : B \to D$ lies in $C4(\mathcal{U})$, and $z \in {}^*B$ it follows that $\kappa\beta\overline{z} \in C4(\mathcal{V})$.*

PROOF: This follows from Lemma (2.4.5) ∎

Structured Algorithms are too large a class of algorithms to stabilize. The purpose of introducing *$\mathcal{F}$ Functioned Algorithms* is to specify a subclass of *structured algorithms* by putting restrictions on the coefficient functions which may appear. This is appropriate since our approach to stability depends upon stabilizing the coefficient functions and stabilizing the predicates.[17] The restriction we put on coefficient functions is to insist that they come from the specified class of functions: $\mathcal{F}$. Because this paper specializes in stabilizing algorithms in ring theory, coefficients will typically be drawn from a ring and the class $\mathcal{F}$ will frequently be specialized to polynomial functions.[18]

Suppose that $\mathcal{F}$ is a **C4** set of functions mapping some subsets of R^m to R^n for various m's and n's. Here is what it means for a structured algorithm to be based upon $\mathcal{F}$.

2.4.8 *DEFINITION $\mathcal{F}$ Functioned Algorithms:* *An algorithm $\mathcal{A}$ is an $\mathcal{F}$ Functioned Algorithm if*

- *$\mathcal{A}$ is a structured algorithm, (2.3.1).*
- *The coefficient functions (2.4.3) of $\mathcal{A}$ lie in $\mathcal{F}$. I.e. for every "local assignment from computation" step in $\mathcal{A}$:*

  $$\textit{at step\_i: ``}X = Function(Y_1, \ldots, Y_n)\textit{''}$$

  where $Function : S \to T$ for $S, T \subseteq \mathcal{S}(R;X)$ and element $z \in {}^*S$, the map $\kappa Function\overline{z} : \kappa S_z \to \kappa T_{Function(z)}$ lies in $\mathcal{F}$.

[17] We shall get control of the predicates in an extremely different fashion from how we get control of the coefficient functions. We shall introduce *discontinuity sets* of predicates and an entire mechanism of *bracket coefficients* and *bracket coefficient (term) rewriting* to deal with predicates. *Bracket coefficients* and *rewriting* also have another payoff, they permit the use of approximate computation to stably approximate exact computation.

[18] For some purposes, it may be convenient or necessary to allow additional functions which might include division, square root, differentiation, etc. As will be explained in Section (3) and Section (4.2), when introducing additional functions one must define corresponding suitable functions in a partially ordered set and in the set of bracket coefficients, so as to guarantee a good propagation of errors like Lemma (4.2.9) in Section (4.2).

We now present *Polynomial Functions*, which is a **C4** class of functions. Suppose that R is (a copy of) a ring and P maps (a subset of) R^m to (a subset of) R^n. Then P has component functions $(P_1, \ldots, P_n)$ where each P_i is a function of m variables.

2.4.9 ***DEFINITION Polynomial functions:*** *P is polynomial if all of the P_i's are polynomial*[19] *functions.*

2.4.10 ***DEFINITION $\mathcal{POLY}$:*** *$\mathcal{POLY}$ denotes the set of polynomial functions from subsets of R^m to R^n for various m's and n's.*

It is left for the reader to check that $\mathcal{POLY}$ is **C4**. The reader may also check that $\mathcal{POLY}$ is generated[20] by the polynomial functions from (subsets of) R^m to R for various m's.

2.4.11 ***DEFINITION Algebraic Algorithms:*** *A $\mathcal{POLY}$ Functioned Algorithm is called an Algebraic Algorithm.*

(2.4.7) has the following important corollary for *algebraic algorithms.*

2.4.12 ***PROPOSITION Ubiquity of Algebraic Algorithms:*** *Let $\mathcal{U}$ be a set of structured functions mapping some subsets of $\mathcal{S}(R; X)$ to other subsets of $\mathcal{S}(R; X)$. Assume that for each $A, C \subseteq \mathcal{S}(R; X)$ and map $\alpha : A \to C$ in $\mathcal{U}$ and element $z \in *A$, the map $\kappa\alpha\overline{z}$ is a polynomial function. Then if $B, D \subseteq \mathcal{S}(R; X)$ and the map $\beta : B \to D$ lies in $C4(\mathcal{U})$, and $z \in *B$ it follows that $\kappa\beta\overline{z}$ is a polynomial function. Hence, any algorithm whose functions lie in $C4(\mathcal{U})$ is an algebraic algorithm.*

2.5 Structured Functions in the Representation of a Free Module

Here is an extended example which illustrates structured functions in the representation of a free module and an algebra structure on the free module. The free module and then the algebra is represented by $T \subset \mathcal{S}(R; X)$. Using that $\mathcal{POLY}$ is **C4** and working through the presentation of the example shows that the various module structure operations: $Zero, N, S, A$ are polynomial functions. So are the algebra structure operations: One, P.

[19] This means that the function P_i is a polynomial with coefficients in R. Also, unless the ring R is commutative, the coefficients and variables of $P_i(Z_1, \ldots, Z_m)$ are not assumed to commute.

[20] *Generated* as described above (2.4.7).

R is a copy of the ring which the free module is a module over. X contains Y a copy of the basis of the free module. X also contains: "(" and ")". If one were simultaneously working with several free modules, each of their bases would be subsets of X.

Certain elements of $\mathcal{S}(R;X)$ – but not all – are used to represent elements of the free module. The subset of $\mathcal{S}(R;X)$ we use to represent elements of the free module is the set of all finite sequences of pairs $((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}'))$ with $r_i{}' \in R$, $y_i{}' \in Y$ and the $y_i{}'$'s distinct. Denote this set by T. Since R is a copy of the ring, for extra clarity in this example, we will typically indicate elements of R as r' where r lies in the ring. Similarly, in this example, we indicate elements of Y as y' where y is in the given basis for the free module. With this convention a sequence $((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}')) \in T$ should be thought of as representing $\sum_{i=1}^{m} r_i y_i$ in the free module. Any such sequence with all of the r_i's equal to $0 \in R$ represents 0 in the free module. We also consider the empty sequence to lie in T and it also represents 0. Because r_i's may be zero, and $y_i{}'$'s may occur in any order, elements of the free module do not have unique representation.[21]

This definition of T lies between the one extreme of using a subset of T where elements of the free module would have unique representation and the other extreme of dropping the condition of the $y_i{}'$'s being distinct. In the latter case a sequence $((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}'))$ would still represent $\sum_{i=1}^{m} r_i y_i$ in the free module. Our intermediate choice of T is based upon a balance between the two considerations of giving adequate representation of elements of the free module and the module operations and algorithms involving the module being structured and easy to describe. When discussing *sum* in the free module we point out the necessity of lack of uniqueness of representation of elements of the free module.

In this example the module operations will be structured functions. Let us start with N, negation. $N : T \to \mathcal{S}(R;X)$, where $N((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}'))$ $= ((-r_1{}', y_1{}'), \ldots, (-r_m{}', y_m{}'))$. Here, for $r' \in R$, $-r'$ means $(-r)'$. Using $*((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}')) = ((*, y_1{}'), \ldots, (*, y_m{}')) = *N((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}'))$ it follows that N is a structured function.

Let us now do S, summation in the free module. Let $T^2 \subset \mathcal{S}(R;X)$ be the set of pairs (t_1, t_2) with $t_1, t_2 \in T$. T^2 may be thought of as $T \times T$. The definition of S is based upon having an operation u on pairs of sequences $((x_1{}', \ldots, x_\ell{}'), (y_1{}', \ldots, y_m{}'))$ where the $x_i{}'$'s are distinct elements of Y and the $y_i{}'$'s are distinct elements of Y. $u((x_1{}', \ldots, x_\ell{}'), (y_1{}', \ldots, y_m{}'))$ must be $(z_1{}', \ldots, z_n{}')$ where the $z_i{}'$'s are distinct elements of Y with $\{x_1{}', \ldots, x_\ell{}'\} \cup \{y_1{}', \ldots, y_m{}'\} = \{z_1{}', \ldots, z_n{}'\}$. One such u is given as follows. Let $z_1{}' = x_1{}'$, $\ldots$, $z_\ell{}' = x_\ell{}'$. If any $y_i{}'$'s do not occur among the $x_i{}'$'s let $z_{\ell+1}{}'$ be the first of the $y_i{}'$'s which do not occur, let $z_{\ell+2}{}'$ be the second of the $y_i{}'$'s which do not occur, etc. In any case, say u is such an operation. $S(t_1, t_2)$ is defined as follows: if $t_1 = ((p_1{}', x_1{}'), \ldots, (p_\ell{}', x_\ell{}'))$ and $t_2 = ((q_1{}', y_1{}'), \ldots, (q_m{}', y_m{}'))$ then $S(t_1, t_2) =$

[21] For example, $((r_1{}', y_1{}'))$ and $((r_1{}', y_1{}'), (0', y_2{}'))$ and $((0', y_2{}'), (r_1{}', y_1{}'))$ all represent the same element of the free module.

$((r_1{}', z_1{}'), \ldots, (r_n{}', z_n{}'))$ where $(z_1{}', \ldots, z_n{}') = u((x_1{}', \ldots, x_\ell{}'), (y_1{}', \ldots, y_m{}'))$ and the $r_k{}'$'s are given by:

$$r_k{}' = \begin{cases} (p_i + q_j)' & \text{if } z_k{}' = x_i{}' \text{ and } z_k{}' = y_j{}' \\ p_i{}' & \text{if } z_k{}' = x_i{}' \text{ and } z_k{}' \text{ does not equal any } y_j{}' \\ q_j{}' & \text{if } z_k{}' = y_j{}' \text{ and } z_k{}' \text{ does not equal any } x_i{}' \end{cases}$$

Notice that $*S(t_1, t_2) = ((*, z_1{}'), \ldots, (*, z_n{}'))$ and hence is determined by $u((x_1{}', \ldots, x_\ell{}'), (y_1{}', \ldots, y_m{}'))$ which is determined by $((*, x_1{}'), \ldots, (*, x_\ell{}')) = *t_1$ and $((*, y_1{}'), \ldots, (*, y_m{}')) = *t_2$. Hence S is a structured function.

Achieving *uniqueness of representation of elements of the free module* – by a different choice of T – is a tempting goal. It appears to be a poisoned attraction. Let us look at a particular case where $x_1 = y_1, x_2 = y_2$. Then $S(((p_1, x_1), (p_2, x_2)), ((q_1, x_1), (q_2, x_2))) = ((p_1 + q_1, x_1), (p_2 + q_2, x_2))$. If one wished uniqueness of representation of the elements of the module, one might modify S by specifying: omit "$(p_i + q_i, x_i)$" if $p_i + q_i = 0$ in the ring. S would no longer be structured because $*S(((p_1, x_1), (p_2, x_2)), ((q_1, x_1), (q_2, x_2)))$ would be $((*, x_1), (*, x_2))$ or $((*, x_1))$ or $((*, x_2))$ or $()$, depending on $(p_1, p_2, q_1, q_2) = \kappa(((p_1, x_1), (p_2, x_2)), ((q_1, x_1), (q_2, x_2)))$. In other words: $*S(s)$ would no longer simply depend upon $*s$.

S expresses the sum of two elements. For later use it will be handy to handle the sum of more than two elements at a time. This is easily done in terms of S and illustrates additional material we have presented. Let $T^h \subset \mathcal{S}(R; X)$ be the set of h-tuples $(t_1, \ldots, t_h)$ with $t_1, \ldots, t_h \in T$. Think of T^h as $T \times \cdots \times T$, h-times. Let $S_1 : T \to T$ be the identity map and assume by induction that $S_h : T^h \to T$ has been defined and is structured. $S_{h+1}(t_0, t_1, \ldots, t_h)$ is defined as $S(t_0, S_h(t_1, \ldots, t_h))$. This is the composition of S with Identity$\times S_h$. By Lemma (2.4.5) it follows that S_{h+1} is structured. Of course S_2 equals S. From now on use S to denote all the S_i's and consider $S : T^+ \to T$ where T^+ is the disjoint union: $T^1 \uplus T^2 \uplus T^3 \uplus \cdots$ and S acts as S_h on T^h.

The action of the ring on the module is captured by A. Let $RT \subset \mathcal{S}(R; X)$ denote the set of pairs (r', s) with $r' \in R$ and $s \in T$. If $s = ((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}'))$ then $A(r', s) = ((rr_1{}', y_1{}'), \ldots, (rr_m{}', y_m{}'))$, where $rr_i{}'$ equals $(rr_i)'$. Here $*(r', s) = (*, (*, y_1{}'), \ldots, (*, y_m{}'))$ and $*A(r', s) = ((*, y_1{}'), \ldots, (*, y_m{}'))$ so that $A : RT \to T$ is structured.

The module element zero is captured by a zero arity map $Zero$ to T, i.e. an element of T. Choose and fix any finite set of distinct elements $y_1{}', \ldots, y_n{}' \in Y$. Let $Zero$ be $((0', y_1{}'), \ldots, (0', y_n{}')) \in T$. Constants or zero arity maps are structured by (c) of Lemma (2.4.5). If $n = 0$ one is using the empty set to represent zero. This is fine when there is just one free module but if several free modules are present, one might have to be careful for which free module $()$ was representing the zero element.

Now assume that the underlying ring is commutative and let us extend this example to make the free module into an algebra (with unit) over the underlying ring. This involves

giving two more operations which are structured functions. The first operation represents the multiplicative unit of the algebra. This operation has arity zero and is automatically structured by (c) of Lemma (2.4.5). In terms of our representation of the module by T, the multiplicative unit is represented in T by: $u = ((r_1{}', y_1{}'), \ldots, (r_m{}', y_m{}')) \in T$, where the actual multiplicative unit in the algebra is equal to: $\sum_{i=1}^{m} r_i y_i$. Call this element *One* and consider it a zero arity map.

Last we must give the operation corresponding to the product structure on the algebra. The product of basis elements x and y of the algebra is a linear combination of basis elements, say $\sum_i r_i z_i$ with r_i's in the ring and z_i's basis elements. Let $t_{x,y} \in T$ represent $\sum_i r_i z_i$ in the module. Then for p, q in the ring the product $(px)(qy) = \sum_i pqr_i z_i$ is represented by $A(pq, t_{x,y})$. Let $RY = \{(p', y') | p$ is in the ring and y is an element in the basis for the algebra$\} \subset \mathcal{S}(R; X)$. $*(pq', t_{x,y}) = (*, *t_{x,y})$ and so is determined by x and y and hence by $((*, x'), (*, y')) = *((p', x'), (q', y'))$. It follows that the map $\gamma : RY^2 \to RT, ((p', x'), (q', y')) \to (pq', t_{x,y})$ is structured.

The next step is based upon distributivity and *flattening* a product of two sums. I.e. in a ring:

$$(u_1 + \cdots + u_\ell)(v_1 + \cdots + v_m) \quad =$$

$$(u_1 v_1 + \cdots + u_1 v_m) + (u_2 v_1 + \cdots + u_2 v_m) + \cdots + (u_\ell v_1 + \cdots + u_\ell v_m)$$

Let $flat : T^2 \to (RY^2)^+$,

$$(((p_1{}', x_1{}'), \ldots, (p_\ell{}', x_\ell{}')), ((q_1{}', y_1{}'), \ldots, (q_m{}', y_m{}'))) \to$$

$$(((p_1{}', x_1{}'), (q_1{}', y_1{}')), \ldots, ((p_1{}', x_1{}'), (q_m{}', y_m{}')),$$
$$((p_2{}', x_2{}'), (q_1{}', y_1{}')), \ldots, ((p_2{}', x_2{}'), (q_m{}', y_m{}')),$$
$$\ldots,$$
$$((p_\ell{}', x_\ell{}'), (q_1{}', y_1{}')), \ldots, ((p_\ell{}', x_\ell{}'), (q_m{}', y_m{}')))$$

$flat$ is structured because it simply rearranges structure. The composite: $SA^+\gamma^+ flat : T^2 \to (RY^2)^+ \to RT^+ \to T^+ \to T$ is structured because each of the factors is structured. Note that for $f : S \to T$ both $f^* : S^* \to T^*$ and $f^+ : S^+ \to T^+$ are defined as $f^h = f \times \cdots \times f$, h-times on S^h. Let P denote the composite and let the reader check that: $P = SA^+\gamma^+ flat : T^2 \to T$ correctly corresponds to algebra product.

This completes the example of free module and algebra.

2.6 Examples of Other Algebraic Structures and Algorithms

Many algorithms in computational algebra are *algebraic*.

Suppose that $P(Z_1, \ldots, Z_n)$ is a polynomial with coefficients from the algebra represented by T. The coefficients of P are not assumed to commute with the variables nor are the variables assumed to commute with each other. Considering $T^n \subset \mathcal{S}(R; X)$, then evaluating P at elements of T^n maps T^n to T. We leave to the reader to show that this is a polynomial function. This enables us to give examples of some algebraic algorithms.

2.6.1 ***DETERMINANT:*** *Consider the determinant of a matrix. There are many ways to compute this, here is one which shows that computing the determinant of a matrix is an algebraic algorithm. To begin, consider an $n \times n$-matrix, $(Z_{i,j})$, where the $Z_{i,j}$'s are variables in $\boldsymbol{Z}[\{Z_{i,j}\}]$. $|Z_{i,j}|$, the determinant of $(Z_{i,j})$ is a polynomial in n^2 variables in $\boldsymbol{Z}[\{Z_{i,j}\}]$. Now consider $n \times n$-matrices over R where R is (a copy of) a commutative ring. Flatten the matrices so that they are represented by a sequence of n^2 elements of R. Flatten $|Z_{i,j}|$ according to the same method used to flatten matrices so that evaluating the polynomial $|Z_{i,j}|$ on a sequence of n^2 elements of R coincides with taking the determinant of the corresponding matrix. Now the algorithm for computing determinant is:*

Initialize:	$(r_1, \ldots, r_{n^2})$
step_1:	$Answer = \lvert Z_{i,j} \rvert (r_1, \ldots, r_{n^2})$
step_2:	*stop* $(Answer)$

Call this algorithm $\mathcal{DET}_n$. It is algebraic because step_1 is polynomial.

2.6.2 ***DISCRIMINANT:*** *This example is based upon computing discriminants of polynomials. See [5] for details about the discriminant of a polynomial. Suppose the underlying ring is commutative and an integral domain. Consider the degree n polynomial $P(Y) = Z_0 Y^n + Z_1 Y^{n-1} + \cdots + Z_{n-1} Y + Z_n \in \boldsymbol{Z}[\{Z_i\}][Y]$. The resultant of P and its derivative, with respect to Y, may be computed as the determinant of the Sylvester matrix coming from P and its derivative. This determinant is a polynomial: $\mathcal{D}(Z_0, \ldots, Z_n) \in \boldsymbol{Z}[\{Z_i\}]$. Here is an algebraic algorithm which operates on degree n polynomials with coefficients from R.*

Initialize:	$(Z_0, \ldots, Z_n)$
step_1:	$Answer = \mathcal{D}(Z_0, \ldots, Z_n)$
step_2:	*goto step_4 if* $Answer = 0$
step_3:	*stop* (1)
step_4:	*stop* (0)

Call this algorithm $\mathcal{DISC}_n$. Suppose that $Q(Y) = r_0 Y^n + r_1 Y^{n-1} + \cdots + r_{n-1} Y + r_n$ is a polynomial and that $Q(Y)$ is strictly of degree n in the sense that r_0 is not zero. Answer is set equal to $\mathcal{D}(r_0, \ldots, r_n)$, at step_1 and this is $(-1)^{n(n-1)/2} r_0$ times the discriminant

of $Q(Y)$. Hence the algorithm returns 0 if $Q(Y)$ has zero discriminant and returns 1 if $Q(Y)$ has non-zero discriminant. Based on the significance of a polynomial having non-zero discriminant, this is an algebraic algorithm where $\mathcal{DISC}_n(r_0, \ldots, r_n)$ returns 1 if $Q(Y)$ has n distinct roots in an algebraic closure of the field of fractions of R and otherwise returns 0.

The following two simple algorithms illustrate difficulties which arise in working with real valued functions and floating point approximation.

2.6.3 *POLY-POSITIVE:* *Here is a very simple algorithm, which is algebraic if $f(Y_1, \ldots, Y_n)$ is a polynomial in n-variables. $R = \boldsymbol{R}$.*

Initialize:	$(Y_1, \ldots, Y_n)$
step_1:	$X = f(Y_1, \ldots, Y_n)$
step_2:	*goto step_4 if* $X > 0$
step_3:	*stop* (0)
step_4:	*stop* (1)

Let us call this algorithm $\mathcal{E}$. For $r_1, \ldots, r_n \in R$, $\mathcal{E}(r_1, \ldots, r_n)$ outputs 0 or 1 depending whether $f(r_1, \ldots, r_n)$ is less than or equal to zero or is greater than zero. If floating point approximation is used in the computation of $f(r_1, \ldots, r_n)$ then the positivity test makes the output of the algorithm likely to output incorrect results. Nevertheless, branching on the basis of a test like: $X > 0$ is quite common, for example this occurs in the Sturm algorithm and the convex hull algorithm.

2.6.4 *REPEAT-ADD:* *Consider the following algebraic algorithm where $R = \boldsymbol{R}$. This algorithm has interesting convergence properties.*

Initialize:	$()$
step_1:	$X = 0$
step_2:	$X = \frac{1}{3} + X$
step_3:	*goto step_5 if* $X \geq 1$
step_4:	*goto step_2*
step_5:	*stop* (X)

This essentially represents a while loop, where the loop is repeated "while" $X < 1$. When performed with exact arithmetic this algorithm terminates at step_5 with an output of 1. When performed with a fixed precision decimal floating point approximation this algorithm terminates at step_5 with an output equal to the floating point approximation to $\frac{4}{3}$. Simply increasing the precision brings the result closer to $\frac{4}{3}$, not closer to 1.

2.6.5 ***BUCHBERGER:*** *Let R represent a field. And let T represent the R algebra which is the ring of polynomials $R[x_1, \ldots, x_n]$ in n variables over R. Then, the Buchberger algorithm (see [2]) for calculating Gröbner bases is an algebraic algorithm. S-polynomials and polynomial reductions may be defined without division. In fact this is typically done when generalizing the Buchberger algorithm to polynomial rings over integral domains. Or see [6] where the non-division form of the Buchberger algorithm is utilized to give floating point approximation methods for the Buchberger algorithm. This is one of the roots of the present paper. We do not attempt to reduce [6] to a few lines here, but rather just mention one aspect of the Buchberger algorithm here. Namely, how selecting the leading term of a polynomial can be accomplished in terms of polynomial functions and predicates. This discussion presumes some familiarity with the Buchberger algorithm.*

Assume that $R[x_1, \ldots, x_n]$ has been assigned a term order and consider a polynomial $f = \sum_\alpha c_\alpha x^\alpha$, where α is a sequence of non-negative integers $(\alpha_1, \ldots, \alpha_n)$ and x^α stands for $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. Suppose that β is the unique maximal exponent in the term order with c_β non-zero. Then c_β is the leading coefficient of f and is denoted $LC(f)$. x^β is the leading power product of f and is denoted $LP(f)$. f may have terms $c_\gamma x^\gamma$ where γ is greater than β in the term order but $c_\gamma = 0$. Hence the map sending f to $LP(f)$ is ***not*** *structured because it depends on the coefficients of f, not just the structure of f. Now let $(\alpha(1), \ldots, \alpha(n_f))$ be all of the exponents which appear in the expression of f: $\sum_\alpha c_\alpha x^\alpha$. Assume that the polynomial ring is represented by T as in the extended example and all x^α are (copies of elements) in X. Then in the representation of f, all of the α_i's are distinct. Assume that the sequence of exponents $(\alpha(1), \ldots, \alpha(n_f))$ of f is ordered so that in the term order: $\alpha(1) > \alpha(2) > \ldots > \alpha(n_f)$. Let exponents denote the map from T to $\mathcal{S}(R; X)$ which sends f to the ordered, descending sequence $(\alpha(1), \ldots, \alpha(n_f))$. exponents is a polynomial function. In fact exponents only depends on structure and is constant in terms of coefficients.*

Extracting exponents(f) is the first part of finding $LC(f)$ and $LP(f)$. Extracting specific coefficients is the second. For an exponent β, M_β denotes the map from T to $\mathcal{S}(R; X)$ which sends $f = \sum_\alpha c_\alpha x^\alpha$ to c_β if β is among the exponents appearing in f and zero otherwise. This map is a polynomial function.

The final part of finding $LC(f)$ and $LP(f)$ involves stepping through the sequence exponents(f). For each $\alpha(i)$ in the sequence one uses the predicate $IS\_ZERO()$ as in: $IS\_ZERO(M_{\alpha(i)})$. This controls whether to continue stepping through the sequence or stop with the critical $\alpha(i)$. If all of the $M_{\alpha(i)}$'s are zero, one fully traverses exponents(f) and f is the zero polynomial. One takes whatever action has been determined for this case. Alternatively, upon determining the first $\alpha(i)$ with $M_{\alpha(i)}$ non-zero, then $LC(f) = M_{\alpha(i)}$ and $LP(f) = x^{\alpha(i)}$.

We used the term "stepping through" a couple of times. One way of stepping through a sequence $(e_1, \ldots, e_n)$ is to enter a loop which keeps removing the leading element of

the sequence.[22] *While there may be predicates within the loop, there is likely to be an empty-set test, actually, empty-sequence test to terminate the loop in case all the elements are exhausted. Such an empty-sequence test is a predicate p(sequence) of the form IS_EMPTY(sequence) and returns TRUE or FALSE depending whether sequence equals () or not.*

3 Approximation

3.1 Posets and Gauges

We must define a notion of approximation in R to discuss stability of $\mathcal{CCB}$ algorithms, (4.3.9). From now on R is (a copy of) a ring. Stability will be with respect to an ordering in R determined by a poset P with two binary operations: $\hat{+}$ and $\hat{\times}$. As will emerge later, these operations correspond to the addition and multiplication in R. Negation in R is handled automatically by the axioms on the map from R to the poset, see (3.1.5,$\boldsymbol{\nu 1}$) below. If R were to have other operations, they would have to be accounted for by other operations for P or other axioms.

If P is a rooted poset, we denote the root element by $\hat{0}$.

Suppose $\{\alpha_n\}_n$ is a sequence in P indexed by non-negative integers. $\alpha_n \to \hat{0}$ $(n \to \infty)$ means $\forall \epsilon \in P \setminus \{\hat{0}\}$, $\exists N$ such that $n \geq N \Rightarrow \alpha_n < \epsilon$.[23] "$\alpha_n$ is bounded" means $\exists \gamma \in P$ such that $\alpha_n \leq \gamma$ for sufficiently large n. It follows immediately that $\alpha_n \to \hat{0}$ $(n \to \infty) \Rightarrow$ "α_n is bounded".[24]

In this paper we use P together with *sequences* to describe convergence and other topological phenomena. The reliance on *sequences* rather than *nets* – another topological technique – presupposes that P has the following countability property[25] : there is a *countable* set $P_0 \subset P$ where $\forall\ \epsilon \in P \setminus \{\hat{0}\}\ \exists\ \delta \in P_0 \setminus \{\hat{0}\}$ such that $\delta < \epsilon$. This condition does not imply that P is countable and is clearly satisfied by all P we introduce in examples.

Suppose that the two binary operations, $\hat{+}$ and $\hat{\times}$ satisfy the following properties,

[22] Of course, one can do this to a copy so as not to *damage* the original sequence.

[23] I.e. $a_n < \epsilon$ *for sufficiently large* n.

[24] Note that the definition of $\alpha_n \to \hat{0}$ $(n \to \infty)$ does not require that P is rooted and contains the element $\hat{0}$. We could use the terminology "α_n gets arbitrarily small" in this case, but instead we use the given notation. Hence the reader should not think that the notation $\alpha_n \to \hat{0}$ $(n \to \infty)$ implies that P is rooted and contains the element $\hat{0}$. On the other hand if P is not rooted but (3.1.1,**P1**) through (3.1.1,**P4**) below are satisfied, one may enlarge P by adding a root element $\hat{0}$ and extend $\hat{+}$ and $\hat{\times}$ to this enlarged set as required by (3.1.1,**P0**). Hence, we may assume that P (or rather this extension) has a (root) element $\hat{0}$.

[25] Which corresponds to the first axiom of countability in topology.

but are not assumed to satisfy associativity. We give two equivalent forms of the fourth property, because at times one is more convenient than the other.

3.1.1 ***DEFINITION Poset with Additional Operations:***

(P0) *If P is rooted:* $\forall \alpha \in P,\ \hat{0} \hat{+} \alpha = \alpha \hat{+} \hat{0} = \alpha$ *and* $\hat{0} \hat{\times} \alpha = \alpha \hat{\times} \hat{0} = \hat{0}$

(P1) $\alpha \leq \alpha'$ *and* $\beta \leq \beta' \Rightarrow \alpha \hat{+} \beta \leq \alpha' \hat{+} \beta'$

(P2) $\alpha \leq \alpha'$ *and* $\beta \leq \beta' \Rightarrow \alpha \hat{\times} \beta \leq \alpha' \hat{\times} \beta'$

(P3) $\alpha_n \to \hat{0}$ *and* $\beta_n \to \hat{0}\ (n \to \infty) \Rightarrow \alpha_n \hat{+} \beta_n \to \hat{0}\ (n \to \infty)$

(P4) *"*$\alpha_n \to \hat{0}\ (n \to \infty)$ *and* β_n *is bounded" or "*$\beta_n \to \hat{0}\ (n \to \infty)$ *and* α_n *is bounded"* $\Rightarrow$ $\alpha_n \hat{\times} \beta_n \to \hat{0}\ (n \to \infty)$

(P4′) $\alpha_n \to \hat{0}\ (n \to \infty) \Rightarrow \alpha_n \hat{\times} \beta \to \hat{0}\ (n \to \infty)$ *and* $\beta_n \to \hat{0}\ (n \to \infty) \Rightarrow \alpha \hat{\times} \beta_n \to \hat{0}\ (n \to \infty)$

Examples of P.

3.1.2 ***EXAMPLE POSREAL:*** $\boldsymbol{R}^+$*, the set of all non-negative real numbers with the ordinary* $\leq$*.* $\hat{0}$*,* $\hat{+}$ *and* $\hat{\times}$ *are the ordinary* 0*,* $+$ *and* $\times$*, respectively.*

3.1.3 ***EXAMPLE INTINF:*** $\boldsymbol{Z} \cup \{\infty\}$*, the set of all integers and a symbol infinity with reverse order* $\geq$ *as* $\leq$*.* $\hat{0} = \infty$*,* $a \hat{+} b = \min(a, b)$ *and* $a \hat{\times} b = a + b$*.*

3.1.4 ***EXAMPLE PRIMEALL:*** *Let* $\boldsymbol{P}$ *be the set of integer primes and let* $\mathcal{P}$ *be the set of* ***finite*** *subsets of* $\boldsymbol{P}$*. In* $\mathcal{P}$ *use reverse set inclusion* $\supseteq$ *as* $\leq$*. With* $\hat{+} = \cap$ *and* $\hat{\times} = \cup$ *(3.1.1,****P1****) through (3.1.1,****P4****) are satisfied. Hence it is possible to adjoin a root* $\hat{0}$ *to* $\mathcal{P}$ *so that (3.1.1,****P0****) is also satisfied. In this case* $\hat{0}$ *has a natural interpretation as the subset of* $\boldsymbol{P}$ *consisting of* $\boldsymbol{P}$ *itself. (Since* $\boldsymbol{P}$ *is an* ***infinite*** *set of primes, it is not already an element of* $\mathcal{P}$*.) With reverse set inclusion* $\supseteq$ *as* $\leq$*,* $\cap$ *as* $\hat{+}$ *and* $\cup$ *as* $\hat{\times}$*,* $\boldsymbol{P}$ *acts as* $\hat{0}$ *in the set* $\mathcal{P} \cup \{\boldsymbol{P}\}$*. Henceforth, when referring to this example we consider* P *to be* $\mathcal{P} \cup \{\boldsymbol{P}\}$ *and* $\boldsymbol{P} = \hat{0}$*. Now (3.1.1,****P0****) through (3.1.1,****P4****) are satisfied.*

To define a notion of approximation in R, requires a function somewhat like a *norm* or *valuation*. Rather than call it a *pseudo norm* or *semi-valuation*, we call it a *gauge*. Now we assume that P is rooted.

3.1.5 *DEFINITION Gauges:* *$\nu : R \to P$ satisfying the following properties is a gauge.*

($\boldsymbol{\nu 0}$) $\nu(a) = \hat{0}$ *iff* $a = 0$

($\boldsymbol{\nu 1}$) $\nu(-a) = \nu(a)$ *for* $a \in R$

($\boldsymbol{\nu 2}$) $\nu(a+b) \leq \nu(a) \hat{+} \nu(b)$ *for* $a, b \in R$

($\boldsymbol{\nu 3}$) $\nu(ab) \leq \nu(a) \hat{\times} \nu(b)$ *for* $a, b \in R$

Let us look at some examples. The examples which follow build upon earlier examples of P.

Examples of ν.

For the first example, P is $\mathbf{R}^+$, as in the example **POSREAL** (3.1.2).

3.1.6 *EXAMPLE ABSOLUTE VALUE:* *Let $R = \boldsymbol{R}$. For $a \in \boldsymbol{R}$ let,*

$$\nu(a) = |a| \text{ the absolute value of } a$$

This is an Archimedean gauge, in the sense of Archimedian valuation.

For the next two examples, P is $\mathbf{Z} \cup \{\infty\}$, as described in the example **INTINF** (3.1.3).

3.1.7 *EXAMPLE p-VAL:* *Let $R = \boldsymbol{Z}$. Fix a prime p. Let $\nu(0) = \infty$ and for $0 \neq a \in \boldsymbol{Z}$,*

$$\nu(a) = a_p \text{ where } a = \pm \textstyle\prod_q q^{a_q}, \text{ the factorization of } a$$

3.1.8 *EXAMPLE p-ADICS:* *Let $R = \boldsymbol{Q}_p$, the field of p-adic numbers. Let $\nu(0) = \infty$ and for $0 \neq \alpha \in \boldsymbol{Q}_p$,*

$$\nu(\alpha) = t \text{ where } \alpha = \textstyle\sum_{n=t}^{\infty} a_n p^n \text{ with } a_t \neq 0 \text{ and } 0 \leq a_n < p$$

$\sum_{n=t}^{\infty} a_n p^n$ is the p-adic expansion of α. This is non-Archimedean in the sense of a p-adic valuation, being non-Archimedean.

For the last example, P is $\mathcal{P} \cup \{\boldsymbol{P}\}$, as described in the example **PRIMEALL** (3.1.4).

3.1.9 *EXAMPLE PRIMEDIVISORS:* *Let $R = \boldsymbol{Z}$. Let $\nu(0) = \hat{0}$ and for $0 \neq n \in \boldsymbol{Z}$,*

$$\nu(n) = \{p \in \boldsymbol{P} \mid p|n\} \text{ for } n \in \boldsymbol{Z},$$

As usual, $p|n$ indicates that p divides n. Notice that $\nu(1)$ is the empty set.

Now let us utilize ν to define approximation for R.

3.1.10 ***DEFINITION Approximation Pairs and Convergent Approximation:*** *A pair of maps $(\rho : R \to R, \alpha : R \to P)$ is an* ***approximation pair*** *for R if* ***Approx0*** *holds. We also may say that ρ is an approximation with precision α or that α is an error bound for ρ. A sequence of pairs of maps: $\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$ is a* ***convergent approximation*** *for R if* ***Approx1*** *and* ***Approx2*** *hold. If in addition,* ***Approx1*** *holds for all k in the sequence, then $\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$ is a* ***strict convergent approximation*** *for R.*

(Approx0) *$\nu(a - \rho(a)) \leq \alpha(a)$ for all $a \in R$.*

(Approx1) *$\exists M$ such that $k \geq M \Rightarrow \alpha_k$ is an error bound for ρ_k. In other words, $(\rho_k : R \to R, \alpha_k : R \to P)$ is an approximation pair for sufficiently large k.*

(Approx2) *For each $a \in R$, $\alpha_k(a) \to \hat{0}$ as $k \to \infty$*

A convergent approximation is called *strict* if (ρ_k, α_k) is an approximation pair for R for all k. A convergent approximation is *bounded* in the sense that

3.1.11 $\forall a \in R$, $\exists \gamma_a \in P$ *such that* $\nu(\rho_k(a)) \leq \gamma_a$ *for sufficiently large* k

γ_a can be chosen as $\epsilon \hat{+} \nu(a)$, $\forall \epsilon \in P \setminus \{\hat{0}\}$. With M as in (3.1.10,**Approx1**) and $k \geq M$:

$$\nu(\rho_k(a)) = \nu((\rho_k(a) - a) + a) \leq \nu(\rho_k(a) - a) \hat{+} \nu(a) \leq \alpha_k(a) \hat{+} \nu(a)$$

by (3.1.5,$\boldsymbol{\nu 2}$) and (3.1.1,**P1**). Now apply (3.1.10,**Approx2**) and (3.1.5,$\boldsymbol{\nu 2}$).

The first example we present of a convergent approximation may appear trivial but plays a significant role in stabilizing algorithms.

3.1.12 ***EXAMPLE The Identity Convergent Approximation Linked to a Convergent Sequence in P:*** *Suppose that $\{p_k\}_k$ is a sequence of elements of P which converges to $\hat{0}$. For each k let ρ_k be the identity map from R to itself.*[26] *Let $\alpha_k : R \to P$ be the constant function sending each element of R to p_k. Then it is immediate that the sequence of pairs of maps $\{(\rho_k, \alpha_k)\}_k$ is a convergent approximation for R. It may be called the identity convergent approximation arising from $\{p_k\}_k$.*

Now some examples which build upon previous examples.

Examples of Convergent Approximations.

The first two examples continue the example **ABSOLUTE VALUE**, (3.1.6).

[26] Meaning that for $r \in R$, $\rho_k(r) = r$ and not $\rho_k(r) = 1$, unless of course $r = 1$.

3.1.13 ***EXAMPLE FLOATING POINT APPROXIMATION:*** *So-called round-off to k digits gives a floating point representation of each element of $\boldsymbol{R}$ with base 10, where α_k is a rounding error upper bound. For example,*

$$\rho_k(4/9) = .\overbrace{444\cdots4}^{k\ digits} \quad and \quad \rho_k(5/9) = .\overbrace{555\cdots6}^{k\ digits}$$

Here we can set

$$\alpha_k(a) = 5 \times 10^{-(k+1)}$$

3.1.14 ***EXAMPLE CONTINUED FRACTION (or DIOPHANTINE) APPROXIMATION:*** *For an irrational number $\omega \in \boldsymbol{R}$, let $\frac{p_k}{q_k} \in \boldsymbol{Q}$ ($(p_k, q_k) = 1$, $q_k > 0$) be the result from the k^{th} truncation of the simple continued fraction expansion of ω, where*

$$\left|\omega - \frac{p_k}{q_k}\right| < \frac{1}{{q_k}^2}$$

We then define $\rho_k : \boldsymbol{R} \to \boldsymbol{Q}$ by

$$\rho_k(\omega) = \begin{cases} \frac{p_k}{q_k} & if\ \omega \notin \boldsymbol{Q} \\ \omega & if\ \omega \in \boldsymbol{Q} \end{cases}$$

and define α_k by

$$\alpha_k(\omega) = \begin{cases} \frac{1}{q_k^2} & if\ \omega \notin \boldsymbol{Q} \\ 0 & if\ \omega \in \boldsymbol{Q} \end{cases}$$

We refer the reader to [4] for the theory of continued fractions.

The next example continues with the previous example **p-ADICS** (3.1.8).

3.1.15 ***EXAMPLE TRUNCATION in p-ADIC EXPANSION:*** *Let $\alpha \in \boldsymbol{Q}_p$ be represented by $\sum_{n=t}^{\infty} a_n p^n$ ($a_t \neq 0$). ρ_k is a truncation at the $(k+1)^{st}$ term, that is,*

$$\rho_k(\alpha) = \sum_{n=t}^{t+k-1} a_n p^n,$$

where

$$\alpha_k(\alpha) = t + k$$

See [4] also for the theory of p-adic numbers.

The last two examples continue with the example **PRIMEDIVISORS** (3.1.9).

3.1.16 ***EXAMPLE TRUNCATION in FACTORIZATION:*** *Let* $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 7$, $p_5 = 11$, ... *so that* p_i *be the* i^{th} *prime. Define* $\rho_k(0) = 0$ *and* $\rho_k(1) = 1$. *Also define* $\alpha_k(0) = \alpha_k(1) = \boldsymbol{P}$. *Recall that* $\boldsymbol{P}$ *is the* $\hat{0}$ *in this* P. *For* $n \in \boldsymbol{Z}$, *suppose that* n *is factorized as* $\pm p_1^{n_1} p_2^{n_2} \cdots p_t^{n_t}$ *(*$n_i \geq 0$ *for* $i \in [1, t-1]$ *and* $n_t \geq 1$*). Then*

$$\rho_k(n) = \begin{cases} \pm p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k} & \text{if } k \leq t \\ \pm p_1^{n_1} p_2^{n_2} \cdots p_t^{n_t} = n & \text{if } k > t \end{cases}$$

Set $P_k = \{p_1, p_2, \ldots, p_k\}$ *and define*

$$\alpha_k(n) = \{p \in P_k \mid p | (n - \rho_k(n))\}$$

3.1.17 ***EXAMPLE MODULAR REMAINDER:*** *Remainder of integers modulo a prime may also be considered to be an approximation in our framework. For* $n \in \boldsymbol{Z}$, $\rho_k : \boldsymbol{Z} \to \boldsymbol{Z}$ *is defined by*

$$\rho_k(n) = \bar{n} \text{ where } 0 \leq |\bar{n}| < p_k, \; n \text{ and } \bar{n} \text{ have the same sign and } n \equiv \bar{n} \pmod{p_k}$$

$$\alpha_k(n) = \{p \in P_k \mid p | (n - \rho_k(n))\}$$

Approx2 in (3.1.10) is suitable for most purposes but does not rule out some anomalous behavior. For example, there are pathological examples of convergent approximations where $\{a_k\}_k$ is a sequence of elements of R which converges to $a \in R$ but the sequence $\{\rho_k(a_k)\}_k$ converges to an element of R different from a, or where the sequence $\{\rho_k(a_k)\}_k$ does not converge at all. When such pathologies must be ruled out, one must replace **Approx2** with a more restrictive condition. Here are two reasonable choices to replace **Approx2**, listed in order of increasing stringency:

3.1.18 (Approx2′) *If* $\{a_k\}_k$ *is a Cauchy sequence*[27] *of elements of* R *then* $\alpha_k(a_k) \to \hat{0}$ *as* $k \to \infty$.

3.1.19 (Approx2″) *If* $\{a_k\}_k$ *is a bounded sequence of elements of* R *then* $\alpha_k(a_k) \to \hat{0}$ *as* $k \to \infty$.

3.1.20 ***DEFINITION Continuous and Locally Compact Convergent Approximation:*** *A sequence of pairs of maps:* $\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$, *is a continuous convergent approximation if it satisfies* ***Approx1*** *and* **Approx2′**. *The sequence of pairs of maps is a locally compact convergent approximation if it satisfies* ***Approx1*** *and* **Approx2″**.

[27] *Cauchy sequences* and *convergent sequences* of elements of R are about to be defined.

Note that the examples **FLOATING POINT APPROXIMATION** (3.1.13), and **TRUNCATION in *p*-ADIC EXPANSION** (3.1.15) satisfy (3.1.19) because $\alpha_k(a)$ does not depend upon a and hence are locally compact convergent approximations.

As will be seen, the notion of convergent approximation is related to pointwise convergence of functions. The notion of locally compact convergent approximation is related to convergence of functions which is locally compactly uniform; i.e. uniform on compact neighborhoods of a point. The notion of continuous convergent approximation is related to continuous convergence of functions as defined and treated in [3].

3.2 Convergence

Gauges, like norms and valuations, support the definition of convergence. Here is the notion of convergence for R, R^n and $\mathcal{S}(R;X)$. Suppose that R has a gauge ν and $\{a_k\}_k$ is a sequence of elements of R:

3.2.1 *DEFINITION Convergence for R:* *We write $a_k \to a$ $(k \to \infty)$ to signify that $\nu(a_k - a) \to \hat{0}$ $(k \to \infty)$ in P.*

$\{a_k\}_k$ is a *Cauchy sequence* if for $\forall\ \epsilon \in P \setminus \{\hat{0}\}$, $\exists N$ such that $k, n \geq N \Rightarrow \nu(a_k - a_n) < \epsilon$. Convergent sequences are Cauchy sequences[28]. Whether *Cauchy sequences* must be converge depends upon R being complete as a topological space. For example, the real numbers are complete but the rational numbers are not.

At (3.1.20) and (4.2.6) we present **Approx2′** and **Approx2″** which are increasingly restrictive alternative hypotheses to **Approx2**. Similarly, at (4.2.1) and (5.4.1) we present **Convergence′** and **Convergence2″** which are increasingly restrictive alternative hypotheses to **Convergence**. For all of these, the fact that **hypothesis′** is more restrictive than the original **hypothesis** follows from the fact that a *constant* sequence, i.e. $\{a_k\}_k$ where all the $a_k's$ are equal, is Cauchy. Also, for all of these, the fact that **hypothesis″** is more restrictive than **hypothesis′** follows from the fact that if $\{a_k\}_k$ is a Cauchy sequence then it is bounded[29].

For R^n, let $\underline{a} \in R^n$ stand for an n–tuple $(a_1, \ldots, a_n)$ with the a_i's in R. Similarly having a sequence $\{\underline{a}_k\}_k$ of elements of R^n is equivalent to having n sequences $\{a_{1,k}\}_k, \ldots, \{a_{n,k}\}_k$ of elements of R where each $\underline{a}_k$ is the n–tuple $(a_{1,k}, \ldots, a_{n,k})$. We use similar **underline**

[28] If $a_k \to a$ $(k \to \infty)$ then $\nu(a_k - a) \to \hat{0}$ $(k \to \infty)$. Hence, $\nu(a_k - a) + \nu(a_k - a) \to \hat{0}$ $(k \to \infty)$. Hence for $\epsilon \in P \setminus \{\hat{0}\}$, $\exists N$ such that $k \geq N \Rightarrow \nu(a_k - a) + \nu(a_k - a) < \epsilon$. If $a_k = a$ for $k \geq N$ then $\{a_k\}_k$ is *Cauchy*. Otherwise there is $M \geq N$ with $a_M \neq a$ and so $\nu(a_M - a) \neq \hat{0}$. Now there is $L \geq M$ where $k \geq L \Rightarrow \nu(a_k - a) < \nu(a_M - a)$. From here it is easy to show that $k, n \geq L \Rightarrow \nu(a_k - a_n) < \epsilon$ and the sequence is *Cauchy*.

[29] This is not very difficult to see and is left to the reader.

notation for P^n. $\underline{\hat{0}}$ of course denotes $(\hat{0},\ldots,\hat{0}) \in P^n$. For $\underline{a} \in R^n$ let $\underline{\nu(a)} \in P^n$ denote the n–tuple $(\nu(a_1),\ldots,\nu(a_n))$. For $\underline{\alpha},\underline{\beta} \in P^n$: $\underline{\alpha} < \underline{\beta}$ signifies that $\alpha_i < \beta_i$ for $i = 1,\ldots,n$. Similarly for "$\leq$", "$\geq$" and "$>$". A sequence $\{\underline{\alpha_k}\}_k$ of elements of P^n is said to be *bounded* if for $j = 1,\ldots,n$ each of the component sequences $\{\alpha_{j,k}\}_k$ is bounded, as defined at the beginning of Section (3.1). A sequence $\{\underline{a}_k\}_k$ of elements of R^n is *bounded* iff the sequence $\{\underline{\nu(a)}_k\}_k$ is bounded.

3.2.2 *DEFINITION Convergence for P^n:* *Suppose that $\{\underline{\epsilon}_k\}_k$ is a sequence of elements of P^n. $\underline{\epsilon}_k \to \underline{\hat{0}}$ $(k \to \infty)$ signifies that $\epsilon_{i,k} \to \hat{0}$ $(k \to \infty)$ for $i = 1,\ldots,n$.*

3.2.3 *DEFINITION Convergence for R^n:* *Suppose that $\{\underline{a}_k\}_k$ is a sequence of elements of R^n and $\underline{a} \in R^n$. $\underline{a}_k \to \underline{a}$ $(k \to \infty)$ signifies that $\underline{\nu(\underline{a}_k - \underline{a})} \to \underline{\hat{0}}$ $(k \to \infty)$.*

In the preceding, $\underline{a}_k - \underline{a}$ is the n–tuple $(a_{1,k} - a_1,\ldots,a_{n,k} - a_n)$. It is easily verified that $\underline{a}_k \to \underline{a}$ $(k \to \infty)$ is equivalent to $a_{i,k} \to a_i$ $(k \to \infty)$ for $i = 1,\ldots,n$. In other words, a sequence of elements of R^n converges to $\underline{a}$ if and only if it converges to $\underline{a}$ componentwise.

Just as for R, a convergent sequence in R^n is *Cauchy*. In R^n *Cauchy* may be defined as "component-wise" *Cauchy*.

3.2.4 *DEFINITION Convergence for $\mathcal{S}(R;X)$:* *Let $\{d_k\}_k$ be a sequence of elements of $\mathcal{S}(R;X)$ and let d be an element in $\mathcal{S}(R;X)$. $d_k \to d$ $(k \to \infty)$ signifies that there exists N such that*
(1) $k \geq N \Rightarrow *d_k = *d$, and
(2) for $\{\kappa(d_k)\}_{k \geq N}$, $\kappa(d_k) \to \kappa(d)$ $(k \to \infty)$.

Note that $\kappa(d_k) \in R^{\sharp d_k}$ and $\kappa(d) \in R^{\sharp d}$. For $k \geq N$: $\sharp d_k = \sharp d$ since $*d_k = *d$. Hence the convergence required in (2) is simply (componentwise) convergence in R^n, just defined.

3.3 Stability

Approximation in R easily extends to $\mathcal{S}(R;X)$. The underlying idea is to simply approximate elements of $\mathcal{S}(R;X)$ on a coefficientwise basis. If $d \in \mathcal{S}(R;X)$, then $\rho_k(d)$ is obtained by replacing each coefficient a in d by $\rho_k(a)$. More specifically, if $\kappa d = (a_1,\ldots,a_n) \in R^n$ then $\rho_k(d)$ is defined to be: $\langle *d, (\rho_k(a_1),\ldots,\rho_k(a_n))\rangle$.

3.3.1 *DEFINITION Stability of Algorithms:* *Suppose $I \in \mathcal{S}(R;X)$ is an input for an algorithm $\mathcal{A}$ and that $\mathcal{A}$ terminates normally on I.*[30] *We call $\mathcal{A}$ stable at I if for*

[30] In general, invoking $\mathcal{A}$ on input data which approximates I will not approximate the output: $\mathcal{A}(I)$.

every sequence $\{I_k\}_k$ *in* $\mathcal{S}(R;X)$ *which converges to* I *there is an* M *such that* $k \geq M \Rightarrow \mathcal{A}$ *with input data* I_k *terminates normally and the sequence of outputs:* $\{\mathcal{A}(I_k)\}_{k\geq M}$ *converges to* $\mathcal{A}(I)$*. Otherwise* $\mathcal{A}$ *is said to be unstable at* I*.*

The term "stable" has different meanings in different areas such as differential equations, economic models, and linear multistep iterations. With the qualifier "numerically", it has a stronger meaning than ours in numerical analysis. We refer the reader to [7] for the details on numerical stability. Let us look at the algebraic algorithms we gave above from the viewpoint of stability.

Examples of Stability.

All the examples below build on the previous example **FLOATING POINT APPROXIMATION** (3.1.13) ρ_k is the rounding to k digits and α_k is the rounding error upper bound.

3.3.2 ***DETERMINANT:*** *The algorithm presented at (2.6.1) is stable because it is simply polynomial evaluation and polynomials are continuous.*

3.3.3 ***DISCRIMINANT:*** *The algorithm presented at (2.6.2) is unstable and any such algorithm must be unstable. Recall that* $\mathcal{DISC}_n(r_0,\ldots,r_n)$ *returns* 1 *if* $Q(Y)$ *has* n *distinct roots in an algebraic closure of the field of fractions of* R *and otherwise returns* 0*. Consider* $Q(Y) = Y^2 + 0$ *as a polynomial with the one coefficient:* 0*. The correct answer is* 0 *since* $Q(Y)$ *has* 0 *as a multiple root. However, vary the coefficient* 0 *slightly and the resulting* $Q(Y)$ *has distinct roots. Hence, with any approximation to the coefficient* 0 *- other than* 0 *itself - the algorithm returns* 1*.*

3.3.4 ***POLY-POSITIVE:*** *The algorithm presented at (2.6.3) is unstable, consider the case* $f(Y_1,\ldots,Y_n)$ *is simply the linear polynomial in one variable* Y_1 *and the input is* 0*. Later we shall consider the situation where not only are input coefficients converted to floating point numbers, but also floating point arithmetic is used for the actual computation during the course of the algorithm. We will discuss a general framework for such a total conversion to floating point in Section 5.4.*

3.3.5 ***REPEAT-ADD:*** *The algorithm presented at (2.6.4) is unstable, see the discussion following the presentation of the algorithm.*

3.3.6 ***BUCHBERGER:*** *The algorithm presented at (2.6.5) is fundamentally unstable. In the univariate case, computing a Gröbner basis from the input polynomials computes a Greatest Common Divisor (GCD) of the input polynomials. This is true no-matter what specific method of computing a Gröbner basis is used. Since the GCD depends unstably on the input polynomials, Gröbner basis algorithms cannot be stable.*

The aim of this paper is to stabilize unstable $\mathcal{CCB}$ algorithms (4.3.9). I.e., given an unstable $\mathcal{CCB}$ algorithm $\mathcal{A}$, the aim is to utilize $\mathcal{A}$ to devise a family of other algorithms $\mathcal{A}_i$, so that for inputs I, $\mathcal{A}_i(I) \to \mathcal{A}(I)$. Moreover if $\{I_j\}_j$ is a sequence of inputs converging to I then:

$$lim_i(lim_j \mathcal{A}_i(I_j)) = \mathcal{A}(I)$$

The precise details appear later, (5.1.7). Of course the algorithms $\mathcal{A}_k$ should be closely related to $\mathcal{A}$. The method we present automatically derives $\mathcal{A}_k$ from $\mathcal{A}$ and knowledge of a certain discontinuity set of $\mathcal{A}$ when $\mathcal{A}$ is not too ill-behaved.

4 Bracket Coefficients

4.1 Basic Bracket Coefficient Considerations

Now fix R, X, $\mathcal{S}(R;X)$, P, ν, $\{\rho_k\}$, and $\{\alpha_k\}$. A *bracket coefficient* or simply BC is $[a,\alpha]$ where $a \in R$ and $\alpha \in P$. Let $\mathcal{BC}[R,P]$, be the set of bracket coefficients, $\{[a,\alpha] \mid a \in R, \alpha \in P\}$. As a set $\mathcal{BC}[R,P]$ is in one to one correspondence with the Cartesian product: $R \times P$. $\mathcal{BC}[R,P]$ is frequently abbreviate as $\mathcal{BC}$. If $\boldsymbol{b} \in \mathcal{BC}$ we use $\boldsymbol{b}_R$ to denote the R component of $\boldsymbol{b}$ and $\boldsymbol{b}_P$ to denote the P component of $\boldsymbol{b}$. So $\boldsymbol{b}_R \in R$, $\boldsymbol{b}_P \in P$ and $\boldsymbol{b} = [\boldsymbol{b}_R, \boldsymbol{b}_P]$.

Arithmetic in $\mathcal{BC}$ is defined as follows:

4.1.1 *DEFINITION BC-Arithmetic:*

Addition $[a,\alpha] + [b,\beta] = [a+b, \alpha \hat{+} \beta]$

Subtraction $[a,\alpha] - [b,\beta] = [a-b, \alpha \hat{+} \beta]$

Multiplication $[a,\alpha] \times [b,\beta] = [ab, (\nu(a) \hat{\times} \beta) \hat{+} (\alpha \hat{\times} \beta) \hat{+} (\alpha \hat{\times} \nu(b))]$

where $w \hat{+} x \hat{+} y \hat{+} \cdots \hat{+} z$ *denotes* $(\cdots((w \hat{+} x) \hat{+} y) \hat{+} \cdots) \hat{+} z$.

Elements of $\mathcal{BC}[R^m, P^m]$ are of the form $[\underline{e}, \underline{\epsilon}]$ with $\underline{e} \in R^m$ and $\underline{\epsilon} \in P^m$.

We must frequently *correspond* $\mathcal{BC}[R,P]^m$ with $\mathcal{BC}[R^m, P^m]$ or $R^m \times P^m$. Typically we *correspond* $([a_1,\alpha_1],\ldots,[a_m,\alpha_m])$ with $[(a_1,\ldots,a_m),(\alpha_1,\ldots,\alpha_m)]$. Going *from* $\mathcal{BC}[R,P]^m$ is done by extending the $()_R$ and $()_P$ notation, as follows:

4.1.2 $$([a_1,\alpha_1],\ldots,[a_m,\alpha_m])_R \equiv (a_1,\ldots,a_m)$$

$$([a_1, \alpha_1], \ldots, [a_m, \alpha_m])_P \equiv (\alpha_1, \ldots, \alpha_m)$$

$()_R$ and $()_P$ are *destructors*. The *constructor* to build elements of $\mathcal{BC}[R,P]^m$ from $\mathcal{BC}[R^m, P^m]$ or $R^m \times P^m$ is "$\overrightarrow{[}\ ,\ \overleftarrow{]}$" and is defined:

4.1.3 $$\overrightarrow{[}\ (a_1, \ldots, a_m), (\alpha_1, \ldots, \alpha_m)\ \overleftarrow{]} \equiv ([a_1, \alpha_1], \ldots, [a_m, \alpha_m])$$

With this notation, for $\underline{a} \in R^m$, $\underline{\alpha} \in P^m$ and $\underline{w} \in \mathcal{BC}[R,P]^m$:

4.1.4 $$\underline{w} = \overrightarrow{[}\ \underline{w}_R, \underline{w}_P\ \overleftarrow{]}$$

$$\underline{a} = \overrightarrow{[}\ \underline{a}, \underline{\alpha}\ \overleftarrow{]}_R$$

$$\underline{\alpha} = \overrightarrow{[}\ \underline{a}, \underline{\alpha}\ \overleftarrow{]}_P$$

Lastly, for a subset $A \subseteq R^m$ and subset $Q \subseteq P^m$ we define $\overrightarrow{[}\ A, Q\ \overleftarrow{]}$ by:

4.1.5 $$\overrightarrow{[}\ A, Q\ \overleftarrow{]} \equiv \{w \in \mathcal{BC}[R^m, P^m] | w = \overrightarrow{[}\ a, q\ \overleftarrow{]}\ \textit{for}\ a \in A, q \in Q\}$$

Most typically Q will be all of P^m.

4.2 Convergent Bounds

BC-Arithmetic combines the addition, subtraction, and multiplication operations of R with the corresponding operations of P in such a way that error is controlled in the sense that Lemmas (4.2.9) and (4.2.10) hold. What property of "+","−", "×" allows them to *usefully* extend to bracket coefficients? The answer is the *convergent bounds* defined below. Many functions, such as polynomials, have convergent bounds. This will appear in the following section where it will also be shown that the class of functions with convergent bounds has the **C4** property, (2.4.6). Eventually we shall also see that *having convergent bounds* is the necessary requirement on functions appearing in algorithms for our stability methods to apply.

To realize the full implication of *convergent bounds* we must consider maps from R^m to R^n for various m's and n's and define what it means for them to have convergent bounds. In the case of polynomial functions from R^m to R^n, a function is considered *polynomial* if its coordinate functions are *polynomial*. In the case of *convergent bounds* we may remain

multidimensional.[31] In keeping with primarily considering R^m and R^n *multidimensionally* rather than primarily in terms of products of copies of R or sequences of elements of R, we now usually omit underlining.[32] So, for example, we write $a \in R^m$ not $\underline{a} \in R^m$. If $a \in R^m$ we write $\nu(a) \in P^m$ not $\underline{\nu(a)} \in P^m$. We write $\hat{0}$ for $\underline{\hat{0}} \in P^m$.

One way of thinking about convergent bounds is that they are a certificate of continuity. See (4.6.4).

As in (3.1.20) and (4.2.6) where **Approx2′** and **Approx2″** are increasingly restrictive alternatives to **Approx2**, we introduce **Convergence′** and **Convergence″** as increasingly restrictive alternatives to **Convergence** in:

4.2.1 ***DEFINITION Convergent Bounds:*** *Let $A \subseteq R^m$, $C \subseteq R^n$ and $op : A \to C$. Let $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$. ω is a* ***convergent bound*** *for op if* ***Bound*** *and* ***Convergence*** *hold. ω is a* ***continuous convergent bound*** *for op if* ***Bound*** *and* **Convergence′** *hold. ω is a* **locally compact convergent bound** *for op if* ***Bound*** *and* ***Convergence″*** *hold.*

Bound: *For $a, a' \in A$ and $\epsilon \in P^m$: if $\nu(a - a') \leq \epsilon$ then $\nu(op(a) - op(a')) \leq \omega(\overrightarrow{[}\ a, \epsilon\ \overleftarrow{]})$.*

Convergence: *For sequences $\{\epsilon_k\}_k$ of elements of P^m where $\epsilon_k \to \hat{0}$ $(k \to \infty)$ and elements a of A, the sequence $\{\omega(\overrightarrow{[}\ a, \epsilon_k\ \overleftarrow{]})\}_k$ has the convergence*

$$\omega(\overrightarrow{[}\ a, \epsilon_k\ \overleftarrow{]}) \to \hat{0}\ (k \to \infty)$$

Convergence′: *ω has the following restricted form of convergence: For sequences $\{\epsilon_k\}_k$ of elements of P^m where $\epsilon_k \to \hat{0}$ $(k \to \infty)$ and* ***Cauchy sequences*** *$\{a_k\}_k$ of elements of A: the sequence $\{\omega(\overrightarrow{[}\ a_k, \epsilon_k\ \overleftarrow{]})\}_k$ has the convergence*

$$\omega(\overrightarrow{[}\ a_k, \epsilon_k\ \overleftarrow{]}) \to \hat{0}\ (k \to \infty)$$

Convergence″: *ω has the following restricted form of convergence: For sequences $\{\epsilon_k\}_k$ of elements of P^m where $\epsilon_k \to \hat{0}$ $(k \to \infty)$ and* ***bounded sequences*** *$\{a_k\}_k$ of elements of A: the sequence $\{\omega(\overrightarrow{[}\ a_k, \epsilon_k\ \overleftarrow{]})\}_k$ has the convergence*

$$\omega(\overrightarrow{[}\ a_k, \epsilon_k\ \overleftarrow{]}) \to \hat{0}\ (k \to \infty)$$

[31] This is a lie for two reasons. The first is that convergence in R^{power} and P^{power} is really a componentwise issue. The second is that the definition we give of $op : R^m \to R^n$ having a convergent bound is equivalent to the individual coordinate functions having a convergent bound. Nevertheless, the viewpoint is honestly multidimensional instead of *iterated unidimensional*.

[32] It is handy to use *underlining* when it is necessary to use a vector and its components. With *underlining*, 'a_i's are *automatically* the component entries of $\underline{a}$.

Let us use op_+, op_- and $op_\times$ as the functional notation for the infix operations "+","−", "×".[33] It is easy to check that if we define $\omega_\pm([r_1,\epsilon_1],[r_2,\epsilon_2])$ as $\epsilon_1 \hat{+} \epsilon_2$ then $\omega_\pm$ is a locally compact convergent bound for **both** op_+ and op_-. The "unary" operator "−", meaning *negation*, may be expressed in terms of the binary "−" by subtracting elements from 0. However it is easy to directly check that defining $\omega_{unary-}([r,\epsilon]) = \epsilon$ yields a locally compact convergent bound for unary "−".

A locally compact convergent bound for $op_\times$ is a bit more delicate. Define $\omega_\times([r_1,\epsilon_1],[r_2,\epsilon_2])$ as $(\nu(r_1) \hat{\times} \epsilon_2) \hat{+} (\epsilon_1 \hat{\times} \epsilon_2) \hat{+} (\epsilon_1 \hat{\times} \nu(r_2))$. *Convergence″* easily follows from (3.1.1,**P0**) – (3.1.1,**P4**).[34] The *bound* property uses the tricky equation: $(r_1 \times r_2) - (s_1 \times s_2) = r_1 \times (r_2 - s_2) - (r_1 - s_1) \times (r_2 - s_2) + (r_1 - s_1) \times r_2$, where the left hand side is $op_\times(r_1, r_2) - op_\times(s_1, s_2)$. Apply ν to both sides and use (3.1.5,$\boldsymbol{\nu 1}$) – (3.1.5,$\boldsymbol{\nu 3}$) together with $\nu(r_1 - s_1) \leq \epsilon_1$ and $\nu(r_2 - s_2) \leq \epsilon_2$ to obtain: $\nu(op_\times(r_1, r_2) - op_\times(s_1, s_2)) \leq (\nu(r_1) \hat{\times} \epsilon_2) \hat{+} (\epsilon_1 \hat{\times} \epsilon_2) \hat{+} (\epsilon_1 \hat{\times} \nu(r_2))$. Note that the right hand side is precisely $\omega_\times([r_1,\epsilon_1],[r_2,\epsilon_2])$.

Here are four more operations and their locally compact convergent bounds. Say $q \in R$ and define the four operators:

$$op_{q+} : R \to R, \quad r \to q + r$$

$$op_{+q} : R \to R, \quad r \to r + q$$

$$op_{q\times} : R \to R, \quad r \to q \times r$$

$$op_{\times q} : R \to R, \quad r \to r \times q$$

Of course $op_{q+} = op_{+q}$, because addition is commutative. $op_{q\times} = op_{\times q}$ if and only if q lies in the center of R. It is easy and left to the reader to check that the following ω's give respective locally compact convergent bounds:

$$\omega_{q+} : \mathcal{BC} \to P, \quad [r,\epsilon] \to \epsilon$$

$$\omega_{+q} : \mathcal{BC} \to P, \quad [r,\epsilon] \to \epsilon$$

$$\omega_{q\times} : \mathcal{BC} \to P, \quad [r,\epsilon] \to \nu(q) \hat{\times} \epsilon$$

$$\omega_{\times q} : \mathcal{BC} \to P, \quad [r,\epsilon] \to \epsilon \hat{\times} \nu(q)$$

If R has additional operations with convergent bounds they extend to $\mathcal{BC}$ as follows:

4.2.2 *DEFINITION BC Extension of Functions with Convergent Bounds:* *Let $A \subseteq R^m$, $C \subseteq R^n$ and $op : A \to C$. Suppose that $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$ is a* ***convergent***

[33] I.e. $op_+(a,b) = a + b$, etc.

[34] The (3.1.1,**P4′**) version of (3.1.1,**P4**) is handy here.

bound *for op. The extension of the pair* (op, ω) *to a map* $\overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to \overrightarrow{[}\ C, P^n\ \overleftarrow{]}$ *is denoted* op_ω *and for* $t \in \overrightarrow{[}\ A, P^m\ \overleftarrow{]}$ *is defined:*

$$op_\omega(t) \equiv \overrightarrow{[}\ op(t_R), \omega(t)\ \overleftarrow{]}$$

Or equivalently, if $t = \overrightarrow{[}\ a, p\ \overleftarrow{]}$ *then* $op_\omega(t) = \overrightarrow{[}\ op(a), \omega(\overrightarrow{[}\ a, p\ \overleftarrow{]})\ \overleftarrow{]}$.

The reader should check that the above extension of the pairs $(op_+, \omega_\pm)$, $(op_-, \omega_\pm)$ and $(op_\times, \omega_\times)$ coincides with the BC-arithmetic previously defined.

The notion of bracket coefficients together with convergent bounds might be considered a kind of algebraicization of *interval* as in *interval analysis*. From this viewpoint, BC-arithmetic is analogous to *interval arithmetic*, see [1] for details on interval analysis.

In order to quantify error control we must use approximation pairs and convergent approximations in R^{power}. All the preceding examples of approximation pairs and convergent approximations in R give approximation pairs and convergent approximations in R^{power}, as will be explained. The following generalizes (3.1.10).

4.2.3 ***DEFINITION Multidimensional Approximation Pairs and Convergent Approximation:*** *Suppose that* $A \subseteq R^m$, $\rho : A \to A$ *and* $\alpha : A \to P^m$. (ρ, α) *is an* ***approximation pair*** *for* A *if* ***Approx0*** *holds. We also may say that* ρ *is an approximation with precision* α *or that* α *is an error bound for* ρ. *A sequence of pairs of maps:* $\{(\rho_k : A \to A, \alpha_k : A \to P^m)\}_k$ *is a* ***convergent approximation*** *for* A *if* ***Approx1*** *and* ***Approx2*** *hold.*

(Approx0) $\nu(a - \rho(a)) \leq \alpha(a)$ *for all* $a \in A$.

(Approx1) $\exists M$ *such that* $k \geq M \Rightarrow \alpha_k$ *is an error bound for* ρ_k. *In other words,* $(\rho_k : A \to A, \alpha_k : A \to P^m)$ *is an approximation pair for* A *for sufficiently large* k.

(Approx2) *For each* $a \in A$, $\alpha_k(a) \to \hat{0}$ *as* $k \to \infty$

It follows that if $\{(\rho_k, \alpha_k)\}_k$ is a convergent approximation for A then for each $a \in A$, $\rho_k(a) \to a$ $(k \to \infty)$. By definition of convergence in R^m it follows that *componentwise*: $\rho_k(a) \to a$ $(k \to \infty)$. Hence, by the reasoning following (3.1.10), *componentwise* $\{\rho_k(a)\}_k$ is bounded. By the definition of "bounded in R^m" it follows that $\{\rho_k(a)\}_k$ is bounded.

As mentioned above (3.1.18), **Approx2** of (3.1.10) must sometimes be replaced by a stronger condition. This will occur in the consideration of utilizing approximate computation in place of exact computation in Section (5.4). In this case, **Approx2** of (4.2.3) must sometimes be replaced by a similar stronger condition. As before, here are two reasonable choices to replace **Approx2**, listed in order of increasing stringency:

4.2.4 (Approx2′) *If $\{a_k\}_k$ is a Cauchy sequence of elements of A then $\alpha_k(a_k) \to \hat{0}$ as $k \to \infty$.*

4.2.5 (Approx2″) *If $\{a_k\}_k$ is a bounded sequence of elements of A then $\alpha_k(a_k) \to \hat{0}$ as $k \to \infty$.*

Parallel to (3.1.20) define:

4.2.6 *DEFINITION Multidimensional Continuous and Locally Compact Convergent Approximation:* *A sequence of pairs of maps: $\{(\rho_k : A \to A, \alpha_k : A \to P^m)\}_k$, is a Continuous convergent approximation if it satisfies **Approx1** and* **Approx2′**. *The sequence of pairs of maps is a locally compact convergent approximation if it satisfies **Approx1** and* **Approx2″**.

Typically, multidimensional approximation pairs and multidimensional convergent approximations arise from one dimensional approximation pairs and one dimensional convergent approximations. This is done by a diagonalization process we now describe. When these diagonal extension techniques are applied to *continuous* or *locally compact* convergent approximations they yield *continuous* or *locally compact* multidimensional convergent approximations.

4.2.7 *EXAMPLE Diagonal Extension of One Dimensional Approximation Pairs to Multidimensional Approximation Pairs:* *This is a long title for a short process. Suppose that $(\rho : R \to R, \alpha : R \to P)$ is an **approximation pair** for R, (3.1.10,**Approx0**), and m is a positive integer. Let ρ^m denote the map $R^m \to R^m$, $(r_1, \ldots, r_m) \to (\rho(r_1), \ldots, \rho(r_m))$. Let α^m denote the map $R^m \to P^m$, $(r_1, \ldots, r_m) \to (\alpha(r_1), \ldots, \alpha(r_m))$. Then it is easy to check – and left to the reader – that $(\rho^m : R^m \to R^m, \alpha^m : R^m \to P^m)$ is an approximation pair for R^m. It is the diagonal extension of the original (ρ, α) pair to R^m and P^m.*

The example can be jazzed up. Instead of using the **same** (ρ, α) in each coordinate, one could use **distinct** (ρ, α)'s, as long as they are all approximation pairs. Similar to the diagonal extension of approximation pairs we have a diagonal extension of convergent approximations. And like the previous example, this one can also be jazzed up by using **distinct** convergent approximations in each coordinate.

4.2.8 *EXAMPLE Diagonal Extension of One Dimensional Convergent Approximations to Multidimensional Convergent Approximations:* *Another long title for a short process. Suppose that $\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$ is a **convergent approximation** for R, (3.1.10), and m is a positive integer. Let ${\rho_k}^m$ denote the map*

*$R^m \to R^m$, and ${\alpha_k}^m$ denote the map $R^m \to P^m$, both defined as in the previous example. Then it is easy to check – and left to the reader – that $\{({\rho_k}^m : R^m \to R^m, {\alpha_k}^m : R^m \to P^m)\}_k$ is a **convergent approximation** for R^m. It is the diagonal extension of the original convergent approximation to R^m and P^m. The extension is a strict convergent approximation if and only if the original convergent approximation is strict.*

The next two lemmas describe error control.

4.2.9 *LEMMA Error Propagation:* *Let $A \subseteq R^m$, $C \subseteq R^n$ and $op : A \to C$. Let $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$ be a **convergent bound** for op. Let $\rho : A \to A$ and $\alpha : A \to P^m$ be an **approximation pair** for A. Then for $a \in A$:*

$$\nu(op(a) - op(\rho(a))) \leq \omega(\overrightarrow{[}\ a, \alpha(a)\ \overleftarrow{]})$$

and

$$\nu(op(a) - op(\rho(a))) \leq \omega(\overrightarrow{[}\ \rho(a), \alpha(a)\ \overleftarrow{]})$$

PROOF: By (4.2.3,**Approx0**): $\nu(a - \rho(a)) \leq \alpha(a)$. Now substitute $\rho(a)$ for a' and $\alpha(a)$ for ϵ in (4.2.1,**Bound**) and the first equation follows. By (3.1.5,$\boldsymbol{\nu 1}$) $\nu(a - \rho(a)) = \nu(\rho(a) - a)$ so that $\nu(\rho(a) - a) \leq \alpha(a)$. Now substitute $\rho(a)$ for a, a for a' and $\alpha(a)$ for ϵ in (4.2.1,**Bound**) and the second equation follows after once again using (3.1.5,$\boldsymbol{\nu 1}$). ■

4.2.10 *LEMMA Error Convergence:* *Let $A \subseteq R^m$, $C \subseteq R^n$ and $op : A \to C$. Let $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$ be a **convergent bound** for op. Let $\{(\rho_k : A \to A, \alpha_k : A \to P^m)\}_k$ be a **convergent approximation** for A. Then:*

$$\omega(\overrightarrow{[}\ a, \alpha_k(a)\ \overleftarrow{]}) \to \hat{0}\ (k \to \infty)$$

and if ω is a continuous convergent bound:

$$\omega(\overrightarrow{[}\ \rho_k(a), \alpha_k(a)\ \overleftarrow{]}) \to \hat{0}\ (k \to \infty)$$

PROOF: $\alpha_k(a) \to \hat{0}\ (k \to \infty)$ and $\rho_k(a) \to a\ (k \to \infty)$ because $\{(\rho_k : A \to A, \alpha_k : A \to P^m)\}_k$ is a **convergent approximation** for A. Hence the first equation holds by (4.2.1,**Convergence**). Since $\{\rho_k(a)\}_k$ is a convergent sequence, it is a Cauchy sequence. Hence the second equation holds by (4.2.1,**Convergence**′). ■

The next theorem identifies when an element of C is $op(a)$ for $a \in A$ and identifies when $op(a)$ lies in a specified subset of C. It is a particularly useful formulation.

4.2.11 ***THEOREM:*** *Let $A \subseteq R^m$, $C \subseteq R^n$ and $op : A \to C$. Let $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$ be a* ***continuous convergent bound*** *for op. Let $\{(\rho_k : A \to A, \alpha_k : A \to P^m)\}_k$ be a* ***convergent approximation*** *for A.*

a. *For $c \in C$, $c = op(a)$ for $a \in A$ iff $\exists M$ s.t. $k \geq M \Rightarrow$*

$$\nu(c - op(\rho_k(a))) \leq \omega(\overrightarrow{[}\ \rho_k(a), \alpha_k(a)\ \overleftarrow{]})$$

If the convergent approximation $\{\rho_k\}$, $\{\alpha_k\}$ is a strict convergent approximation, the above formula holds for all k.

b. *For a subset S of C, $op(a) \in S$ iff $\exists s \in S$ and M such that $k \geq M \Rightarrow$*

$$\nu(s - op(\rho_k(a))) \leq \omega(\overrightarrow{[}\ \rho_k(a), \alpha_k(a)\ \overleftarrow{]})$$

If the convergent approximation $\{\rho_k\}$, $\{\alpha_k\}$ is a strict convergent approximation, the above formula holds for all k.

PROOF: **a.** ($\Rightarrow$) follows from Lemma (4.2.9) if M is chosen so that (ρ_k, α_k) is an approximation pair for $k \geq M$.

($\Leftarrow$) For all k, $\nu(c - op(a)) \leq \nu(c - op(\rho_k(a))) \hat{+} \nu(op(\rho_k(a)) - op(a))$ by (3.1.5,$\boldsymbol{\nu 2}$). By hypothesis $\nu(c - op(\rho_k(a)) \leq \omega(\overrightarrow{[}\ \rho_k(a), \alpha_k(a)\ \overleftarrow{]})$ for sufficiently large k and by (4.2.10), $\omega(\overrightarrow{[}\ \rho_k(a), \alpha_k(a)\ \overleftarrow{]}) \to \hat{0}$ $(k \to \infty)$. Hence, $\nu(c - op(\rho_k(a)) \to \hat{0}$ $(k \to \infty)$. Of course, $\nu(op(\rho_k(a)) - op(a)) = \nu(op(a) - op(\rho_k(a)))$ and by (4.2.9), $\nu(op(a) - op(\rho_k(a)))$ $\leq \omega(\overrightarrow{[}\ a, \alpha_k(a)\ \overleftarrow{]})$. By (4.2.10), $\omega(\overrightarrow{[}\ a, \alpha_k(a)\ \overleftarrow{]}) \to \hat{0}$ $(k \to \infty)$. Now we have that $\nu(c - op(a))$ is bounded by the sum of elements from two sequences which go to $\hat{0}$. By (3.1.1,**P3**) it follows that the sum goes to $\hat{0}$ and so $\nu(c - op(a))$ must equal 0. By (3.1.5,$\boldsymbol{\nu 0}$) it follows that $c = op(a)$.

b. This follows immediately from **a.**. ∎

4.3 Technical Results about Convergent Bounds

Like section (2.4), we recommend skipping this section until the results are called upon.

We have said that many functions, such as polynomial functions, have convergent bounds and that functions with convergent bounds form a **C4** class. We prove this in this section. The key is (4.3.3) which is the *convergent bound* version of (2.4.5) and (2.4.6). But first a needed technical lemma.

4.3.1 *LEMMA Preservation of Boundedness:* *Let $A \subseteq R^m$, $C \subseteq R^n$, and let $op : A \to C$ have a convergent bound $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$. If $\{a_k\}_k$ is a **bounded sequence** of elements of A then $\{op(a_k)\}_k$ is a **bounded sequence** of elements of C. In other words, functions with convergent bounds carry bounded sequences to bounded sequences.*

PROOF: Since $\{a_k\}_k$ is a bounded sequence, there is $q \in P^m$ and an integer M where $\nu(a_k) \leq q$ for $k \geq M$. Let us apply the **bound** condition using any element $a \in A$, a_k for a' and q for ϵ. This gives:

$$\nu(op(a_k) - op(a)) = \nu(op(a) - op(a_k)) \leq \omega(\overrightarrow{[}\ a, q\ \overleftarrow{]})$$

Hence $\nu(op(a_k)) \leq \nu(op(a_k) - op(a)) + \nu(op(a)) \leq \omega(\overrightarrow{[}\ a, q\ \overleftarrow{]}) \hat{+} \nu(op(a))$. Thus $\{op(a_k)\}_k$ is a **bounded sequence** of elements of C. ∎

4.3.2 *LEMMA Preservation of Cauchy:* *Let $A \subseteq R^m$, $C \subseteq R^n$, and let $op : A \to C$ have a continuous convergent bound $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$. If $\{a_k\}_k$ is a **Cauchy sequence** of elements of A then $\{op(a_k)\}_k$ is a **Cauchy sequence** of elements of C. In other words, functions with continuous convergent bounds carry Cauchy sequences to Cauchy sequences.*

PROOF: Suppose that $\{a_k\}_k$ is a Cauchy sequence. Let $\{\epsilon_k\}_k$ be a sequence of elements of $P \setminus \{\hat{0}\}$ which converges to $\hat{0}$. Since $\{a_k\}_k$ is a Cauchy sequence, for each ϵ_i there is an integer M_i where: $\nu(a_j - a_k) \leq \epsilon_i$ for $j, k > M_i$. Hence by the bound condition:

$$\nu(op(a_j) - op(a_k)) \leq \omega(\overrightarrow{[}\ a_j, \epsilon_j\ \overleftarrow{]}) \text{ for } j, k > M_i$$

By **Convergence′**, the sequence $\{\omega(\overrightarrow{[}\ a_j, \epsilon_j\ \overleftarrow{]})\}_j$ converges to $\hat{0}$. ∎

The following shows that functions with convergent bounds are a **C4** class of functions. However, it does more. Namely, it shows how to *construct* a convergent bound for the function which is claimed to have a convergent bound.

4.3.3 *PROPOSITION:* *Suppose that $A \subseteq R^m$, $C \subseteq R^n$, $B \subseteq R^{m'}$, $D \subseteq R^{n'}$ and $f : A \to C$, $g : B \to D$. In the following convergent bound may uniformly be replaced by continuous convergent bound or locally compact convergent bound. Let $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$ be a convergent bound for f and $\xi : \overrightarrow{[}\ B, P^{m'}\ \overleftarrow{]} \to P^{n'}$ be a convergent bound for g. Then:*

a. $f \times g : A \times B \to C \times D$, $(a, b) \to (f(a), g(b))$ has convergent bound:

$$\overrightarrow{[}\ A \times B, P^{m+m'}\ \overleftarrow{]} \to P^{n+n'}$$

$$\overrightarrow{[}\ a \times b, \underline{q}\ \overleftarrow{]} \to (\omega(\overrightarrow{[}\ a, (q_1, \ldots, q_m)\ \overleftarrow{]}), \xi(\overrightarrow{[}\ b, (q_{m+1}, \ldots, q_{m+m'})\ \overleftarrow{]}))$$

which will be called the product convergent bound of ω *and* ξ.

b. If $D \subseteq A$ *then the composite map,* $fg : B \to C$ *has convergent bound:*

$$\overrightarrow{[}\ B, P^{m'}\ \overleftarrow{]} \to P^n$$

$$\overrightarrow{[}\ b, \underline{q}\ \overleftarrow{]} \to \omega(\overrightarrow{[}\ g(b), \xi(\overrightarrow{[}\ b, \underline{q}\ \overleftarrow{]})\ \overleftarrow{]})$$

which will be called the composite convergent bound of ω *and* ξ.

The following general set theoretic functions have the indicated convergent bounds.

c. Constant functions from A *to* C. *The constant function sending* A *to* $c \in C$, *has convergent bound:*

$$\overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$$

$$\overrightarrow{[}\ a, \underline{q}\ \overleftarrow{]} \to \hat{0}$$

d. Diagonal functions, $\Delta_\ell : A \to A^\ell$, *for each non-negative integer* ℓ, *sending elements* $a \in A$ *to* $(a, \ldots, a)$, ℓ*-times. Such functions have the convergent bound:*

$$\overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^{m\ell}$$

$$\overrightarrow{[}\ a, \underline{q}\ \overleftarrow{]} \to \Delta_\ell(\underline{q}) \in (P^m)^\ell \equiv P^{m\ell}$$

e. Projection functions such as: $\Pi_A : A \times B \to A$, $(a, b) \to a$. *Such a function has the convergent bound:*

$$\overrightarrow{[}\ A \times B, P^{m+m'}\ \overleftarrow{]} \to P^m$$

$$\overrightarrow{[}\ (a, b), \underline{q}\ \overleftarrow{]} \to (q_1, \ldots, q_m)$$

Here $\underline{q}$ *is* $(q_1, \ldots, q_m, q_{m+1}, \ldots, q_{m+\ell})$. *Similarly for projection onto* B.

f. Permutation functions such as $A^\ell \to A^\ell$, $(a_1, \ldots, a_\ell) \to (a_{e_1}, \ldots, a_{e_\ell})$ *where* $(e_1, \ldots, e_\ell)$ *is a permutation of* $(1, \ldots, \ell)$. *Such a function has the convergent bound:*

$$\overrightarrow{[}\ A^\ell, P^{m\ell}\ \overleftarrow{]} \to P^{m\ell}$$

$$\overrightarrow{[}\ (a_1, \ldots, a_\ell), (\underline{q}_1, \ldots, \underline{q}_\ell)\ \overleftarrow{]} \to (\underline{q}_{e_1}, \ldots, \underline{q}_{e_\ell}) \in (P^m)^\ell \equiv P^{m\ell}$$

Here each $\underline{q}_j$ *lies in* P^m.

g. For $c \in R^n$ *the function* $A \to A \times \{c\}$, $a \to (a, c)$.*Such a function has the convergent bound:*

$$\overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^{m+\ell}$$

$$\overrightarrow{[}\ a, \underline{q}\ \overleftarrow{]} \to (\underline{q}, \hat{0}, \ldots, \hat{0})$$

PROOF: We prove the *locally compact* case of (b) – which has a slight subtlety – and leave the rest to the reader. Let $h : B \to C$ be the composite fg and let $\zeta : \overrightarrow{[}\ B, P^{m'}\ \overleftarrow{]} \to P^n$ be the so-called *composite convergent bound of* ω *and* ξ defined in (b). Now we show that ζ is truly a locally compact convergent bound for h.

Bound: Suppose that $b, b' \in B$, $\epsilon' \in P^{m'}$ and $\nu(b - b') \leq \epsilon'$. Then since ξ is a convergent bound for g it follows that:

$$\nu(g(b) - g(b')) \leq \xi(\overrightarrow{[}\ b, \epsilon'\ \overleftarrow{]})$$

Now use $g(b)$, $g(b')$ and $\xi(\overrightarrow{[}\ b, \epsilon'\ \overleftarrow{]})$ for a, a' and ϵ in (4.2.1) expressing that ω is a convergent bound for f. This gives:

$$\nu(f(g(b)) - f(g(b'))) \leq \omega(\overrightarrow{[}\ g(b), \xi(\overrightarrow{[}\ b, \epsilon'\ \overleftarrow{]})\ \overleftarrow{]})$$

The left hand side is: $\nu(h(b) - h(b'))$. The right hand side is: $\zeta(\overrightarrow{[}\ b, \epsilon'\ \overleftarrow{]})$ giving:

$$\nu(h(b) - h(b')) \leq \zeta(\overrightarrow{[}\ b, \epsilon'\ \overleftarrow{]})$$

Thus ζ satisfies the **bound** condition for h.

Convergence″: Let $\{\epsilon'_k\}_k$ be a sequence of elements of $P^{m'}$ where $\epsilon'_k \to \hat{0}$ $(k \to \infty)$ and let $\{b_k\}_k$ be a **bounded sequence** of elements of B. We must show that $\zeta(\overrightarrow{[}\ b_k, \epsilon'_k\ \overleftarrow{]}) \to \hat{0}$ $(k \to \infty)$. Let $a_k = g(b_k)$ and let $\epsilon_k = \xi(\overrightarrow{[}\ b_k, \epsilon'_k\ \overleftarrow{]})$. By the **convergence** condition for ξ as a convergent bound for g it follows that $\epsilon_k \to \hat{0}$ $(k \to \infty)$. The subtlety referred to earlier is that we must verify that $\{a_k\}_k$ is a **bounded sequence** of elements of A in order to utilize the **convergence** condition for ω as a convergent bound for f. Lemma (4.3.1) tells us that $\{a_k\}_k$ is a **bounded sequence** of elements of A. By the **convergence** condition for ω as a convergent bound for f we have:

$$\omega(\overrightarrow{[}\ a_k, \epsilon_k\ \overleftarrow{]}) \to \hat{0}\ (k \to \infty)$$

The left hand side – i.e. what is converging to $\hat{0}$ – is: $\omega(\overrightarrow{[}\ g(b_k), \xi(\overrightarrow{[}\ b_k, \epsilon'_k\ \overleftarrow{]})\ \overleftarrow{]})$ which equals $\zeta(\overrightarrow{[}\ b_k, \epsilon'_k\ \overleftarrow{]})$. Hence have established that ζ satisfies the **convergence** condition for h. ∎

The preceding immediately gives.

4.3.4 ***SUMMARY C4-ness of Functions with Convergent Bounds:*** *The class of functions with convergent bounds (or continuous convergent bounds or locally compact convergent bounds) from subsets of R^m to subsets of R^n for various m's and n's is tb C4, (2.4.6).*

Because (4.3.3) show how to construct convergent bounds we have:

4.3.5 ***SUMMARY Constructive Convergent Bounds:*** *Let $\mathcal{V}$ be a set of functions mapping some subsets of R^m to subsets of R^n for various m's and n's. Assume that for each $A \subseteq R^m$, $C \subseteq R^n$ and map $f : A \to C$ in $\mathcal{V}$ there is an explicit convergent bound ω_f for f. If $B \subseteq R^{m'}$, $D \subseteq R^{n'}$ and $g : B \to D$ is a map in $C4(\mathcal{V})$ then the constructions in (4.3.3) give an explicit convergent bound, ω_g, for g. The same holds for continuous convergent bounds and locally compact convergent bounds mutatis mutandis.*

Turning this around:

4.3.6 ***DEFINITION Constructive Convergent Bounds:*** *Suppose that $B \subseteq R^{m'}$ and $D \subseteq R^{n'}$. A function $g : B \to D$ is said to have a constructive convergent bound if $g \in C4(\mathcal{V})$ for a set of functions $\mathcal{V}$ mapping some subsets of R^m to subsets of R^n for various m's and n's and for each $A \subseteq R^m$, $C \subseteq R^n$ and map $f : A \to C$ in $\mathcal{V}$ there is an explicit convergent bound for f. In the preceding convergent bound may uniformly be replaced by continuous convergent bound or locally compact convergent bound.*

In section (4.2) we gave explicit convergent bounds for the functions op_{unary-}, op_+, op_-, $op_\times$, op_{q+}, op_{+q}, $op_{q\times}$ and $op_{\times q}$. Let $\mathcal{V}$ denote the set consisting of:

$$\{op_{unary-}\} \cup \{op_+\} \cup \{op_-\} \cup \{op_\times\} \cup \{op_{q+}\}_{q\in R} \cup \{op_{+q}\}_{q\in R} \cup \{op_{q\times}\}_{q\in R} \cup \{op_{\times q}\}_{q\in R}$$

Then $C4(\mathcal{V})$ is $\mathcal{POLY}$, the *polynomial functions*, defined at (2.4.10). Thus we have:

4.3.7 ***Polynomial Functions Have Constructive Locally Compact Convergent Bounds.***

Previously, we distinguished various classes of structured functions on $\mathcal{S}(R; X)$ by the stipulation that their coefficient functions (2.4.3) be of that class.[35]

4.3.8 ***DEFINITION Functions on $\mathcal{S}(R; X)$ with Constructive Convergent Bounds:*** *Let $S, T \subseteq \mathcal{S}(R; X)$. A structured function $\varphi : S \to T$ is said to have constructive convergent bounds if for each $z \in {}^*S$ the function $\kappa\varphi\overline{z} : \kappa S_z \to \kappa T_{\varphi(z)}$ has a constructive convergent bound. Again, in the preceding convergent bound may uniformly be replaced by continuous convergent bound or locally compact convergent bound.*

The condition of having some form of constructive convergent bound is required of functions to stabilize algorithms.[36]

[35] This is done in the second item of (2.4.8).

[36] But we still have to worry about the predicates.

4.3.9 *DEFINITION $\mathcal{CCB}$ Algorithms:* *An algorithm $\mathcal{A}$ is called a constructive convergent bound algorithm or $\mathcal{CCB}$ algorithm if:*

- *$\mathcal{A}$ is a structured algorithm, (2.3.1).*
- *The coefficient functions (2.4.3) of $\mathcal{A}$ have constructive convergent bounds.*

If all the convergent bounds are continuous convergent bounds the algorithm is referred to as a continuous $\mathcal{CCB}$ algorithm. If all the convergent bounds are locally compact convergent bounds the algorithm is referred to as a locally compact $\mathcal{CCB}$ algorithm.

Since Algebraic Algorithms are those whose coefficient functions are polynomial and polynomial functions have locally compact constructive convergent bounds:

4.3.10 *Algebraic Algorithms have Constructive Convergent Bounds and are locally compact $\mathcal{CCB}$ algorithms.*

4.4 Rewriting Magic

Our fundamental strategy for building the sequence of algorithms $\{\mathcal{A}_k\}_k$ which stabilizes $\mathcal{A}$ is to modify $\mathcal{A}$ by replacing coefficients in R with bracket coefficients from $\mathcal{BC}$ and then using arithmetic in $\mathcal{BC}$ instead of arithmetic in R. To be successful, we must be able to "eliminate" a discrete[37] set whose complement is open and where p is continuous on this open complement. This is done by way of a rewriting rule for $\mathcal{BC}$. We refer to this as "Rewriting Magic" because it is so simple yet effective it seems like magic. Theorem (4.2.11) is fundamental to verifying that everything in this section behaves as claimed but not needed for an understanding of the technique.

4.4.1 *DEFINITION S-Rewriting:* *The S-rewriting of a bracket coefficient $[e, \epsilon] \in \mathcal{BC}$ is denoted $S \bowtie [e, \epsilon]$. It lies in $\mathcal{BC}$ and is defined as:*

$$\begin{cases} [s, \hat{0}] & \text{if there is a unique } s \in S \text{ with } \nu(e - s) \leq \epsilon \\ [e, \epsilon] & \text{otherwise} \end{cases}$$

It may happen that $S \bowtie [e, \epsilon] = [e, \epsilon]$ because e is not within ϵ of any element of S or e is within ϵ of more than one element of S or $e \in S$ and $\epsilon = \hat{0}$. If S is the empty set then S-rewriting makes no changes. For now assume that there is an *oracle* announcing when the condition holds that "there is a unique $s \in S$ with $\nu(e - s) \leq \epsilon$". When S is finite, as is often the case, this can be ascertained by direct check.

[37] See (4.6.2) for the definition of discrete.

4.4.2 *DEFINITION Approximation Sequence:* *A sequence $\{E_k\}_k$, in $\mathcal{BC}$, converges to $[r, \hat{0}]$ if $(E_k)_R \to r \in R$ and $(E_k)_P \to \hat{0} \in P$. If in addition there is an integer M s.t. $k \geq M \Rightarrow (E_k)_P \geq \nu((E_k)_R - r)$ then $\{E_k\}_k$ is called an approximation sequence for r. In case $(E_k)_P \geq \nu((E_k)_R - r)$ for all k then $\{E_k\}_k$ is called a strict approximation sequence for r. Typically we speak of (strict) approximation sequences and drop the "for r". r is implicit because the sequence converges to $[r, \hat{0}]$.*

4.4.3 *NOTE:* *If $r \in R$ and $\{E_k\}_k$ is a sequence of elements of $\mathcal{BC}$ where $\{(E_k)_P\}_k$ converges to $\hat{0}$ in P and there is an integer M where $k \geq M \Rightarrow (E_k)_P \geq \nu((E_k)_R - r)$ then $\{(E_k)_R\}_k$ must converge to r. Hence, $\{E_k\}_k$ is an approximation sequence for r.*

An important way in which approximation sequences arise is from convergent approximations, (3.1.10).

4.4.4 *EXAMPLE How Approximation Sequences Arise from Convergent Approximations:* *Suppose that we have a sequence of pairs of maps: $\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$ which form a **convergent approximation** (3.1.10) for R. Let $r \in R$. Then the sequence $\{[\rho_k(r), \alpha_k]\}_k$ is an approximation sequence for r and is a strict approximation sequence for r if $\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$ is a **strict convergent approximation**.*

How does S-rewriting affect convergence and approximation? Under reasonable assumptions S-rewriting: a) does not introduce unwarranted convergence; b) preserves convergence; c) converts approximation to *finite convergence*. This is why S-rewriting stabilizes algorithms. Here is the definition of *finite convergence* and then the formal statement and proof of how S-rewriting affects convergence.

Convergence is defined for P, R, P^n, R^n and $\mathcal{S}(R; X)$ in Sections 3.1 and 3.2. There is a naive notion of convergence which does not use any sense of *nearness*. Namely, that the sequence *gets there* in a finite number of steps. It is useful and necessary to consider convergent sequences with the property that when they converge to certain pre-specified values – such as 0 or 1 in R – the convergence is actually *finite convergence*.[38]

4.4.5 *DEFINITION S-Convergence:* *Let $\{d_k\}_k$ be a sequence of elements in a set $\mathcal{T}$ where convergence is defined*[39] *and let $S \subset \mathcal{T}$. We say that $\{d_k\}_k$ is an S-convergent sequence if:*

1. *$\{d_k\}_k$ is a convergent sequence, say converging to d.*

[38] *Finite convergence* has essentially appeared with regard to continuity of predicates.

[39] For example, $\mathcal{T}$ may equal R, R^n or $\mathcal{S}(R; X)$.

2. *If $d \in S$ then the convergence is finite convergence, i.e. there exists an integer K such that $k \geq K \Rightarrow d_k = d$.*

We write $d_k \xrightarrow{S} d$ as a shorthand for "$\{d_k\}_k$ S-converges to d".

Typically when considering S-convergence, S is a discrete set (4.6.2). If S is the empty set then S-convergence is simply ordinary convergence.

4.4.6 *PROPOSITION Rewriting Magic:* *Let $\{[e_k, \epsilon_k]\}_k$ be a sequence in $\mathcal{BC}$.*

a. *If $\{\epsilon_k\}_k$ converges to $\hat{0}$ and $\{S \bowtie [e_k, \epsilon_k]\}_k$ converges to $[r, \hat{0}]$ then $\{[e_k, \epsilon_k]\}_k$ converges to $[r, \hat{0}]$.*

Now assume that $\{[e_k, \epsilon_k]\}_k$ converges to $[r, \hat{0}]$.

b. *$\{S \bowtie [e_k, \epsilon_k]\}_k$ converges to $[r, \hat{0}]$.*

c. *If $\{[e_k, \epsilon_k]\}_k$ is an approximation sequence and r is a discrete point of S then $\{S \bowtie [e_k, \epsilon_k]\}_k$ finitely converges to $[r, \hat{0}]$.*

PROOF: To simplify the notation in the proof let $[f_k, \phi_k] = S \bowtie [e_k, \epsilon_k]$ for each k.

a. Since $\epsilon_k \to \hat{0}$ by assumption, we only have to show that $e_k \to r$. We claim that:

4.4.7 $$\nu(r - e_k) \leq \nu(r - f_k) \hat{+} \epsilon_k$$

This is because f_k is either e_k or an element $s_k \in S$ where $\nu(e_k - s_k) \leq \epsilon_k$. If $f_k = e_k$ then $\nu(r - e_k) = \nu(r - f_k) = \nu(r - f_k) \hat{+} \hat{0} \leq \nu(r - f_k) \hat{+} \epsilon_k$ by (3.1.1,**P0**) and (3.1.1,**P1**). If $f_k = s_k$ then $\nu(r - e_k) = \nu((r - f_k) + (s_k - e_k)) \leq \nu(r - f_k) \hat{+} \nu(s_k - e_k) \leq \nu(r - f_k) \hat{+} \epsilon_k$ by (3.1.5,***ν*1**), (3.1.5,***ν*2**) and (3.1.1,**P1**). $\nu(r - f_k) \to \hat{0}$ by definition of $f_k \to r$ and $\epsilon_k \to \hat{0}$ by assumption. (3.1.1,**P3**) then implies that $\nu(r - f_k) \hat{+} \epsilon_k \to \hat{0}$. Together with (4.4.7) this implies that $\nu(r - e_k) \to \hat{0}$ and so $e_k \to r$.

b. We claim that:

4.4.8 $$\nu(r - f_k) \leq \nu(r - e_k) \hat{+} \epsilon_k$$

This is because f_k is either e_k or an element $s_k \in S$ where $\nu(e_k - s_k) \leq \epsilon_k$. If $f_k = e_k$ then $\nu(r - f_k) = \nu(r - e_k) = \nu(r - e_k) \hat{+} \hat{0} \leq \nu(r - e_k) \hat{+} \epsilon_k$ by (3.1.1,**P0**) and (3.1.1,**P1**). If $f_k = s_k$ then $\nu(r - f_k) = \nu((r - e_k) + (e_k - s_k)) \leq \nu(r - e_k) \hat{+} \nu(e_k - s_k) \leq \nu(r - e_k) \hat{+} \epsilon_k$ by (3.1.5,***ν*2**) and (3.1.1,**P1**). $\nu(r - e_k) \to \hat{0}$ by definition of $e_k \to r$ and $\epsilon_k \to \hat{0}$ by assumption. (3.1.1,**P3**) then implies that $\nu(r - e_k) \hat{+} \epsilon_k \to \hat{0}$. Together with (4.4.8) this implies that $\nu(r - f_k) \to \hat{0}$ and so $f_k \to r$.

ϕ_k is either ϵ_k or $\hat{0}$. Hence $\phi_k \leq \epsilon_k$. Then $\epsilon_k \to \hat{0} \Rightarrow \phi_k \to \hat{0}$.

c. Since $\epsilon_k \to \hat{0}$, (3.1.1,**P3**) implies that $\epsilon_k \hat{+} \epsilon_k \to \hat{0}$. Since r is a discrete point of S there is an integer M_1 such that $k \geq M_1 \Rightarrow$ there is no $s \in S$ with $\nu(r-s) < \epsilon_k \hat{+} \epsilon_k$ except for $s = r$. (Otherwise one may construct a sequence of elements of S converging to r.) Since $\{[e_k, \epsilon_k]\}_k$ is an approximation sequence there is an integer M_2 such that $k \geq M_2 \Rightarrow \epsilon_k \geq \nu(e_k - r)$. Let $M = \max\{M_1, M_2\}$. Say $k \geq M$ and suppose that there is $s \in S$ with $\nu(e_k - s) < \epsilon_k$. Then $\nu(r-s) = \nu((r-e_k)+(e_k-s)) \leq \nu(r-e_k) \hat{+} \nu(e_k - s) \leq \epsilon_k \hat{+} \epsilon_k$. By choice of M_1 this implies that $s = r$. Hence for $k \geq M$, r is the unique element of S with $\nu(e_k - r) < \epsilon_k$. Hence $k \geq M \Rightarrow S \bowtie [e_k, \epsilon_k] = [r, \hat{0}]$. Thus the convergence to $[r, \hat{0}]$ is finite. ■

Up to this point S-rewriting has only been with respect to R with no mention of R^n or $\mathcal{S}(R;X)$. The extension to R^n is basically a matter of replacing R by R^n as we now explain. Then we explain the extensions to $\mathcal{S}(R;X)$.

4.4.9 *DEFINITION S-Rewriting for $\mathcal{BC}[R^n, P^n]$:* *Suppose that $S \subset R^n$. The S-rewriting of a bracket coefficient: $[\underline{e}, \underline{\epsilon}] \in \mathcal{BC}[R^n, P^n]$ is denoted $S \bowtie [\underline{e}, \underline{\epsilon}]$. It lies in $\mathcal{BC}[R^n, P^n]$ and is defined as:*

$$\begin{cases} [\underline{s}, \hat{\underline{0}}] & \text{if there is a unique } \underline{s} \in S \text{ where } \underline{\nu(\underline{e} - \underline{s})} \leq \underline{\epsilon} \\ [\underline{e}, \underline{\epsilon}] & \text{otherwise} \end{cases}$$

In the same spirit elements of $(\mathcal{BC}[R,P])^n$ have a corresponding S-rewriting.

4.4.10 *DEFINITION S-Rewriting for $(\mathcal{BC}[R,P])^n$:* *Suppose that $S \subset R^n$. The S-rewriting of a bracket coefficient: $\underline{w} \in (\mathcal{BC}[R,P])^n$ is denoted: $S \bowtie \underline{w}$. It lies in $(\mathcal{BC}[R,P])^n$ and is defined as:*

$$\begin{cases} \overrightarrow{[} \ \underline{s}, \hat{\underline{0}} \ \overleftarrow{]} & \text{if there is a unique } \underline{s} \in S \text{ where } \underline{\nu(\underline{w}_R - \underline{s})} \leq \underline{w}_P \\ \underline{w} & \text{otherwise} \end{cases}$$

We could define approximation sequences for $\mathcal{BC}[R^n, P^n]$ and $(\mathcal{BC}[R,P])^n$ and prove the corresponding proposition to (4.4.6). Instead we wait until the next section and do this for $\mathcal{S}(\mathcal{BC}[R,P];X)$ where it is really needed.

4.5 $\mathcal{S}(\mathcal{BC}[R,P];X)$

$\mathcal{S}(R;X)$ is defined in Section (2.2). If R is replaced by $\mathcal{BC}[R,P]$, this gives the proper technical definition of $\mathcal{S}(\mathcal{BC}[R,P];X)$. However, it is handy to think of $\mathcal{S}(\mathcal{BC}[R,P];X)$ by thinking of it as being $\mathcal{S}(R;X)$ but with the R coefficients replaced by bracket coefficients from $\mathcal{BC}[R,P]$. This section presents fundamental considerations about $\mathcal{BC}[R,P]$, especially the extension of previous concepts to $\mathcal{BC}[R,P]$.

Let us now present S-rewriting for $\mathcal{S}(\mathcal{BC}[R,P];X)$. First, here is an example.

Suppose that: $([e_1,\epsilon_1],x_1,([e_2,\epsilon_2],[e_3,\epsilon_3],([e_4,\epsilon_4],x_2))) \in \mathcal{S}(\mathcal{BC}[R,P];X)$ where the e_i's lie in R, the ϵ_i's lie in P and the x_j's lie in X. If there is a unique element $(s_1,x_1,(s_2,s_3,(s_4,x_2))) \in S$ where for each i: $\nu(e_i - s_i) \leq \epsilon_i$ then the original element is S-rewritten to: $([s_1,\hat{0}],x_1,([s_2,\hat{0}],[s_3,\hat{0}],([s_4,\hat{0}],x_2)))$. As usual, if there is not such unique element of S then S-rewriting makes no change.

With this same example let us illustrate the extension of the $\boldsymbol{b}_R$ and $\boldsymbol{b}_P$ notation to $\mathcal{S}(\mathcal{BC}[R,P];X)$.

$([e_1,\epsilon_1],x_1,([e_2,\epsilon_2],[e_3,\epsilon_3],([e_4,\epsilon_4],x_2)))_R = (e_1,x_1,(e_2,e_3,(e_4,x_2))) \in \mathcal{S}(R;X)$ and $([e_1,\epsilon_1],x_1,([e_2,\epsilon_2],[e_3,\epsilon_3],([e_4,\epsilon_4],x_2)))_P = (\epsilon_1,x_1,(\epsilon_2,\epsilon_3,(\epsilon_4,x_2))) \in \mathcal{S}(P;X)$.

Unfortunately, the precise definitions are more technical.

4.5.1 *DEFINITION $()_R$ and $()_P$ Projections for $\mathcal{S}(\mathcal{BC}[R,P];X)$:* *Suppose that $u \in \mathcal{S}(\mathcal{BC}[R,P];X)$. Let m denote $\sharp u$. Let $([e_1,\epsilon_1],\ldots,[e_m,\epsilon_m])$ be κu. Then*

$$\kappa u_R = (e_1,\ldots,e_m)$$

$$\kappa u_P = (\epsilon_1,\ldots,\epsilon_m)$$

u_R is an element of $\mathcal{S}(R;X)$ and is defined as:

$$u_R \equiv \langle *u, \kappa u_R\rangle = \langle *u,(e_1,\ldots,e_m)\rangle$$

u_P is an element of $\mathcal{S}(P;X)$ and is defined as:

$$u_P \equiv \langle *u, \kappa u_P\rangle = \langle *u,(\epsilon_1,\ldots,\epsilon_m)\rangle$$

Note that with this definition:[40]

$$*(u_R) = *u \text{ and } \kappa(u_R) = (\kappa u)_R$$

$$*(u_P) = *u \text{ and } \kappa(u_P) = (\kappa u)_P$$

As defined at (4.1.3), "$\overrightarrow{[\,,\,]}$" is the *constructor* to reassemble what "$()_R$" and "$()_P$" take apart. Now that "$()_R$" and "$()_P$" have been extended so that they can take apart elements of $\mathcal{S}(\mathcal{BC}[R,P];X)$, we must extend "$\overrightarrow{[\,,\,]}$" to put them back together again.

[40]This shows that we may simply write "κu_R" since both interpretations: $\kappa(u_R)$ and $(\kappa u)_R$ are equal. Similarly we may simply write "κu_P'.

4.5.2 *DEFINITION* "$\overrightarrow{[}\ ,\ \overleftarrow{]}$" *for* $\mathcal{S}(\mathcal{BC}[R,P];X)$: *Suppose that* $\alpha \in P^m$ *and* $a \in \mathcal{S}(R;X)$ *where* $\sharp a = m$. *Then* $\overrightarrow{[}\ a, \alpha\ \overleftarrow{]}$ *is an element of* $\mathcal{S}(\mathcal{BC}[R,P];X)$ *and is defined as:*

$$\overrightarrow{[}\ a, \alpha\ \overleftarrow{]} \equiv \langle *a, \overrightarrow{[}\ \kappa a, \alpha\ \overleftarrow{]}\rangle$$

Note that the "$\overrightarrow{[}\ \kappa a, \alpha\ \overleftarrow{]}$" makes sense because $\kappa a \in R^m$ since $\sharp a = m$. Also, $\langle *a, \overrightarrow{[}\ \kappa a, \alpha\ \overleftarrow{]}\rangle$ makes sense because $\overrightarrow{[}\ \kappa a, \alpha\ \overleftarrow{]} \in \mathcal{BC}[R,P]^m$. With this definition:

$$*(\overrightarrow{[}\ a, \alpha\ \overleftarrow{]}) = *a \text{ and } \kappa(\overrightarrow{[}\ a, \alpha\ \overleftarrow{]}) = \overrightarrow{[}\ \kappa a, \alpha\ \overleftarrow{]}$$

Here is how "$()_R$", "$()_P$" and "$\overrightarrow{[}\ ,\ \overleftarrow{]}$" interact. Suppose that $\alpha \in P^m$ and $a \in \mathcal{S}(R;X)$ where $\sharp a = m$. Then:

4.5.3
$$\overrightarrow{[}\ a, \alpha\ \overleftarrow{]}_R = a \text{ and } \overrightarrow{[}\ a, \alpha\ \overleftarrow{]}_P = \langle *a, \alpha\rangle$$

Suppose that $u \in \mathcal{S}(\mathcal{BC}[R,P];X)$ and let $m = \sharp u$. Then $m = \sharp(u_R)$ and $\kappa u_P \in P^m$. Hence the following equation makes sense.[41]

4.5.4
$$u = \overrightarrow{[}\ u_R, \kappa u_P\ \overleftarrow{]}$$

Now S-Rewriting for $\mathcal{S}(\mathcal{BC}[R,P];X)$. Roughly speaking, when this S rewriting makes a change, it replaces each *bracket* $[u_i, \epsilon_i]$ which appears in u with $[s_i, \hat{0}]$, where s_i is the entry of s appearing in the same position as the *bracket* from u. Formally speaking:

4.5.5 *DEFINITION S-Rewriting for* $\mathcal{S}(\mathcal{BC}[R,P];X)$: *Suppose that* $S \subset \mathcal{S}(R;X)$. *The* S*-rewriting of an element* $u \in \mathcal{S}(\mathcal{BC}[R,P];X)$ *is denoted* $S \bowtie u$. *It lies in* $\mathcal{S}(\mathcal{BC}[R,P];X)$ *and is defined as follows: If there is a* ***unique*** *element* $s \in S$ *satisfying 1 and 2 below then* $S \bowtie u$ *is defined to be:* $\overrightarrow{[}\ s, \hat{0}\ \overleftarrow{]}$. *Otherwise* $S \bowtie u$ *is simply* u, *i.e. there is no change. Conditions 1 and 2 are:*

1. $*u = *s$.

This implies that $\sharp u = \sharp s$. *Let* ℓ *denote this integer. Then* $\kappa u \in \mathcal{BC}[R,P]^\ell$ *so that* $(\kappa u)_R, \kappa s \in R^\ell$ *and* $(\kappa u)_P \in P^\ell$.

2. $\nu((\kappa u)_R - \kappa s) \leq (\kappa u)_P$.

[41] It is also true and is the $\mathcal{S}(\mathcal{BC}[R,P];X)$ generalization of (4.1.4).

In the same family of ideas we extend the notion of *approximation sequence* defined at (4.4.2) to $\mathcal{S}(\mathcal{BC}[R,P];X)$.

4.5.6 *DEFINITION Approximation Sequences for $\mathcal{S}(\mathcal{BC}[R,P];X)$:* *Suppose that $a \in \mathcal{S}(R;X)$ and $\{a_k\}_k$ is a sequence in $\mathcal{S}(\mathcal{BC}[R,P];X)$. We say that $\{a_k\}_k$ is an approximation sequence for a if there is an integer M where for $k \geq M$ the following three conditions hold:*

1. $*a = *a_k$.

This implies that $\sharp a = \sharp a_k$ for $k \geq M$. Let m denote this integer. Now $\kappa a \in R^m$ and for $k \geq M$, $\kappa a_k \in \mathcal{BC}[R,P]^m$. Thus $\kappa a_{kR} \in R^m$ and $\kappa a_{kP} \in P^m$, for $k \geq M$.

2. Starting with $k \geq M$: $\kappa a_{kP} \to \hat{0}$ $(k \to \infty)$.

3. $\nu(\kappa a_{kR} - \kappa a) \leq \kappa a_{kP}$.

Typically we speak of approximation sequences and drop the "for a". a is implicit because the sequence $\{a_k\}_k$ converges to $\overrightarrow{[}\ a, \hat{0}\ \overleftarrow{]} \in \mathcal{S}(\mathcal{BC}[R,P];X)$. An approximation sequence is called a strict approximation sequence if the conditions hold for all k.

(4.4.4) showed how convergent approximations for R yield approximation sequences for any $r \in R$. The following construction shows how a convergent approximation for R yields approximation sequences in $\mathcal{S}(\mathcal{BC}[R,P];X)$ for every element of $\mathcal{S}(R;X)$. It is also a kind of "diagonal extension" of convergent approximations for R to $\mathcal{S}(R;X)$, see (4.2.8). **This is one of the fundamental constructions in stabilizing algorithms.**

4.5.7 *EXAMPLE How Approximation Sequences Arise from Convergent Approximations:* *Suppose that we have a sequence of pairs of maps:*

$$\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$$

which form a ***convergent approximation*** *for R. We construct a sequence of maps:*

$$\{\mathcal{S}((\rho_k,\alpha_k);X) : \mathcal{S}(R;X) \to \mathcal{S}(\mathcal{BC}[R,P];X)\}_k,$$

where for any $a \in \mathcal{S}(R;X)$:

$$\{\mathcal{S}((\rho_k,\alpha_k);X)(a)\}_k$$

is an approximation sequence for a.

Let us define: $\mathcal{S}((\rho_k,\alpha_k);X)(a)$. Let m denote the integer $\sharp a$ and let

$$\{({\rho_k}^m : R^m \to R^m, {\alpha_k}^m : R^m \to P^m)\}_k$$

be the diagonal extension of $\{(\rho_k, \alpha_k)\}_k$*, (4.2.8). Then* $\kappa a \in R^m$ *as is* ${\rho_k}^m \kappa a$*.* ${\alpha_k}^m \kappa a \in P^m$*.* $\mathcal{S}((\rho_k, \alpha_k); X)(a)$ *is defined as:*

$$\mathcal{S}((\rho_k, \alpha_k); X)(a) \equiv \langle *a, \overrightarrow{[}\ {\rho_k}^m \kappa a, {\alpha_k}^m \kappa a\ \overleftarrow{]} \rangle$$

It easily follows from the definitions that this is an approximation sequence for a*. For later reference we mention:*

4.5.8 $$\kappa \mathcal{S}((\rho_k, \alpha_k); X)(a)_P = {\alpha_k}^m \kappa a$$

4.5.9 ***PROPOSITION Rewriting Magic for*** $\mathcal{S}(\mathcal{BC}[R, P]; X)$***:*** *Let* $\{a_k\}_k$ *be a sequence in* $\mathcal{S}(\mathcal{BC}[R, P]; X)$ *and let* $a \in \mathcal{S}(R; X)$*. Let* m *denote* $\sharp a$*.*

a. *If* $\{\kappa {a_k}_P\}_k$ *converges*[42] *to* $\hat{0}$ *and* $\{S \bowtie a_k\}_k$ *converges to* $\overrightarrow{[}\ a, \hat{0}\ \overleftarrow{]}$ *then* $\{a_k\}_k$ *converges to* $\overrightarrow{[}\ a, \hat{0}\ \overleftarrow{]}$*.*

Now assume that $\{a_k\}_k$ *converges to* $\overrightarrow{[}\ a, \hat{0}\ \overleftarrow{]}$*.*

b. $\{S \bowtie a_k\}_k$ *converges to* $\overrightarrow{[}\ a, \hat{0}\ \overleftarrow{]}$*.*

c. *If* $\{a_k\}_k$ *is an approximation sequence and* a *is a discrete point of* S *then* $\{S \bowtie a_k\}_k$ *finitely converges to* $\overrightarrow{[}\ a, \hat{0}\ \overleftarrow{]}$*.*

PROOF: The idea is that for specific $a \in \mathcal{S}(R; X)$ the argument comes down to working in R^m, $\mathcal{BC}[R, P]^m$ and $\mathcal{BC}[R^m, P^m]$. It then comes down to componentwise reasoning and (4.5.9). There is one subtlety in (part a) which we now discuss. It may happen that for a sequence $\{a_k\}_k$ in $\mathcal{S}(\mathcal{BC}[R, P]; X)$ that $\{\kappa {a_k}_P\}_k$ converges to $\hat{0}$ without $\{a_k\}_k$ having any convergence. Consider the sequence $\{([r, \hat{0}], x_k)\}_k$ where $r \in R$, $\hat{0} \in P$ and the x_k's lie in X. For this sequence, $\kappa {a_k}_P = \hat{0}$ for all k and so $\{\kappa {a_k}_P\}_k$ certainly converges to $\hat{0}$. However, this sequence converges if and only if the x_k's eventually become constant. I.e. there is an M and $x \in X$ where $k \geq M \Rightarrow x_k = x$. In this case the sequence converges to $([r, \hat{0}], x)$. The assumption in (part a) that $\{S \bowtie a_k\}_k$ converges to $\overrightarrow{[}\ a, \hat{0}\ \overleftarrow{]}$ implies that there is an M where $k \geq M \Rightarrow *a_k = *a$ and no $*a_k$ instability can occur. The remainder is left to the reader. ∎

In (4.2.2) we presented the extension to bracket coefficients of functions with convergent bounds. These were functions from subsets of R^m to subsets of R^n for various m's and n's. At (4.3.8) we presented the notion of functions on $\mathcal{S}(R; X)$ having constructive convergent bounds. We now show how functions on $\mathcal{S}(R; X)$ with constructive convergent bounds get extended to $\mathcal{S}(\mathcal{BC}[R, P]; X)$ in the spirit of (4.2.2).

[42] The following $\hat{0}$ is $\hat{0}, \ldots, \hat{0}$ in P^m.

4.5.10 ***CONSTRUCTION Extension of Maps with Convergent Bounds to $\mathcal{S}(\mathcal{BC}[R,P];X)$:*** *Let $A, C \subseteq \mathcal{S}(R;X)$, $\varphi : A \to C$ be a structured function with constructive convergent bound, (4.3.8). We shall construct $A', C' \subseteq \mathcal{S}(\mathcal{BC}[R,P];X)$ and a structured map $\Phi : A' \to C'$ and we do this slice-by-slice.*[43] *Suppose that $z \in *A$ and suppose $\kappa A_z \subseteq R^m$. Consider the set: $\overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}$, (4.1.5), (2.4.1). Although A' is not yet defined, this set $\overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}$ will eventually be $\kappa A'_z$. For now, we use the $\overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}$'s to build up A'. We apply $\overline{z}$ to $\overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}$, (2.4.2), to obtain elements in $\mathcal{S}(\mathcal{BC}[R,P];X)$. Let ${A_z}'$ denote:*

$$\{q \in \mathcal{S}(\mathcal{BC}[R,P];X) | \exists u \in \overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]} \text{ where } q = \overline{z}(u)\ (= \langle z, u\rangle)\}$$

In other words, ${A_z}'$ can be described:

4.5.11 $$ {A_z}' = \langle z, \overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}\rangle \text{ where } \kappa A_z \subseteq R^m$$

Since $*$ takes the value z on all elements of ${A_z}'$, it follows that the ${A_z}'$'s are disjoint, for distinct z's. Finally, define A' by:

$$A' \equiv \uplus_{z \in *A} {A_z}'$$

Now that A' has been defined, observe: $*A' = *A$, $A'_z = {A_z}'$ and that $\kappa A'_z = \overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}$.

Let us illustrate some of these considerations. Suppose that $z = (*, x_1, (*, *, (*, x_2)))$ and A contains an element $\underline{a} = (a_1, x_1, (a_2, a_3, (a_4, x_2)))$, so that $z = *\underline{a}$. Then $\underline{a}$ contributes:

$$\{([a_1,p_1],[a_2,p_2],[a_3,p_3],[a_4,p_4]) = \overrightarrow{[}\ \kappa\underline{a}, (p_1,\ldots,p_4)\ \overleftarrow{]}\ | p_1,p_2,p_3,p_4 \in P\}$$

to $\overrightarrow{[}\ \kappa A_z, P^4\ \overleftarrow{]} = \kappa A'_z$ and contributes:

$$\{([a_1,p_1], x_1, ([a_2,p_2],[a_3,p_3],([a_4,p_4],x_2))) = \langle z, \overrightarrow{[}\ \kappa\underline{a}, (p_1,\ldots,p_4)\ \overleftarrow{]}\rangle | p_1,p_2,p_3,p_4 \in P\}$$

to ${A_z}' = A'_z$.

For $C \subseteq \mathcal{S}(R;X)$, the set C' is defined similarly and we may use that: $*C' = *C$ and for $v \in *C$: $C'_v = {C_v}'$ and $\kappa C'_v = \overrightarrow{[}\ \kappa C_v, P^n\ \overleftarrow{]}$.

Since φ is a structured map with constructive convergent bound, for each $z \in *A$, $\kappa\varphi\overline{z}$ has a convergent bound, (4.3.8). Assume that $\kappa A_z \subseteq R^m$ and $\kappa C_{\varphi(z)} \subseteq R^n$, so

[43] See (2.4.1) – and especially the footnote there – for the *slice* view of A. (2.4.2) shows how φ is determined *slice-by-slice*.

that $\kappa\varphi\overline{z} : \kappa A_z \to \kappa C_{\varphi(z)}$. *Let* $\omega_{\kappa\varphi\overline{z}}$ *denote a convergent bound for* $\kappa\varphi\overline{z}$ *so that* $\omega_{\kappa\varphi\overline{z}} : \overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]} \to P^n$, *(4.2.1). As defined at (4.2.2),* $\kappa\varphi\overline{z}$ *extends to a map from* $\overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}$ *to* $\overrightarrow{[}\ \kappa C_{\varphi(z)}, P^n\ \overleftarrow{]}$. *Using the formal notation of (4.2.2), this map would be denoted:* $\kappa\varphi\overline{z}_{\omega_{\kappa\varphi\overline{z}}}$. *We abbreviate this to:* $\kappa\varphi\overline{z}_\omega$. *Thus*

$$\kappa\varphi\overline{z}_\omega : \overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]} \to \overrightarrow{[}\ \kappa C_{\varphi(z)}, P^n\ \overleftarrow{]}$$

and since $\overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]} = \kappa A'_z$ *and* $\overrightarrow{[}\ \kappa C_{\varphi(z)}, P^m\ \overleftarrow{]} = \kappa C'_{\varphi(z)}$ *we have:*

$$\kappa\varphi\overline{z}_\omega : \kappa A'_z \to \kappa C'_{\varphi(z)}$$

Since $*A = *A'$ *and* $*C = *C'$ *we have that*

$$Struc_\varphi : *A' \to *C'$$

Use $Struc_\varphi$ *as the map Struc in (2.4.4). Use* $\kappa\varphi\overline{z}_\omega$ *as the map* $Coef(z)$ *in (2.4.4). Then by (2.4.4) there is a unique structured map* $\Phi : A' \to C'$ *with* $Struc_\Phi = Struc = Struc_\varphi$ *and* $\kappa\Phi\overline{z} = \kappa\varphi\overline{z}_\omega$ *for* $z \in *A' = *A$.

Φ *is the desired extension of* φ.

4.5.12 ***NOTE*** *for later reference that with the notation in (4.5.10) if* $E \in \mathcal{S}(\mathcal{BC}[R, P]; X)$ *it is easily verified that* $E_R \in A$ *if and only if* $E \in A'$.

4.5.13 ***PROPOSITION Approximation Sequences Mapping to Approximation Sequences:*** *Let* $A, C \subseteq \mathcal{S}(R; X)$. *Let* $\varphi : A \to C$ *be a structured function with continuous constructive convergent bound. Let* $\Phi : A' \to C'$ *be the structured map just constructed in (4.5.10). Suppose that* $\{E_k\}_k$ *is a sequence of elements of* $\mathcal{BC}$ *which is an approximation sequence for* $a \in \mathcal{S}(R; X)$ *and suppose that there is an integer* M *where* $k \geq M \Rightarrow E_k \in A'$. *Then the sequence* $\{\Phi(E_k)\}_{k \geq M}$ *is an approximation sequence for the element* $\varphi(a)$ *of* C.

PROOF: $\{(E_k)_R\}_k$ converges to a because $\{E_k\}_k$ is an *approximation sequence* for a. Hence, there is an integer N where $k \geq N \Rightarrow *(E_k)_R = *a$. Then $E_k \in A'_{*a}$ for $k \geq \max\{M, N\}$. For the duration of the proof assume that $k \geq \max\{M, N\}$. Since Φ is a structured function, and $*\Phi(a) = Struc_\varphi(*a)$, it follows that $*\Phi(E_k) = Struc_\varphi(*a)$. Hence it remains to show that $\{\kappa\Phi(E_k)\}_{k \geq \max\{M,N\}}$ is an *approximation sequence* for $\kappa\varphi(a)$. By (4.4.3) it suffices to show that $\{\kappa\Phi(E_k)_P\}_{k \geq \max\{M,N\}}$ converges to $\hat{0}$ and that there is an integer $L \geq \max\{M, N\}$ where $k \geq L \Rightarrow$

$$\kappa\Phi(E_k)_P \geq \nu(\kappa\Phi(E_k)_R - \kappa\varphi(a))$$

$\kappa\varphi\overline{*a}$ has a convergent bound – call it ω – because φ is a structured map with constructive convergent bound, (4.3.8). Assume that $\kappa A_{*a} \subseteq R^m$ and $\kappa C_{\varphi(*a)} \subseteq R^n$, so that $\kappa\varphi\overline{*a} : \kappa A_{*a} \to \kappa C_{\varphi(*a)}$ and $\omega : \overrightarrow{[}\ \kappa A_{*a}, P^m\ \overleftarrow{]} \to P^n$, (4.2.1). Then

$$\kappa\Phi(E_k) = \overrightarrow{[}\ \kappa\varphi\overline{*a}(\kappa(E_k)_R), \omega(\kappa E_k)\ \overleftarrow{]}$$

$\{\kappa(E_k)_R\}_k$ converges to κa – and so is a Cauchy sequence – because $\{E_k\}_k$ is an *approximation sequence* for a. $\{\kappa(E_k)_P\}_k$ converges to $\hat{0}$, also because $\{E_k\}_k$ is an *approximation sequence*. Hence $\{\omega(\kappa E_k)\}_k$ converges to $\hat{0}$ by (4.2.1,**Convergence′**). This shows that $\{\kappa\Phi(E_k)_P\}_{k\geq\max\{M,N\}}$ converges to $\hat{0}$.

There is $L \geq \max\{M, N\}$ where $k \geq L \Rightarrow \kappa(E_k)_P \geq \nu(\kappa(E_k)_R - \kappa a)$ because $\{E_k\}_k$ is an *approximation sequence* for a. Thus $k \geq L \Rightarrow \omega(\kappa E_k) = \omega(\overrightarrow{[}\ \kappa(E_k)_R, \kappa(E_k)_P\ \overleftarrow{]}) \geq$

$$\nu(\kappa\varphi\overline{*a}(\kappa(E_k)_R) - \kappa\varphi\overline{*a}(\kappa a))$$

because ω is a continuous convergent bound for $\kappa\varphi\overline{*a}$. This concludes the proof since:

$$\omega(\kappa E_k) = \kappa\Phi(E_k)_P$$

$$\kappa\varphi\overline{*a}(\kappa(E_k)_R) = \kappa\Phi(E_k)_R$$

$$\kappa\varphi\overline{*a}(\kappa a) = \kappa\varphi(a)$$

■

4.6 Practical Topology

In vocational schools "Practical Mathematics" refers to the minimal amount of arithmetic needed for daily survival. Very minimal, about what is needed to balance a checkbook. Here "Practical Topology" refers to the minimal amount of point set topology used in this paper. Again very minimal. Moreover, definitions are phrased in the most restricted or useful way for this paper. There are no proofs in this section.

The concept of convergence in $\mathcal{S}(R; X)$, naturally leads to the concept of open and closed sets and continuity of functions and predicates.

4.6.1 *DEFINITION Interior, Open and Closed:* *Let $\mathcal{X}$ denote R^m for any positive integer m or $\mathcal{S}(R; X)$. Assume that $A \subseteq \mathcal{X}$. An element $a \in A$ is an interior point (element) of A if for every sequence $\{E_k\}_k$ of elements of $\mathcal{X}$ which converges to a there is an integer M[44] where $k \geq M \Rightarrow E_k \in A$. A is an open subset of $\mathcal{X}$ if every element of A is an interior point of A. A set is a closed subset of $\mathcal{X}$ if its complement is open.*

[44]The integer M depends upon the specific sequence $\{E_k\}_k$.

The following are easily verified:

- The empty set and $\mathcal{X}$ itself are each an open and closed subset of $\mathcal{X}$.
- The union of open sets is open, the intersection of closed sets is closed.
- For the case $\mathcal{X} = \mathcal{S}(R; X)$, a is an *interior point* of A if and only if κa is an *interior point* of κA_{*a}.
- For the case $\mathcal{X} = \mathcal{S}(R; X)$, A is an *open* or *closed* subset of $\mathcal{X}$ if and only if κA_z is an *open* or *closed* subset of R^m for each $z \in {}*A$.
- In terms of sequences, A is closed if and only if for every convergent sequence $\{E_k\}_k$ of elements of A the sequence converges to an element of A.
- The set of interior points of A is an open subset of $\mathcal{X}$ and is the largest open subset of $\mathcal{X}$ lying in A.[45]
- Let $\overline{A}$ denote the set of points in X to which sequences of elements of A converge.[46] $\overline{A}$ is a closed subset of X and is the smallest subset of X containing A. $\overline{A}$ is called the closure of A.

4.6.2 *DEFINITION Isolated and Discrete:* *Let $\mathcal{X}$ denote R^m for any positive integer m or $\mathcal{S}(R; X)$. Assume that $A \subseteq \mathcal{X}$. An element $a \in A$ is an isolated point of A if for every sequence $\{a_k\}_k$ of elements of A which converges to a the sequence finitely converges to a. I.e. there is an integer M[47] where $k \geq M \Rightarrow a_k = a$. A is a discrete set if every element of A is an isolated point of A. In the same spirit $\hat{0}$ is an isolated point of P if for every sequence $\{p_k\}_k$ of elements of P which converges to $\hat{0}$ the sequence finitely converges to $\hat{0}$.*

[45] Actually, it is a bit tricky to show that the set of interior points of A forms an open set. Suppose that $\{E_k\}_k$ is a sequence converging to a and there is no integer M where $k \geq M \Rightarrow E_k$ is an interior point of A. Then there is an infinite subsequence where none of the elements is an interior point of A. Restrict to – and renumber – the subsequence, so that we may assume that $\{E_k\}_k$ is a sequence converging to a and none of the E_k's is an interior point of A. If there is k with $E_k = a$, it immediately follows that a is not an interior point of A and we are done. So we assume that $E_k \neq a$ for all k. Now suppose that $\mathcal{X} = R^m$. For each E_k there is E'_k in the complement to A where $\nu(E_k - E'_k) < \nu(E_k - a)$. Then $\{E'_k\}_k$ is a sequence of elements in the complement to A converging to a. Hence, a does not lie in the interior of A. For the case $\mathcal{X} = \mathcal{S}(R; X)$ one first restricts to a subsequence of the E_k's where ${}*E_k = {}*a$ and then one works with $\{\kappa E_k\}_k$ converging to κa as in the case $\mathcal{X} = R^m$.

[46] I.e. the sequences must lie wholly within A but they may converge to points of X outside of A. $A \subseteq \overline{A}$ because for any element of A, the sequence consisting of just that point is a sequence lying in A which converges to the given point.

[47] The integer M depends upon the specific sequence $\{a_k\}_k$.

It is easily verified that for the case $\mathcal{X} = \mathcal{S}(R;X)$, a is an isolated point of A if and only if κa is an isolated point of κA_{*a}.

Discreteness may be characterized in terms of S-convergence. Suppose that $S \subseteq \mathcal{S}(R;X)$. S is discrete if and only if every convergent sequence $\{s_k\}_k$ of elements of S is S-convergent. This is not quite as trivial as it first appears because a sequence of elements of S may converge to an element outside of S. Consider the case where S is the set of positive rational numbers of the form $1/n$ for positive integers n. The descending sequence $\{1, 1/2, 1/3, 1/4, \ldots\}$ of elements of S converges to 0 and the convergence is **not** finite convergence. However, $0 \notin S$ and so this is still S-convergence. S is a discrete subset of the real numbers.

4.6.3 *DEFINITION Continuity of Functions:* *Suppose that α is a map from A to C where $A \subseteq R^M$ and $C \subseteq R^N$ or $A, C \subseteq \mathcal{S}(R;X)$. α is continuous at $a \in A$ if $\{\alpha(a_k)\}_k$ converges to $\alpha(a)$ for any sequence $\{a_k\}_k$ of elements of A which converges to a.*[48] *α is continuous on A if it is continuous at all $a \in A$.*

It is easy to verify that for the case $A, C \subseteq \mathcal{S}(R;X)$ and α a structured map, α is continuous if and only if $\kappa\alpha\overline{z}$ is continuous for each $z \in {}^*A$.

4.6.4 *PROPOSITION Continuity of Functions with Convergent Bounds:* *Suppose that α is a map from A to C where $A \subseteq R^M$ and $C \subseteq R^N$ and α has a convergent bound or $A, C \subseteq \mathcal{S}(R;X)$ and α has a constructive convergent bound. Then α is continuous on A.*

We lied when we said there are no proofs in this section, but this is the only one, except for the proof of (4.6.13).

PROOF: Suppose that $A \subseteq R^m$, $C \subseteq R^n$, $\alpha : A \to C$ has convergent bound $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$. Suppose that $\{a_k\}_k$ is a sequence of elements of A converging to $a \in A$. We must show that $\{\alpha(a_k)\}_k$ converges to $\alpha(a) \in C$. To do this we must show that for $\epsilon \in P^n$ – where no component of ϵ is equal to $\hat{0}$ – there is M where for $k \geq M$, $\nu(\alpha(a) - \alpha(a_k)) < \epsilon$. Since $a_k \to a$ in R^m, the sequence $\{\nu(a - a_k)\}_k$ converges to $\hat{0}$ in P^m. Let $\delta_k \equiv \nu(a - a_k)$, By the convergence condition, (4.2.1):

$$\omega(\overrightarrow{[}\ a, \delta_k\ \overleftarrow{]}) \to \hat{0}\ (k \to \infty)$$

Hence there is M where for $k \geq M$:

$$\omega(\overrightarrow{[}\ a, \delta_k\ \overleftarrow{]}) < \epsilon$$

[48] Points of A where α is not continuous are called *discontinuity points* of α.

By the bound condition, it follows that for $k \geq M$:

$$\nu(\alpha(a) - \alpha(a_k)) \leq \omega(\overrightarrow{[}\ a, \delta_k\ \overleftarrow{]}) < \epsilon$$

The case $A, C \subseteq \mathcal{S}(R; X)$ is left to the reader. ∎

Continuity of predicates is with respect to {TRUE, FALSE} having the discrete topology. In other words:

4.6.5 ***DEFINITION Continuity of Predicates:*** *Suppose that π is a map from A to $\{TRUE, FALSE\}$ where $A \subseteq R^M$ or $A \subseteq \mathcal{S}(R; X)$. π is continuous at $a \in A$ if for any sequence $\{a_k\}_k$ of elements of A which converges to a there is an M such that $k \geq M \Rightarrow \pi(a_k) = \pi(a)$.*[49] *$\pi$ is continuous on A if it is continuous at all $a \in A$.*

4.6.6 ***DEFINITION DUO Functions and Predicates:*** *Let γ be a function or predicate defined on A where $A \subseteq \mathcal{X}$ for $\mathcal{X} = R^m$ or $\mathcal{S}(R; X)$. γ is a DUO*[50] *function or predicate if A has a decomposition:*[51] *$A = S \cup Q$ where S is a discrete subset of $\mathcal{X}$, Q is an open subset of $\mathcal{X}$, and γ is continuous on Q. Such a decomposition of A is called a DUO decomposition of A for γ. If γ is a function, it is a DUO function with convergent bound if A has a decomposition: $A = S \cup Q$ where S is a discrete subset of $\mathcal{X}$, Q is an open subset of $\mathcal{X}$, and γ has a convergent bound on Q. I.e. the restriction of γ to Q has a convergent bound. γ is a D function or predicate if A is a discrete subset of $\mathcal{X}$, i.e. Q is empty. γ is an O function*[52] *or predicate if A is an open subset of $\mathcal{X}$, and γ is continuous on A. I.e. S is empty. If γ is a function, it is an O function with convergent bound if A is an open subset of X and γ has a convergent bound on Q.*

4.6.7 ***EXAMPLE:*** *The predicate $IS\_ZERO(X)$ is a DUO predicate. $R \setminus \{0\}$ is the open set Q on which it is continuous and $\{0\}$ is a discrete set S where it need not be continuous.*[53]

4.6.8 ***EXAMPLE:*** *If the real numbers have their usual topology then the INTEGER_PART_OF function is a DUO function. Let S be the integers and let Q be the reals which are not integers.*

[49] Points of A where π is not continuous are called *discontinuity points* of π.

[50] *DUO* stands for "Discrete Union Open".

[51] We emphasize that the union is not necessarily a disjoint union.

[52] There is a notion of *open* functions which is that the function carries open sets to open sets. This is not related to a function being an O function.

[53] Typically $IS\_ZERO(X)$ is not continuous at 0 but it is easy to construct examples of P and ν where $IS\_ZERO(X)$ is continuous at 0.

4.6.9 ***EXAMPLE:*** *Polynomial functions are O functions. In fact polynomial functions have O convergent bounds.*

4.6.10 ***DEFINITION $\mathcal{C}$, $\mathcal{D}$, Ω and Δ Sets of Functions and Predicates:*** *Suppose that γ is a function or predicate defined on A where $A \subseteq \mathcal{X}$ for $\mathcal{X} = R^m$ or $\mathcal{S}(R; X)$. Let $\mathcal{C}(\gamma)$ be the set of points where γ is continuous and $\mathcal{D}(\gamma)$ be the set of points where γ is not continuous. Namely $\mathcal{D}(\gamma)$ denotes the complement to $\mathcal{C}(\gamma)$ in A. Considering $\mathcal{C}(\gamma) \subseteq \mathcal{X}$, $\Omega(\gamma)$ denotes the set of interior points of $\mathcal{C}(\gamma)$. $\Delta(\gamma)$ denotes the complement to $\Omega(\gamma)$ in A.*[54]

4.6.11 ***EXAMPLE:*** *This example illustrates the importance of $\Delta(\gamma)$. More specifically, in this example instability arises from the discrete nature of the domain rather than any lack of continuity of the predicate on its domain. This algorithm (fragment) has R as the real numbers. For some purposes in trigonometry, one wishes to know if an angle is 2π or an integer multiple of 2π.*

Initialize:	$(angle)$
.	.
step_i:	$X = angle/\pi$
step_*$(i+1)$:*	*if* $IS\_EVEN\_INTEGER(X)$ *goto* $\cdots$
.	.

The predicate $IS\_EVEN\_INTEGER$ has domain equal to the integers. The predicate is continuous on this set because the integers are discrete. The instability of the algorithm comes from the fact that the domain of the predicate is not an open set in the real numbers. I.e. the difficulty comes from the discrete nature of the domain not lack of continuity of the predicate on the domain. If a sequence in R converges to an integer – and does not reach the integer in a finite number of steps – then it does not eventually get in and stay in the domain of $IS\_EVEN\_INTEGER$. This is what causes instability of the algorithm.

Of course there is an artificial way to cause a function to be defined everywhere. Suppose we have function or predicate F mapping to a set Z. Let us add the *isolated point* $NOT\text{-}DEFINED$ to Z. Wherever the function or predicate is not defined we now define it by mapping the point to $NOT\text{-}DEFINED$. Now our function or predicate is defined everywhere. Call this new defined-everywhere function G. Then $\Delta(F)$ is the intersection of the domain of F with $\mathcal{D}(G)$.

[54] Beware of the subtlety. Consider when A is the X-axis in $\mathcal{X}$, the X–Y plane. Suppose γ is continuous, so that $\mathcal{C}(\gamma) = A$. The interior of the X-axis as a subset of the X–Y plane is empty! I.e. $\Omega(\gamma) = \emptyset$ and $\Delta(\gamma) = A$.

Since complementation reverses inclusion: $\Delta(\gamma) \supseteq \mathcal{D}(\gamma)$. In general $\Delta(\gamma)$ contains both points where γ is continuous and points where γ is not continuous. The extreme cases where $\Delta(\gamma)$ only contains points where γ is continuous or only contains points where γ is not continuous is characterized in the following, which is not difficult to verify:

4.6.12 *PROPOSITION Extreme Cases for $\Delta(\gamma)$:* *Let γ be a function or predicate defined on A. $\mathcal{C}(\gamma) = A$ if and only if $\mathcal{D}(\gamma) = \emptyset$.*[55] *$\mathcal{C}(\gamma)$ is an open set in $\mathcal{X}$ if and only if $\Delta(\gamma) = \mathcal{D}(\gamma)$.*

$\Delta(\gamma)$ is key to characterizing *DUO* functions and predicates.

4.6.13 *PROPOSITION Characterization of DUO:* *Suppose that γ is a function or predicate defined on $A \subseteq \mathcal{X}$ for $\mathcal{X} = R^m$ or $\mathcal{S}(R; X)$. γ is a DUO function or predicate if and only if $\Delta(\gamma)$ is a discrete set. In this case $A = \Delta(\gamma) \uplus \Omega(\gamma)$ is a DUO decomposition of A for γ.*[56] *If $A = S \cup Q$ is any DUO decomposition of A for γ then $Q \subseteq \Omega(\gamma)$ and $\Delta(\gamma) \subseteq S$.*

PROOF: $\Omega(\gamma)$ is an **open** subset of $\mathcal{X}$ on which γ is continuous. Also, A is the disjoint union of $\Delta(\gamma)$ and $\Omega(\gamma)$. Thus if $\Delta(\gamma)$ is discrete, $A = \Delta(\gamma) \uplus \Omega(\gamma)$ is a *DUO decomposition of A for γ* and γ is a *DUO* function or predicate.

Conversely, suppose that γ is a *DUO* function or predicate and $A = S \cup Q$ is a *DUO decomposition of A for γ*. Since, $\Omega(\gamma)$ is the largest **open** subset of $\mathcal{X}$ on which γ is continuous, it follows that $Q \subseteq \Omega(\gamma)$. This implies that $\Delta(\gamma)$ – the complement to $\Omega(\gamma)$ in A – is a subset of the complement to Q in A which is a subset of S. Hence, $\Delta(\gamma)$ is discrete, as claimed.

We have also shown that $Q \subseteq \Omega(\gamma)$ and $\Delta(\gamma) \subseteq S$. ∎

Instability of algorithms derives from $\Delta(\gamma)$ not being empty for both predicates γ and functions γ. For our main results we restrict consideration to functions having *convergent bounds*, (4.2.1) on open sets with discrete complements. Such functions are automatically continuous, (4.6.4), on the open sets. Hence, such functions are *DUO* functions. The "magic" of bracket coefficients and "rewriting" is that they enable us to obtain a kind of stability of algorithms even when the discrete *problem sets* are not empty.

4.6.14 *EXAMPLE Isolated $\Delta(\gamma)$:* *Suppose that X is the set of real numbers with its usual topology and A is the unit interval, $[0, 1]$. For $a \in A$ let $\gamma(a) = a$. Then $\Delta(\gamma)$ consists of the two end points $\{0, 1\}$ and is a discrete set. $\Delta(\gamma)$ contains no discontinuity points of γ. γ with this A is a DUO function.*

[55] Of course, $\Delta(\gamma)$ may be empty.

[56] $\uplus$ indicates that the union is a "disjoint union".

The examples of algorithms in Section (2.6) have polynomial functions which are continuous. Let us look at the discontinuity sets of the predicates in each algorithm. For this the following is useful:

4.6.15 ***PROPOSITION:*** *If γ is a predicate defined on A,*

$$\mathcal{D}(\gamma) = \overline{\gamma^{-1}(TRUE)} \cap \overline{\gamma^{-1}(FALSE)}.$$

Now the examples.

Examples of Discontinuity Sets of Predicates.

4.6.16 ***DETERMINANT:*** *The algorithm $\mathcal{DET}_n$ presented at (2.6.1) has no predicates.*

4.6.17 ***DISCRIMINANT:*** *The algorithm $\mathcal{DISC}_n$ presented at (2.6.2) has one predicate, p, with argument Answer. $p(Answer)$ is: "$IS\_ZERO(Answer)$", where $Answer \in R$. $\overline{p^{-1}(TRUE)} = \{0\}$ while $\overline{p^{-1}(FALSE)} = R$. Hence, $\mathcal{D}(p) = \{0\}$.*

4.6.18 ***POLY-POSITIVE:*** *The algorithm presented at (2.6.3) has one predicate, p, with argument X. The predicate $p(X)$ is: "$IS\_GREATER\_THAN\_ZERO(X)$", where $X \in R$. $\overline{p^{-1}(TRUE)} = \{r \in R | r \geq 0\}$ while $\overline{p^{-1}(FALSE)} = \{r \in R | r \leq 0\}$. Hence, $\mathcal{D}(p) = \{0\}$.*

4.6.19 ***REPEAT-ADD:*** *The algorithm presented at (2.6.4) has one predicate, p, with argument X. The predicate $p(X)$ is: "$IS\_GREATER\_THAN\_OR\_EQUAL\_TO\_ONE(X)$", where $X \in R$. $\overline{p^{-1}(TRUE)} = \{r \in R | r \geq 1\}$ while $\overline{p^{-1}(FALSE)} = \{r \in R | r \leq 1\}$. Hence, $\mathcal{D}(p) = \{1\}$.*

4.6.20 ***BUCHBERGER:*** *As before (2.6.5), (3.3.6) we discuss selected aspects of this algorithm. In the previous discussion of extracting leading term, two predicates appeared. One was IS_ZERO and as explained in (2), $\mathcal{D}(IS\_ZERO) = \{0\}$. The other was $p(sequence) = IS\_EMPTY(sequence)$. The empty sequence $()$ is an element of $\mathcal{S}(R; X)$. $IS\_EMPTY^{-1}(TRUE) = \{()\}$ and $\overline{IS\_EMPTY^{-1}(TRUE)} = \{()\}$. It is easily checked that if $d_m \to ()$ in $\mathcal{S}(R; X)$ then there is an M such that $m \geq M \Rightarrow d_m = ()$. Hence, $()$ is an isolated point[57] of $\mathcal{S}(R; X)$. Since $() \notin IS\_EMPTY^{-1}(FALSE)$, the fact that $()$ is isolated implies that $() \notin \overline{IS\_EMPTY^{-1}(FALSE)}$. Thus $\overline{IS\_EMPTY^{-1}(TRUE)} \cap \overline{IS\_EMPTY^{-1}(FALSE)} = \emptyset$. Hence, $\mathcal{D}(IS\_EMPTY) = \emptyset$.*

[57] See (4.6.2) for the definition of isolated.

There are a few additional considerations we wish to mention here. Consider the common predicate $ARE\_EQUAL(d, e)$ for $d, e \in U \subset \mathcal{S}(R; X)$. It may be decomposed into more elementary predicates. For example, first d and e may be checked for having the same structure. This may be done as follows. Let z be an element of X which does not appear in any of the elements of U. If necessary, enlarge X and adjoin such an element z. Let Z be a map which is defined like the map $*$ except that Z replaces elements of R in sequences by z instead of by $*$. Then Z is a polynomial structured map $Z : U \to \mathcal{S}(R; X)$ and the image of Z lies in $\mathcal{S}(R; X)_0$ a discrete[58] subset of $\mathcal{S}(R; X)$. Hence the first part of a test for equality factors through a polynomial map Z and a predicate on $\mathcal{S}(R; X)_0$. Since $\mathcal{S}(R; X)_0$ is discrete, the predicate on $\mathcal{S}(R; X)_0$ has empty discontinuity set. For d, e which pass the first test, i.e. $Z(d) = Z(e)$, the second test is simply to check whether $\kappa d = \kappa e$. Such κd and κe lie in the same R^n and can be written $\kappa d = ({d_1}', \ldots, {d_n}')$, $\kappa e = ({e_1}', \ldots, {e_n}')$ with the d_i's and e_i's in R. One can form $\kappa d - \kappa e = ({d_1}' - {e_1}', \ldots, {d_n}' - {e_n}')$. Then testing if $\kappa d = \kappa e$ reduces to testing if each ${d_i}' - {e_i}'$ is zero with IS_ZERO and an IS_EMPTY predicate for stepping through the sequence: $({d_1}' - {e_1}', \ldots, {d_n}' - {e_n}')$. As before, $\mathcal{D}(IS\_EMPTY) = \emptyset$ and $\mathcal{D}(IS\_ZERO) = \{0\}$. The bottom line is that by decomposing $ARE\_EQUAL(d, e)$ into more elementary predicates one is able to reach the case where the cumulative discontinuity set for all the predicates replacing ARE_EQUAL is simply 0.

The presence of a rich discontinuity set need not cause an algorithm to be unstable. The following algorithm takes inputs from R and the output is precisely the same as the input. Hence the algorithm is stable. The example has been designed for simplicity to illustrate the point. A complex algorithm may contain one or more program branches with respect to predicates with large discontinuity sets and they may not contribute instability to the algorithm. We do not know if such inessential predicates can be detected and removed automatically.

4.6.21 *EXAMPLE IRRELEVANT-DISCONTINUITY-SET:* *$R = \boldsymbol{R}$ and we are in the situation of the example* ***ABSOLUTE VALUE*** *(3.1.6). The inputs are elements of R.*

Initialize:	(X)
step_1:	*goto step_3 if X is rational*
step_2:	*stop* (X)
step_3:	*stop* (X)

The discontinuity set of $IS\_RATIONAL$ is R.

It should be noted that for algebraic algorithms and $\mathcal{CCB}$ algorithms, defined at (4.3.9),

[58] See (4.6.2) for the definition of discrete.

if $\Delta(p) = \emptyset$ for all predicates p in $\mathcal{A}$, then $\mathcal{A}$ is stable. Our methods provide a way to stabilize $\mathcal{A}$ if $\Delta(p) \neq \emptyset$ for some predicate p as long as $\Delta(p)$ is discrete.

5 Stability Theorems

5.1 Overview of Main Results

At this point it may be helpful if we sketch our main results before getting bogged down in technical details. To this end we begin with a basic set of hypotheses for an algorithm $\mathcal{A}$.

5.1.1 *HYPOTHESES:*

- *$\mathcal{A}$ is a continuous $\mathcal{CCB}$ algorithm, (4.3.9).*
- *The domain of the function in each "local assignment from computation" step of $\mathcal{A}$ is an open set of $\mathcal{S}(R; X)$.*
- *For the predicate $Predicate$ in each "conditional goto" step of $\mathcal{A}$ there is a discrete set $S_{Predicate}$ where the domain of $Predicate$ equals $S_{Predicate} \cup \Omega(Predicate)$.*

The hypotheses about functions having open domains and constructive convergent bounds may be weakened. We do so in a later section but until further notice we assume that the conditions in (5.1.1) are satisfied. Let us now describe a stability theorem which eventually gets put in a more general form.

We will pass from the algorithm $\mathcal{A}$ defined on (a subset of) $\mathcal{S}(R; X)$ to a new algorithm, denoted $\mathcal{BC}(\mathcal{A})$, defined on $\mathcal{S}(\mathcal{BC}[R, P]; X)$. In fact, the syntax of the original algorithm is unchanged, one simply changes the semantics. The input of $\mathcal{BC}(\mathcal{A})$ consists of bracket coefficients, i.e. elements of $\mathcal{S}(\mathcal{BC}[R, P]; X)$. The main change to the semantics is how operations are performed and how a "conditional goto" is treated. "Initialize", absolute "goto" and "stop" steps are essentially unchanged. The functions with constructive convergent bounds of a $\mathcal{CCB}$ algorithm are extended to bracket coefficients using (4.2.2). More about this when we give precise definitions.[59]

The execution of a "conditional goto" is conditional upon the result of evaluating a predicate. Thus, extending a "conditional goto" from $\mathcal{A}$ to $\mathcal{BC}(\mathcal{A})$ involves describing

[59]Of course an algorithm may utilize arithmetic operations utilizing arithmetic values which do not come from the input. For example suppose $r \in R$ is *built-in* to the algorithm $\mathcal{A}$ and at some stage $\mathcal{A}$ involves multiplication by r or addition of r to a data value. This is equivalent to applying one of the operators: op_{r+}, op_{+r}, $op_{r\times}$ or $op_{\times r}$ for which the convergent bounds were presented between (4.2.1) and (4.2.2).

how predicates are extended to bracket coefficients. The answer is to perform $S_{Predicate}$-rewriting on the arguments of the predicates of $\mathcal{BC}(\mathcal{A})$ and then apply the original predicate from $\mathcal{A}$ to the first component(s) of the bracket coefficients. I.e. a "conditional goto" step involves a predicate with arguments (from $\mathcal{BC}(\mathcal{A})$) which are now elements of $\mathcal{S}(\mathcal{BC}[R,P];X)$. Suppose these arguments are $\boldsymbol{a_1,\ldots,a_t}$. First do $S_{Predicate}$-rewriting for each of these arguments to obtain (possibly) different arguments $\boldsymbol{b_1,\ldots,b_t}$. Now apply the original predicate (from $\mathcal{A}$) to the arguments $\boldsymbol{b}_{\boldsymbol{1}R},\ldots,\boldsymbol{b}_{\boldsymbol{t}R}$ and "goto" the step indicated by that result.[60] The output of $\mathcal{BC}(\mathcal{A})$ lies in $\mathcal{S}(\mathcal{BC}[R,P];X)$.

The stability theorem concerning $\mathcal{BC}(\mathcal{A})$ follows. An example showing that the converse fails appears in (5.6.1).

5.1.2 *THEOREM $\mathcal{BC}(\mathcal{A})$-Stability:* *Suppose that $a_1,\ldots,a_m \in \mathcal{S}(R;X)$ is input data for the algorithm $\mathcal{A}$ on which $\mathcal{A}$ terminates normally. Let the output of $\mathcal{A}(a_1,\ldots,a_m)$ be $b_1,\ldots,b_n \in \mathcal{S}(R;X)$. Assume that for each a_i there is an approximation sequence $\{a'_{ik}\}_k$ of elements of $\mathcal{S}(\mathcal{BC}[R,P];X)$. Then there is N where for $k \geq N$ $\mathcal{BC}(\mathcal{A})$ terminates normally with input data $(a'_{1k},\ldots,a'_{mk})$. Let $b'_{1k},\ldots,b'_{nk} \in \mathcal{S}(\mathcal{BC}[R,P];X)$ denote the output of $\mathcal{BC}(\mathcal{A})(a'_{1k},\ldots,a'_{mk})$, for k where $\mathcal{BC}(\mathcal{A})(a'_{1k},\ldots,a'_{mk})$ terminates normally. Then each $\{b'_{j_k}\}_{k\geq N}$ is an approximation sequence for b_j.*

To utilize the $\mathcal{BC}(\mathcal{A})$ construction and Theorem (5.1.2), one coerces the output of $\mathcal{BC}(\mathcal{A})$ back to $\mathcal{S}(R;X)$.

5.1.3 *DEFINITION Output Variable:* *An output variable of the algorithm $\mathcal{A}$ is an internal, local variable which appears in the text of $\mathcal{A}$ in a "stop" step:*[61]

at step_i: "stop $(X_1,\ldots,X_n)$"	*Execution halts. The algorithm outputs the sequence consisting of the values of the X_i's.*

Suppose that for each output variable OV of $\mathcal{A}$, there is a discrete set $S_{OV} \subset \mathcal{S}(R;X)$. If no S_{OV} is specified, let S_{OV} be the empty set. The output of $\mathcal{BC}(\mathcal{A})$ is coerced to $\mathcal{S}(R;X)$ by applying S_{OV}-rewriting to the value of OV for output variables in "stop" steps and then projecting onto $\mathcal{S}(R;X)$ using $()_R$. For more detail see (5.2.1). Let us call this *coerced* algorithm $\mathcal{BC}(\mathcal{A})_R$. It has input data from $\mathcal{S}(\mathcal{BC}[R,P];X)$ and output back in $\mathcal{S}(R;X)$. The next important result is an immediate corollary to Theorem (5.1.2) or at least to the proof of Theorem (5.1.2) in Section (5.3):

[60] This is the minimal amount of S-rewriting which may be done and still achieve stabilization. It may be desirable to do S-rewriting more often, such as after every function step. If S is well chosen for the problem, such additional S-rewriting may encourage sparsity of the data carried along by the algorithm.

[61] In the "stop" step presented, all the X_i's which are internal, local variables – as opposed to elements of $\mathcal{S}(R;X)$ – are *output variable* of $\mathcal{A}$. Output variable may also appear in other steps of $\mathcal{A}$ besides a "stop" step. In fact, if they do not also appear in a "local assignment from computation" step, they will not be assigned values.

5.1.4 *COROLLARY $\mathcal{BC}(\mathcal{A})_R$-Stability:* *Suppose that $a_1, \ldots, a_m \in \mathcal{S}(R; X)$ is input data for the algorithm $\mathcal{A}$, on which it terminates normally. Assume that for each a_i there is an approximation sequence $\{a'_{ik}\}_k$ of elements of $\mathcal{S}(\mathcal{BC}[R, P]; X)$.*

Now suppose that the output of $\mathcal{A}(a_1, \ldots, a_m)$ is $b_1, \ldots, b_n \in \mathcal{S}(R; X)$. Then there is N where for $k \geq N$ the output of $\mathcal{BC}(\mathcal{A})_R(a'_{1k}, \ldots, a'_{mk})$ is $b_{1k}, \ldots, b_{nk} \in \mathcal{S}(R; X)$. For each b_j, the sequence $\{b_{j_k}\}_{k \geq N}$ converges to b_j. If b_j comes from an output variable OV then the sequence $\{b_{j_k}\}_{k \geq N}$, S_{OV}-converges to b_j.[62]

As mentioned earlier, in the construction of $\mathcal{BC}(\mathcal{A})$ and $\mathcal{BC}(\mathcal{A})_R$, no change was made to the *text* or *syntax* of the original algorithm $\mathcal{A}$. We simply changed how to interpret $\mathcal{A}$; i.e. changed the semantics. For $\mathcal{BC}(\mathcal{A})_R$, the output is coerced to back to $\mathcal{S}(R; X)$, which is independent of the original algorithm $\mathcal{A}$.

The previous results require *approximation sequences* for the input data to $\mathcal{A}$. Suppose that $\{p_k\}_k$ is a sequence of elements of P which converges to $\hat{0}$. Let

$$\{(\rho_k : R \to R, \alpha_k : R \to P)\}_k$$

be the *identity convergent approximation arising from* $\{p_k\}_k$, (3.1.12). This is a strict convergent approximation. Plug this strict convergent approximation into (4.5.7) to obtain the sequence of maps:

$$\{\mathcal{S}((\rho_k, \alpha_k); X) : \mathcal{S}(R; X) \to \mathcal{S}(\mathcal{BC}[R, P]; X)\}_k,$$

where for any $a \in \mathcal{S}(R; X)$:

$$\{\mathcal{S}((\rho_k, \alpha_k); X)(a)\}_k$$

is a strict approximation sequence for a.

5.1.5 *DEFINITION $\mathcal{A}_{p_k}$:* *Suppose that $\mathcal{A}$ is a $\mathcal{CCB}$ algorithm and $\{p_k\}_k$ is a sequence of elements of P which converges to $\hat{0}$. If $a_1, \ldots, a_m \in \mathcal{S}(R; X)$ is input data for $\mathcal{A}$ then:*

$$\{\mathcal{S}((\rho_k, \alpha_k); X)(a_1)\}_k, \ldots, \{\mathcal{S}((\rho_k, \alpha_k); X)(a_m)\}_k$$

are strict approximation sequences for $a_1, \ldots, a_m$. Let $\mathcal{A}_{p_k}(a_1, \ldots, a_m)$ denote:

$$\mathcal{BC}(\mathcal{A})_R(\mathcal{S}((\rho_k, \alpha_k); X)(a_1), \ldots, \mathcal{S}((\rho_k, \alpha_k); X)(a_m))$$

It immediately follows from Corollary (5.1.4):

[62] If, instead of coming from an output variable, b_j comes from a constant in $\mathcal{S}(R; X)$ then $\{b_{j_k}\}_{k \geq N}$ is the constant sequence where $b_{j_k} = b_j$ for each $k \geq N$.

5.1.6 *COROLLARY $\mathcal{A}_{p_k}$-Stability:* *Suppose that $a_1, \ldots, a_m \in \mathcal{S}(R;X)$ is input data for the algorithm $\mathcal{A}$. Suppose that the output of $\mathcal{A}(a_1, \ldots, a_m)$ is $b_1, \ldots, b_n \in \mathcal{S}(R;X)$. Then there is N where for $k \geq N$ the output of $\mathcal{A}_{p_k}(a_1, \ldots, a_m)$ is $b_{1k}, \ldots, b_{nk} \in \mathcal{S}(R;X)$. For each b_j, the sequence $\{b_{jk}\}_{k \geq N}$ converges to b_j. If b_j comes from an output variable OV then the sequence $\{b_{jk}\}_{k \geq N}$, S_{OV}-converges to b_j.*[63]

Typically, the sequence $\{p_k\}_k \subset P$ where $p_k \to \hat{0}$ is kept fixed. When using floating point approximation the sequence might be something like $\{1/10^k\}_k$. When the sequence is kept fixed, we may write $\mathcal{A}_k$ in place of $\mathcal{A}_{p_k}$ to avoid the extra level of subscript. Whether we write $\mathcal{A}_k$ or $\mathcal{A}_{p_k}$, here is another interpretation of what is happening. View $\{\mathcal{A}_k\}_k$ as a sequence of algorithms approximating[64] the original algorithm $\mathcal{A}$.

The preceding corollary is an approximation result because it shows that for fixed input $a_1, \ldots, a_m$: $\mathcal{A}_k(a_1, \ldots, a_m) \to \mathcal{A}(a_1, \ldots, a_m)$. In other words: $\{\mathcal{A}_k\}_k$ *pointwise* converges to $\mathcal{A}$. We will present a stronger result, namely, that a degree of continuity or stability is achieved.

It is natural to wonder if $\mathcal{A}$ can be written as a limit of stable or continuous algorithms. The answer is "no". Consider the algorithm $\mathcal{Z}$ which takes a rational number as input and returns "TRUE" if the input is 0 and "FALSE" if the input is not 0. Any stable or continuous function from the rationals - with the usual topology or notion of nearness - to the set {"TRUE","FALSE"} - with the discrete topology - must be a constant function which always returns "TRUE" or always returns "FALSE".[65] And of course any limit of constant functions is a constant function. Hence, $\mathcal{Z}$ cannot be approximated by stable or continuous algorithms or functions. Let us show how $\mathcal{Z}$ is approximated by our techniques. For each positive integer k let $\mathcal{Z}_k$ be the algorithm which takes a rational number as input and returns "TRUE" if the input has absolute value less than or equal to $1/10^k$ and "FALSE" if the input has absolute value greater than $1/10^k$. If $\{q_\ell\}_\ell$ is any sequence of rational numbers with limit q - including the possibility that $q = 0$ - then $lim_k(lim_\ell \mathcal{Z}_k(q_\ell)) = \mathcal{Z}(q)$.

5.1.7 *THEOREM $\mathcal{A}$-Stabilization:* *Suppose that $p_i \to \hat{0}$ slowly.*[66] *We write $\mathcal{A}_i$ in place of $\mathcal{A}_{p_i}$. Suppose that $a_1, \ldots, a_m \in \mathcal{S}(R;X)$ is input data for the algorithm $\mathcal{A}$*

[63] If, instead of coming from an output variable, b_j comes from a constant in $\mathcal{S}(R;X)$ then $\{b_{jk}\}_{k \geq N}$ is the constant sequence where $b_{jk} = b_j$ for each $k \geq N$.

[64] For good approximation behaviour we will have to assume that $\{p_k\}$ does not reach $\hat{0}$ too quickly. For example the case where for each k: $p_k = \hat{0}$, is not very interesting. In this case each $\mathcal{A}_k$ is simply $\mathcal{A}$.

[65] This comes down to the fact that a continuous function carries a connected set to a connected set.

[66] The definition of $p_i \to \hat{0}$ *slowly* is that $p_i \to \hat{0}$ and if (o_i) is any other sequence of elements of P with $o_i \to \hat{0}$ then for any M there is N where $i \geq N \Rightarrow o_i \leq p_M$. $\hat{0}$ is *not isolated* in P, if there exists a sequence of non-$\hat{0}$ elements of P which converges to $\hat{0}$. It is easy to check that if $\hat{0}$ is *not isolated* in P then $p_i \to \hat{0}$ *slowly* if and only if (p_i) consists of non-$\hat{0}$ elements of P and if $\hat{0}$ is *isolated* in P then any sequence which approcahes $\hat{0}$ does so *slowly*. In this latter case a sequence approaches $\hat{0}$ if and only if the terms eventually become $\hat{0}$.

on which it terminates normally. Assume that for each a_k there is a sequence $\{a_{kj}\}_k$ of elements of $\mathcal{S}(R;X)$ which converges to a_k.

A. There is N which depends on $a_1,\ldots,a_m$ where for $i \geq N$ there is an integer M_i and for $j \geq M_i$ $\mathcal{A}_i(a_{1j},\ldots,a_{mj})$ terminates normally. Moreover, the sequence $\{\mathcal{A}_i(a_{1j},\ldots,a_{mj})\}_{j\geq M_i}$ converges.

Let $b_{1i},\ldots,b_{ni}$ denote the limit: $lim_{j\geq M_i}\mathcal{A}_i(a_{1j},\ldots,a_{mj})$.

B. The sequence $\{b_{1i},\ldots,b_{ni}\}_{i\geq N}$ converges to $\mathcal{A}(a_1,\ldots,a_m)$.

Put differently this shows that:

5.1.8 $$lim_i(lim_j\mathcal{A}_i(a_{1j},\ldots,a_{mj})) = \mathcal{A}(a_1,\ldots,a_m)$$

This is a strong stability result because it shows that as the input data to the $\mathcal{A}_i$'s gets close to a specific input of $\mathcal{A}$ then the outputs of the $\mathcal{A}_i$'s gets close to the output of $\mathcal{A}$. This may be as strong a notion of stability or continuity as can be achieved!

The next key idea is an important variation of the construction of $\mathcal{BC}(\mathcal{A})$. This will permit the use of inexact or approximate computation while still achieving the results of: Theorem (5.1.2), Corollary (5.1.4), Corollary (5.1.6) and Theorem (5.1.7).

$\mathcal{BC}(\mathcal{A})$ extended $\mathcal{A}$ from $\mathcal{S}(R;X)$ to $\mathcal{S}(\mathcal{BC}[R,P];X)$ by using BC-arithmetic and its generalization (4.2.2) which is built upon arithmetic and convergent bounds in R and P. Since we are using approximation anyway, we do not have to use exact arithmetic in R. This is where approximate computation (5.4.1) will come in. Approximate computation allows us to use approximate arithmetic in R for the R component of a bracket coefficient. For example, if R is the real numbers or lies in the real numbers, approximate arithmetic could be the arithmetic operations resulting from using fixed precision floating point operations instead of exact arithmetic. Alternatively, if R is a topological ring consisting of the inverse limit of a ring modulo powers of an ideal, the approximate arithmetic could be the result of working in the ring modulo a specific power of the ideal instead of the inverse limit.

$\mathcal{BC}_k(\mathcal{A})$ will be used to denote this variant of $\mathcal{BC}(\mathcal{A})$ which utilizes approximate computation.[67] In the same way that we coerced the output of $\mathcal{BC}(\mathcal{A})$ to lie in $\mathcal{S}(R;X)$ and called the resulting algorithm $\mathcal{BC}(\mathcal{A})_R$, we coerce the output of $\mathcal{BC}_k(\mathcal{A})$ to lie in $\mathcal{S}(R;X)$ and call the resulting algorithm $\mathcal{BC}_k(\mathcal{A})_R$. The results: Theorem (5.1.2), Corollary (5.1.4) for $\mathcal{BC}(\mathcal{A})$ and $\mathcal{BC}(\mathcal{A})_R$ have extensions for $\mathcal{BC}_k(\mathcal{A})$ and $\mathcal{BC}_k(\mathcal{A})_R$.

Just above (5.1.5), $\mathcal{A}_{p_k}(a_1,\ldots,a_m)$ was defined as:

$$\mathcal{BC}(\mathcal{A})_R(\mathcal{S}((\rho_k,\alpha_k);X)(a_1),\ldots,\mathcal{S}((\rho_k,\alpha_k);X)(a_m))$$

[67] The k is for the k^{th} (degree of) approximate computation.

This has an extension using the $\mathcal{BC}_k(\mathcal{A})$'s as does Corollary (5.1.6) and Theorem (5.1.7). Thus one may use approximate computation to stably execute the algorithm $\mathcal{A}$. The extension to approximate computation is explained in Section (5.4).

Now some details.

5.2 Extension of $\mathcal{A}$ to $\mathcal{BC}(\mathcal{A})$

Let us now carefully specify the extension of a $\mathcal{CCB}$ algorithm $\mathcal{A}$ defined on $\mathcal{S}(R; X)$ to an algorithm $\mathcal{BC}(\mathcal{A})$ defined on $\mathcal{S}(\mathcal{BC}[R, P]; X)$. To be precise we require that (5.1.1) is satisfied.

Since $\mathcal{BC}(\mathcal{A})$ is unchanged *textually* from $\mathcal{A}$, what we must do is specify how to interpret $\mathcal{A}$ on data from $\mathcal{S}(\mathcal{BC}[R, P]; X)$. We must specify how to deal with the **Initialize** line and the four types of instructions:

- "stop with *Assignment of output_data*"
- "goto"
- "conditional goto"
- "local assignment from computation"

The **Initialize** line of $\mathcal{A}$ now assigns *input_data* from $\mathcal{S}(\mathcal{BC}[R, P]; X)$ to the internal, local variables, if there are any. If the **Initialize** line is simply:

- Initialize: ()

Then – as for the original algorithm – no internal, local variable assignment is made. In general the **Initialize** line does contain internal, local variables and while the internal, local variables of $\mathcal{A}$ originally are assigned values from $\mathcal{S}(R; X)$, the internal, local variables of $\mathcal{BC}(\mathcal{A})$ are assigned values from $\mathcal{S}(\mathcal{BC}[R, P]; X)$.

The **goto** instruction is an easy case. As for the original $\mathcal{A}$ algorithm:

at step_i: "goto step_j"	Local variable assignments are unchanged. Continue at step_j.

What remains to explain are the **conditional goto** instruction and the **local assignment from computation** instruction and the **stop** instruction. Superficially, as for the original $\mathcal{A}$ algorithm we are dealing with one of:

at step_i: "stop $(Y_1, \ldots, Y_n)$"	Execution halts. The algorithm outputs the sequence consisting of the values of the Y_i's.
at step_i: "goto step_j if $Predicate(Y_1, \ldots, Y_n)$"	Local variable assignments are unchanged. "$Predicate(Y_1, \ldots, Y_n)$" is evaluated. If: TRUE continue at step_j. FALSE continue at step_$(i+1)$.
at step_i: "$X = Function(Y_1, \ldots, Y_n)$"	$Function(Y_1, \ldots, Y_n)$ is evaluated. The result is assigned to X. Continue at step_$(i+1)$.

The obstacle to specifying how "$Predicate()$" and "$Function()$" are evaluated on arguments from $\mathcal{S}(\mathcal{BC}[R,P];X)$ is that "$Predicate()$" and "$Function()$" originally have arguments from $\mathcal{S}(R;X)$.

Here is the treatment of the **stop** instruction. As mentioned in the definition of algorithms in section (2.1), what appears in the parentheses of the "stop" step or as arguments to the predicate or function, if any such Y_i's appear, are internal, local variables or elements of $\mathcal{S}(R;X)$. Any of the Y_i's which are internal, local variables are expected to hold values from $\mathcal{S}(\mathcal{BC}[R,P];X)$.[68] In the stop, predicate and function case, the first step is to coerce any arguments which are not internal, local variables – i.e. Y_i's which lie in $\mathcal{S}(R;X)$ – to $\mathcal{S}(\mathcal{BC}[R,P];X)$. Here is how. For $i = 1, \ldots, n$ set:

$$w_i = u \text{ if } Y_i \text{ is an internal, local variable with value } u \in \mathcal{S}(\mathcal{BC}[R,P];X)$$

$$w_i = [Y_i, \hat{0}] \text{ if } Y_i \text{ is not an internal, local variable but simply an element of } \mathcal{S}(R;X)$$

Now in the case of a "stop" step the execution halts and the algorithm returns $(w_1, \ldots, w_n)$.

5.2.1 ***NOTE*** *that the "stop" instruction and output is handled differently for* $\mathcal{BC}(\mathcal{A})_R$. *Here is the description how*

at step_i: "stop $(Y_1, \ldots, Y_n)$"	*Execution halts. The algorithm outputs the sequence consisting of the values of the Y_i's.*

is treated for $\mathcal{BC}(\mathcal{A})_R$. *For* $i = 1, \ldots, n$ *set:*

$$v_i = Y_i \text{ if } Y_i \text{ is not an internal, local variable but simply an element of } \mathcal{S}(R;X)$$

[68] As usual, if any of the internal, local variables are not assigned or any of the arguments are inappropriate data, the algorithm crashes.

If Y_i is an internal, local variable, it is an output variable (5.1.3) and there may be a discrete set $S_{Y_i} \subset \mathcal{S}(R;X)$ associated with Y_i. If not S_{Y_i} is simply considered to be the empty set. For $i = 1, \ldots, n$ set:

$$v_i = (S_{Y_i} \bowtie u)_R \text{ if } Y_i \text{ is an internal, local variable with value } u \in \mathcal{S}(\mathcal{BC}[R,P];X)$$

At this "stop" instruction execution halts and the algorithm returns $(v_1, \ldots, v_n)$.

Now the handling of the "predicate" step and the "function" step for $\mathcal{BC}(\mathcal{A})$. We have already formed $(w_1, \ldots, w_n)$. Consider $(w_1, \ldots, w_n)$ to be an element in $\mathcal{S}(\mathcal{BC}[R,P];X)$.

Here is how to proceed with the predicate case. $S_{Predicate}$-rewrite the element $(w_1, \ldots, w_n)$ and evaluate predicate on:

$$(S_{Predicate} \bowtie (w_1, \ldots, w_n))_R \in \mathcal{S}(R;X)$$

In other words, evaluate:

$$Predicate((S_{Predicate} \bowtie (w_1, \ldots, w_n))_R)$$

If $(S_{Predicate} \bowtie (w_1, \ldots, w_n))_R$ is suitable data for $Predicate$ – i.e. lies in the domain of definition of $Predicate$ – then

$$Predicate((S_{Predicate} \bowtie (w_1, \ldots, w_n))_R)$$

evaluates to "TRUE" or "FALSE" and the **conditional goto** may be carried-out. If $(S_{Predicate} \bowtie (w_1, \ldots, w_n))_R$ does not lie in the domain of definition of $Predicate$ then

$$Predicate((S_{Predicate} \bowtie (w_1, \ldots, w_n))_R)$$

is not defined and the algorithm crashes. This finishes the specification of how to evaluate predicates and next we show how to proceed with the function case.

Let us rename $Function$ to φ. Since $\mathcal{A}$ is a $\mathcal{CCB}$ algorithm, (2.4.11), φ is a function with constructive convergent bound. (4.5.10) shows how to extend φ to Φ defined on a subset of $\mathcal{S}(\mathcal{BC}[R,P];X)$. If $(w_1, \ldots, w_n)$ lies in A' – the domain of definition of Φ – then simply evaluate $\Phi(w_1, \ldots, w_n)$ and assign the result to X. If $(w_1, \ldots, w_n)$ does not lie in the domain of definition of Φ then the algorithm crashes. This completes the rigorous definition of the extension of $\mathcal{A}$ to $\mathcal{BC}(\mathcal{A})$.

5.3 Stability

This section recounts how stability works. Beyond containing the technical details, the principles which emerge here are key to an understanding of stability and to the continued

development in (5.4). Suppose that $\mathcal{A}$ satisfies (5.1.1). Suppose that $a_1, \ldots, a_m \in \mathcal{S}(R; X)$ is suitable input data for $\mathcal{A}$. Let $\{\boldsymbol{a_k}\}_k$ be an approximation sequence in $\mathcal{S}(\mathcal{BC}[R, P]; X)$ for $\underline{a}$. We shall show that if $\mathcal{A}$ terminates normally when invoked with input data $\underline{a}$ then there exists M where for $k \geq M$ the algorithm $\mathcal{BC}(\mathcal{A})$ with input data $\{\boldsymbol{a_k}\}_k$ takes exactly the same *execution path*[69] as the algorithm $\mathcal{A}$ with input data $\underline{a}$.

Moreover, if $b_1, \ldots, b_n \in \mathcal{S}(R; X)$ is the output data for $\mathcal{A}$ invoked with input data $\underline{a}$ and $\underline{b}$ denotes $(b_1, \ldots, b_n) \in \mathcal{S}(R; X)$ then the sequence $\{\mathcal{BC}(\mathcal{A})(\boldsymbol{a_k})\}_{k \geq M}$ forms an approximation sequence in $\mathcal{S}(\mathcal{BC}[R, P]; X)$ for $\underline{b}$. This will prove (5.1.2) from which (5.1.4), (5.1.6) and (5.1.7) follow.

Executing an algorithm $\mathcal{A}$ with input data $\underline{a}$ results in a series of steps being performed, as described in (2.1). It begins with the *Initialize* step which is always followed by *step_1*. If *step_1* is a *goto* or *conditional goto*, it may happen that some other step than *step_2* is performed immediately after *step_1*. Let $\mathcal{EP}$ be the ordered list or sequence – starting with *step_1* – of the steps which are performed when executing the algorithm $\mathcal{A}$ with input data $\underline{a}$. $\mathcal{EP}$ is simply the sequence of step **numbers** in the order they are performed.[70] $\mathcal{EP}$ is a finite list when $\mathcal{A}$ terminates normally – i.e. after a finite number of steps. $\mathcal{EP}$ always has "1" as the first element and so is never an empty list. If the algorithm *runs forever* – in the terminology of Section (2.1) – then $\mathcal{EP}$ will be an infinite sequence of integers. In case $\mathcal{A}$ crashes we treat $\mathcal{EP}$ as follows. Each step successfully performed is added to the list. The step where the crash occurs is also added to the list and is followed by a final 0 on the list.[71] $\mathcal{EP}$ may have duplicate numbers and duplicate subsequences of numbers. Looping structures in $\mathcal{A}$ would typically cause duplicates.

5.3.1 ***DEFINITION Execution Path:*** *$\mathcal{EP}$ is the execution path of $\mathcal{A}$ invoked with input data $\underline{a}$.*

For specific k the algorithm $\mathcal{BC}(\mathcal{A})$ invoked with input data $\boldsymbol{a_k}$ may or may not terminate normally. In any event, let $\mathcal{EP}_k$ denote the execution path of $\mathcal{BC}(\mathcal{A})$ invoked with input data $\boldsymbol{a_k}$. Our claim about execution paths becoming the same can now be stated precisely:

5.3.2 *If $\mathcal{A}$ invoked with input data $\underline{a}$ terminates normally, then there is an integer M where $k \geq M \Rightarrow \mathcal{EP}_k = \mathcal{EP}$.*

[69] By *execution path* we mean an ordered list of the steps which are executed. This will be explained in greater detail.

[70] For a specific number on the list $\mathcal{EP}$ we can simply look at the text of the algorithm $\mathcal{A}$ to see what is the actual *instruction* at that step number.

[71] Let us make a convention that in case of a "*(conditional) goto*" to a step number which does not exist, the crash occurs at the location of the errant "*(conditional) goto*" as opposed to the missing location of the missing step. This insures that except for a final 0 in the case of a crash, only positive numbers appear on the list since actual step numbers are positive integers.

We prove our main result by proving a result which is valid when $\mathcal{A}$ terminates normally or runs forever, when invoked with input data $\underline{a}$.[72] We shall show that in this case the sequence of lists $\mathcal{EP}_k$ approach $\mathcal{EP}$ in the sense that bigger and bigger initial segments coincide with the corresponding initial segment of $\mathcal{EP}$. At the same time the internal, local variables of $\mathcal{BC}(\mathcal{A})$ will be approximation sequences for the internal, local variables of $\mathcal{A}$. This will do the trick because when $\mathcal{A}$ terminates normally on $\underline{a}$ then $\mathcal{EP}$ is finite and so when the initial segment reaches all of $\mathcal{EP}$ we will be done. The proof is by induction and here is the precise inductive hypothesis:

5.3.3 *At stage N in the induction we have M_N where for $k \geq M_N$ conditions* ***A*** *and* ***B*** *are satisfied:*

A. $\mathcal{EP}$ coincides[73] with $\mathcal{EP}_k$ up to length N.

Since $\mathcal{A}$ and $\mathcal{BC}(\mathcal{A})$ are textually the same, by **A.** the same steps or instructions in $\mathcal{A}$ and $\mathcal{BC}(\mathcal{A})$ will have been performed and precisely the same internal, local variables will have been assigned data in the performance of the first N steps of each algorithm.[74] For each internal, local variable LV which is assigned data in the performance of the first N steps of each algorithm, let $LV(N)$ be the value of the internal, local variable LV in the execution of $\mathcal{A}$ when N steps have been performed. Or the final value of LV if $\mathcal{A}$ terminated normally in fewer than N steps. Let $\mathcal{BC}(LV(N))_k$ be the value of the internal, local variable LV in the execution of $\mathcal{BC}(\mathcal{A})$ when N steps have been performed. Or the final value of LV if $\mathcal{BC}(\mathcal{A})$ terminated normally in fewer than N steps.

B. $\{\mathcal{BC}(LV(N-1))_k\}_{k \geq M_N}$ is an approximation sequence for $LV(N-1)$.

5.3.4 *Initial Induction Invocation and Verification of the Induction Hypothesis:*

The induction begins with $N = 1$ for which $M_1 = 0$ or wherever the approximation sequence $\{\boldsymbol{a_k}\}_k$ for $\underline{a}$ begins. $\mathcal{EP}$ coincides with $\mathcal{EP}_k$ up to length 1 because all execution paths begin with "1". Hence, **A.** is satisfied. As for **B.**, internal, local variables will only have been assigned data before *step_1* by assignment in the *Initialize* step in $\mathcal{A}$ and $\mathcal{BC}(\mathcal{A})$. Thus the approximation sequence requirement of **B.** is satisfied because $\{\boldsymbol{a_k}\}_k$ is assumed to be an approximation sequence for $\underline{a}$. This verifies the initial induction invocation.

Next comes the verification of the induction hypothesis. Assume that it holds true for N and we must verify it for $N + 1$. This will be handled in cases. The easy or trivial

[72] We do not know crash behaviour possibilities particularly well.

[73] Two sequences *coincide up to length N* means that if either sequence has length less than N then both do and they are equal. Otherwise the first N entries in the sequences are the same.

[74] I.e. the steps in the algorithm which correspond to the first N entries of $\mathcal{EP}$ or equivalently $\mathcal{EP}_k$.

case is where $\mathcal{A}$ or $\mathcal{BC}(\mathcal{A})$ terminated normally in **fewer** than N steps. In this case by **A.** if one terminated in fewer than N steps then both terminated at the same point and nothing new happens with the internal, local variables. I.e. $LV(N) = LV(N-1)$ and $\mathcal{BC}(LV(N))_k = \mathcal{BC}(LV(N-1))_k$ for $k \geq M_N$. Hence, we may let M_{N+1} equal M_N and the induction hypothesis has been verified.

The real work now begins, where both $\mathcal{A}$ and $\mathcal{BC}(\mathcal{A})$ did reach an N^{th} instruction. This breaks into cases according to whether the N^{th} instruction is:

- "stop with *Assignment of output_data*"
- "goto"
- "conditional goto"
- "local assignment from computation"

CONDITIONAL GOTO: We are assuming that i is the N^{th} entry in the list $\mathcal{EP}$ and that the i^{th} step in $\mathcal{A}$ is:

$$\text{step\_}i\text{: "goto step\_}j \text{ if } Predicate(X_1, \ldots, X_n)\text{"}$$

As described in Section (2.1), internal, local variable assignments are unchanged. Hence **B.** will be satisfied automatically and the problem is to satisfy **A.**. Since for $k \geq M_N$, $\mathcal{EP}$ coincides with $\mathcal{EP}_k$ up to length N, we must show that the $N+1^{st}$ entries of the two are the same. Or rather that there is $M_{N+1} \geq M_N$ where for $k \geq M_{N+1}$ the $N+1^{st}$ entries of the two are the same. Let LV be the conglomeration of the internal, local variables $\{X_1, \ldots, X_n\}$.[75] We think of LV as standing for $(X_1, \ldots, X_n)$. Since convergence is component-wise, convergence for LV can be expressed in terms of convergence for the X_i's and vice-versa. For $\mathcal{A}$ the N^{th} instruction to be performed is:

$$\text{step\_}i\text{: "goto step\_}j \text{ if } Predicate(LV(N-1))\text{"}$$

Hence the $N+1^{st}$ entry in $\mathcal{EP}$ is j if $Predicate(LV)$ is "TRUE" and is $i+1$ if $Predicate(LV)$ is "FALSE". Similarly for $\mathcal{BC}(\mathcal{A})$ invoked with the input data $\boldsymbol{a_k}$, the N^{th} instruction to be performed is:

$$\text{step\_}i\text{: "goto step\_}j \text{ if } Predicate(\mathcal{BC}(LV(N-1))_k)\text{"}$$

Hence the $N+1^{st}$ entry in $\mathcal{EP}_k$ is j if $Predicate(\mathcal{BC}(LV(N-1))_k)$ is "TRUE" and is $i+1$ if $Predicate(\mathcal{BC}(LV(N-1))_k)$ is "FALSE".

Thus we have to produce M_{N+1} where:

5.3.5 $$k \geq M_{N+1} \Rightarrow Predicate(LV(N-1)) = Predicate(\mathcal{BC}(LV(N-1))_k)$$

[75] We could work with the X_i's individually but this simplifies notation.

For this we must utilize the fact that by the induction hypothesis $\{\mathcal{BC}(LV(N-1))_k\}_{k\geq M_N}$ is an approximation sequence for $LV(N-1)$.

As described in the previous section, in the execution of $\mathcal{BC}(\mathcal{A})$, $Predicate$ is evaluated at $\mathcal{BC}(LV(N-1))_k$, by doing $S_{Predicate}$ rewriting of $\mathcal{BC}(LV(N-1))_k$ to form $V(N-1)_k$. and then $Predicate$ is evaluated at $(V(N-1)_k)_R$ which is an element of $\mathcal{S}(R;X)$. By hypothesis, $\{\mathcal{BC}(LV(N-1))_k\}_{k\geq M_N}$ is an approximation sequence for $LV(N-1)$. By (4.5.9), $\{\mathcal{BC}(LV(N-1))_k\}_{k\geq M_N}$ converges to $\overrightarrow{[}\ LV(N-1), \hat{0}\ \overleftarrow{]}$ and the convergence is finite convergence if $LV(N-1)$ is in $S_{Predicate}$. Hence, $\{(\mathcal{BC}(LV(N-1))_k)_R\}_{k\geq M_N}$ converges to $LV(N-1)$ and the convergence is finite convergence if $LV(N-1)$ is in $S_{Predicate}$.

By hypothesis the domain of $Predicate$ – call it A – equals $S_{Predicate} \cup \Omega(Predicate)$. Thus $LV(N-1)$ must lie in $S_{Predicate}$ or $\Omega(Predicate)$ or both. If $LV(N-1) \in \Omega(Predicate)$ then $Predicate$ is continuous at $LV(N-1)$. Also, since $\Omega(Predicate)$ is an open set and $\{(\mathcal{BC}(LV(N-1))_k)_R\}_{k\geq M_N}$ converges to $LV(N-1)$, there is an integer $M' \geq M_N$ where $k \geq M' \Rightarrow (\mathcal{BC}(LV(N-1))_k)_R \in \Omega(Predicate)$. Now $\{(\mathcal{BC}(LV(N-1))_k)_R\}_{k\geq M'}$ is a sequence of elements of A converging to $LV(N-1)$. Since $Predicate$ is continuous at $LV(N-1)$ (4.6.5) there is $M'' \geq M'$ where $k \geq M'' \Rightarrow Predicate((\mathcal{BC}(LV(N-1))_k)_R) = Predicate(LV(N-1))$. Hence, letting $M_{N+1} = M''$, (5.3.5) is satisfied. If $LV(N-1) \in S_{Predicate}$ then by (4.5.9), the convergence of $\{\mathcal{BC}(LV(N-1))_k\}_{k\geq M_N}$ to $\overrightarrow{[}\ LV(N-1), \hat{0}\ \overleftarrow{]}$ is finite. Thus there is an $M' \geq M_N$ where $k \geq M' \Rightarrow$:

$$\mathcal{BC}(LV(N-1))_k)_R) = LV(N-1)$$

and so $k \geq M'$ implies that

$$Predicate((\mathcal{BC}(LV(N-1))_k)_R) = Predicate(LV(N-1))$$

Now let M_{N+1} be M'. Then (5.3.5) is satisfied. This completes the case for **CONDITIONAL GOTO:**.

STOP: We are assuming that i is the N^{th} entry in the list $\mathcal{EP}$ and that the i^{th} step in $\mathcal{A}$ is:

$$\text{step\_}i\text{: ``stop } (X_1, \ldots, X_n)\text{''}$$

This is an easy case. For $\mathcal{A}$ the N^{th} instruction to be performed is:

$$\text{step\_}i\text{: ``stop } (LV(N-1))\text{''}$$

As described in Section (2.1), the execution of $\mathcal{A}$ halts and $LV(N-1)$ is assigned as the output of $\mathcal{A}$. Hence $\mathcal{EP}$ ends with the N^{th} and last entry i and has no $N+1^{st}$ entry.

Similarly for $\mathcal{BC}(\mathcal{A})$ invoked with the input data $\boldsymbol{a_k}$, the N^{th} instruction to be performed is:

$$\text{step\_}i\text{: ``stop } (\mathcal{BC}(LV(N-1))_k)\text{''}$$

The execution of $\mathcal{BC}(\mathcal{A})$ halts and $\mathcal{EP}_k$ ends with the N^{th} and last entry i and has no $N+1^{st}$ entry. Thus the inductive step for **A.** is satisfied. As described in Section (2.1), internal, local variable assignments are unchanged at a *stop* step, and so **B.** will be satisfied automatically, simply letting $M_{N+1} = M_N$.

GOTO: We are assuming that i is the N^{th} entry in the list $\mathcal{EP}$ and that the i^{th} step in $\mathcal{A}$ is:

$$\text{step\_}i\text{: ``goto step\_}j\text{''}$$

This is another easy case. In both $\mathcal{A}$ and $\mathcal{BC}(\mathcal{A})$, the N^{th} instruction is the same "goto step_j" instruction. Hence the $N+1^{st}$ entries of both $\mathcal{EP}$ and $\mathcal{EP}_k$ are the same: j. Thus **A.** is satisfied for $N+1$. As described in Section (2.1), internal, local variable assignments are unchanged at a *goto* step, and so **B.** will be satisfied automatically, simply letting $M_{N+1} = M_N$.

LOCAL ASSIGNMENT FROM COMPUTATION: We are assuming that i is the N^{th} entry in the list $\mathcal{EP}$ and that the i^{th} step in $\mathcal{A}$ is:

$$\text{step\_}i\text{: ``}X = Function(Y_1, \ldots, Y_n)\text{''}$$

First we verify **B.** for $N+1$. Suppose that $Function$ is φ and $\varphi : A \to C$ where $A, C \subseteq \mathcal{S}(R; X)$. Since $\mathcal{A}$ is a $\mathcal{CCB}$ algorithm, φ is a structured function with constructive convergent bound, (4.3.8). Also by assumption (5.1.1), functions are defined on open sets. Thus A is an open set in $\mathcal{S}(R; X)$.

As described in Section (2.1), $\mathcal{BC}(\mathcal{A})$ performs the N^{th} instruction by setting:

$$X := \varphi(LV(N-1))$$

where $LV(N-1) \in A$ and $X \in C$. On the other hand, $\mathcal{BC}(\mathcal{A})$ utilizes $\Phi : A' \to C'$, the extension of φ to $\mathcal{S}(\mathcal{BC}[R, P]; X)$, (4.5.10). That is, $\mathcal{BC}(\mathcal{A})$ performs the N^{th} instruction by setting:

$$X := \Phi(\mathcal{BC}(LV(N-1))_k)$$

For both $\mathcal{A}$ and $\mathcal{BC}(\mathcal{A})$, X is the only internal, local variable whose value is changed[76] at the N^{th} step. Hence for all internal, local variables, with the possible exception of X, **B.** will be satisfied automatically by simply letting $M_{N+1} = M_N$. Thus we only have to show that **B.** is satisfied by X for suitable M_{N+1} which is greater than or equal to M_N.

To clarify what we are dealing with, we let $X(N)$ stand for the value of the internal, local variable X after this step N in the execution of $\mathcal{A}$ with input data $\underline{a}$. We let $\mathcal{BC}(X(N))_k$ stand for the value of the internal, local variable X after this step N in the

[76]This may be where X is first assigned a value. "Changed" is meant to include this possible meaning.

execution of $\mathcal{BC}(\mathcal{A})$ with input data $\{\boldsymbol{a_k}\}_k$. We must produce $M_{N+1} \geq M_N$ where

$$\{\mathcal{BC}(X(N))_k\}_{k \geq M_{N+1}} \text{ is an approximation sequence for } X(N)$$

Suppose that $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$ is a **continuous convergent bound** for φ. $\Phi(\mathcal{BC}(LV(N-1))_k)$ is defined as:

$$\overrightarrow{[}\ \varphi(\mathcal{BC}(LV(N-1))_{k_R}), \omega(\mathcal{BC}(LV(N-1))_k)\ \overleftarrow{]}$$

By assumption

$$\{\mathcal{BC}(LV(N-1))_k\}_{k \geq M_N} \text{ is an approximation sequence for } LV(N-1)$$

This implies that $\{(\mathcal{BC}(LV(N-1))_k)_R\}_{k \geq M_N}$ converges to $LV(N-1)$. Since $LV(N-1) \in A$ an open set, there is $M' \geq M_N$ where $k \geq M' \Rightarrow (\mathcal{BC}(LV(N-1))_k)_R \in A$. Hence, $k \geq M' \Rightarrow (\mathcal{BC}(LV(N-1))_k) \in A'$ by (4.5.12). Thus we may apply (4.5.13) to conclude that

$$\{\Phi(\mathcal{BC}(LV(N-1))_k)\}_{k \geq M'}$$

is an approximation sequence for $\varphi(LV(N-1))$. Since $\mathcal{BC}(X(N))_k = \Phi(\mathcal{BC}(LV(N-1))_k$, by letting $M_{N+1} = M'$ we have verified **B.**.

Finally, if $k \geq M'$ $\mathcal{BC}(\mathcal{A})$ continues at "step_$(i+1)$", since $(\mathcal{BC}(LV(N-1))_k) \in A'$. Hence "$i+1$" is the $N+1^{st}$ entry in both $\mathcal{EP}$ and $\mathcal{EP}_k$. I.e. **A.** is satisfied for $N+1$. This completes the case of **LOCAL ASSIGNMENT FROM COMPUTATION:**.

5.4 Approximate Arithmetic and More General Functions

This section discusses two matters. The first is how inexact – but arbitrarily precise – computation may be used and still obtain the analogous results to (5.1.2), (5.1.4), (5.1.6) and (5.1.7). The second matter is how the restriction:

- The domain of each function in each "local assignment from computation" step of $\mathcal{A}$ is an open set of $\mathcal{S}(R; X)$.

in (5.1.1) may be relaxed. The style of presentation is *detailed expository form.*

We begin by defining the notion of approximate computation. For examples and motivation, consider the *rounding-off* in previous examples. The idea behind approximate computation is to only perform computation as far as needed to obtain the *rounded* answer. Rounded arithmetic is limited in precision. The use of rounded arithmetic to approximate exact computation is accomplished by considering successive rounded computations with

increasing precision. The same must be done with general approximate computation. That is why approximate computation takes the form of a collection or sequence of maps which converge (pointwise) to a given map or computation.

As in (3.1.20) and (4.2.6) where **Approx2′** and **Approx2″** are increasingly restrictive alternatives to **Approx2**, we introduce **Convergence′** and **Convergence″** as increasingly restrictive alternatives to **Convergence** in:

5.4.1 ***DEFINITION Approximate Computation:*** *Let $A \subseteq R^m$, $C \subseteq R^n$ and $\varphi : A \to C$. A pair of sequences of maps $\{\varphi_k\}_k$ and $\{\lambda_k\}_k$ where each φ_k maps A to C and each λ_k maps A to P^n is an approximate computation for φ if* ***Bound*** *and* ***Convergence*** *are satisfied. The pair of sequences of maps is a continuous approximate computation for φ if* ***Bound*** *and* **Convergence′** *are satisfied. The pair of sequences of maps is a locally compact approximate computation for φ if* ***Bound*** *and* **Convergence″** *are satisfied.*

Bound: *For $a \in A$: $\nu(\varphi(a) - \varphi_k(a)) \leq \lambda_k(a)$.*

Convergence: *For each $a \in A$, $\lambda_k(a) \to \hat{0}$ as $k \to \infty$*

Convergence′: *The sequence $\{\lambda_k\}_k$ has the following restricted form of convergence: For* ***Cauchy sequences*** *$\{a_k\}_k$ of elements of A, the sequence $\{\lambda_k(a_k)\}_k$ has the convergence:*

$$\lambda_k(a_k) \to \hat{0}\ (k \to \infty)$$

Convergence″: *The sequence $\{\lambda_k\}_k$ has the following restricted form of convergence: For* ***bounded sequences*** *$\{a_k\}_k$ of elements of A, the sequence $\{\lambda_k(a_k)\}_k$ has the convergence:*

$$\lambda_k(a_k) \to \hat{0}\ (k \to \infty)$$

φ_k is called the k^{th} approximate computation for φ.

Notice that under *approximate computation*, $\{\varphi_k\}_k$ converges point-wise to φ. Under *continuous approximate computation*, the convergence of $\{\varphi_k\}_k$ to φ is continuous convergence, [3]. And under *locally compact approximate computation*, the convergence of $\{\varphi_k\}_k$ to φ is uniform on compact neighborhoods of points. I.e. the convergence is uniform, locally compactly. Just plain *approximate computation* is not strong enough to insure that the conclusion holds for the proposition (5.4.8), which corresponds to (4.5.13). Such failure would be fatal to the correctness of certain stability theorems if they relied upon just plain *approximate computation*. Hence, one of the more restrictive conditions – *continuous approximate computation* or *locally compact approximate computation* – will be necessary.

Convergent approximations yield approximate computations and are an important means in which approximate computations arise. The construction follows. With this construction continuous convergent approximations yield continuous approximate computations by (4.6.4). Also, with this construction locally compact convergent approximations yield locally compact approximate computations by (4.3.1).

5.4.2 *EXAMPLE Approximate Computation from Convergent Approximation:* *Suppose that $\varphi : A \to C$ and that C has the convergent approximation: $\{\rho_k\}_k$, $\{\alpha_k\}_k$, where each $\rho_k : C \to C$ and $\alpha_k : C \to P^n$. Set $\varphi_k \equiv \rho_k \circ \varphi$ and $\lambda_k \equiv \alpha_k \circ \varphi$. Then$\{\varphi_k\}_k$ and $\{\lambda_k\}_k$ is an approximate computation for φ.*

$\mathcal{A}_{p_k}$ is defined at (5.1.5). Let us massage the definition of $\mathcal{A}_{p_k}$ to also include approximate computation. $\mathcal{A}_{p_k}$ was obtained from $\mathcal{A}$ by applying $\mathcal{BC}(\mathcal{A})_R$ to the k^{th} element[77] of an approximation sequence obtained from the original input to $\mathcal{A}$. To introduce approximate computation we must say what it means for a function on $\mathcal{S}(R;X)$ to have an approximate computation. In fact, this is only defined for structured functions.

5.4.3 *DEFINITION Functions on $\mathcal{S}(R;X)$ with Approximate Computation:* *Let $S, T \subseteq \mathcal{S}(R;X)$. A structured function $\varphi : S \to T$ is said to have an approximate computation if for each $z \in {}^*S$ the function $\kappa\varphi\overline{z} : \kappa S_z \to \kappa T_{\varphi(z)}$ has an approximate computation $\{\kappa\varphi\overline{z}_k\}_k$.*

Now suppose that the algorithm $\mathcal{A}$ satisfies (5.1.1) and the functions appearing in "local assignment from computation" step of $\mathcal{A}$ have an approximate computation. More formally:

5.4.4 *HYPOTHESES II:*

- *$\mathcal{A}$ is a continuous $\mathcal{CCB}$ algorithm, (4.3.9).*
- *The domain of the function in each "local assignment from computation" step of $\mathcal{A}$ is an open set of $\mathcal{S}(R;X)$.*
- *For the predicate $Predicate$ in each "conditional goto" step of $\mathcal{A}$ there is a discrete set $S_{Predicate}$ where the domain of $Predicate$ equals $S_{Predicate} \cup \Omega(Predicate)$.*
- *For the function $Function$ in each "local assignment from computation" step of $\mathcal{A}$, $Function$ has a continuous approximate computation.*

[77]The approximation sequence is obtained from $\{p_k\}_k \subset P$ using (3.1.12) and (4.5.7) as described earlier.

The approximate computations for functions in $\mathcal{A}$ are ultimately used to massage the definition of $\mathcal{BC}(\mathcal{A})$. First we must massage the definition of the extension of maps with convergent bounds at (4.2.2).

5.4.5 ***DEFINITION BC Extension of Functions with Convergent Bounds and Approximate Computation:*** *Let $A \subseteq R^m$, $C \subseteq R^n$ and $\varphi : A \to C$. Suppose that $\omega : \overrightarrow{[}\ A, P^m\ \overleftarrow{]} \to P^n$ is a* ***convergent bound*** *for φ and suppose that $\{\varphi_k\}_k$ and $\{\lambda_k\}_k$ is an approximate computation for φ. All together, $(\varphi, \omega, \{\varphi_k\}_k, \{\lambda_k\}_k)$ gives a sequence of maps $\{\varphi_{\omega k}\}_k$ from $\overrightarrow{[}\ A, P^m\ \overleftarrow{]}$ to $\overrightarrow{[}\ C, P^n\ \overleftarrow{]}$. $\varphi_{\omega k}$ is defined:*

$$\varphi_{\omega k}(t) \equiv \overrightarrow{[}\ \varphi_k(t_R), \omega(t) \hat{+} \lambda_k(t_R)\ \overleftarrow{]}$$

for $t \in \overrightarrow{[}\ A, P^m\ \overleftarrow{]}$. Equivalently, if $t = \overrightarrow{[}\ a, p\ \overleftarrow{]}$ then $\varphi_{\omega k}(t) = \overrightarrow{[}\ \varphi_k(a), \omega(\overrightarrow{[}\ a, p\ \overleftarrow{]}) \hat{+} \lambda_k(a)\ \overleftarrow{]}$.

We use this to massage the construction of the extension of maps on $\mathcal{S}(R; X)$ with convergent bounds. At (4.5.10) one began with $A, C \subseteq \mathcal{S}(R; X)$ and $\varphi : A \to C$ a structured function with constructive convergent bound. The construction yielded $A', C' \subseteq \mathcal{S}(\mathcal{BC}[R, P]; X)$ and a structured map $\Phi : A' \to C'$.

5.4.6 ***CONSTRUCTION Extension of Maps with Convergent Bounds and Approximate Computation to $\mathcal{S}(\mathcal{BC}[R, P]; X)$:*** *Unlike (4.5.10) this time we begin with $A, C \subseteq \mathcal{S}(R; X)$ and $\varphi : A \to C$ a structured function with constructive convergent bound and approximate computation. Follow (4.5.10) except that the extension of the map $\kappa\varphi\overline{z}$ to a map $\kappa\varphi\overline{z}_{\omega_{\kappa\varphi\overline{z}}}$ from $\overrightarrow{[}\ \kappa A_z, P^m\ \overleftarrow{]}$ to $\overrightarrow{[}\ \kappa C_{\varphi(z)}, P^n\ \overleftarrow{]}$ is replaced by $\kappa\varphi\overline{z}_{\omega_{\kappa\varphi\overline{z}}k}$ where $\kappa\varphi\overline{z}_{\omega_{\kappa\varphi\overline{z}}k}$ is defined at (5.4.5). The A' and C' one constructs are the same as before. The map from A' to C' one constructs is now called Φ_k since it utilizes the k^{th} approximate computation.*

One may show that for $a' \in A' \subseteq \mathcal{S}(\mathcal{BC}[R, P]; X)$:

5.4.7 $$\{\Phi_k(a')_R\}_k \text{ converges to } \{\Phi(a')_R\}_k$$

Using approximate computation, one may also generalize (4.5.13) to the following proposition. While it is easy to prove the proposition along the lines of the proof of (4.5.13), it is at the heart of why approximate computation works.

5.4.8 ***PROPOSITION Approximation Sequences Mapping to Approximation Sequences Using Approximate Computation:*** *Let $A, C \subseteq \mathcal{S}(R; X)$. Let*

$\varphi : A \to C$ be a structured function with constructive continuous convergent bound and continuous approximate computation. Let $\Phi_k : A' \to C'$ be the structured map just described in (5.4.6). Suppose that $\{E_\ell\}_\ell$ is a sequence of elements of $\mathcal{BC}$ which is an approximation sequence for $a \in \mathcal{S}(R; X)$. Suppose that there is an integer M where $\ell \geq M$ $\Rightarrow E_\ell \in A'$.

Then the sequence $\{\Phi_k(E_k)\}_{k \geq M}$ is an approximation sequence for the element $\varphi(a)$ of C.

In applications one uses Φ_k to replace Φ in the construction of $\mathcal{BC}(\mathcal{A})$. More specifically, one replaces Φ by Φ_k in the description how to handle the step:

at step_i: "$X = Function(Y_1, \ldots, Y_n)$"	$Function(Y_1, \ldots, Y_n)$ is evaluated. The result is assigned to X. Continue at step_$(i+1)$.

in Section (5.2). Now the construction depends upon the specific k and we call the resulting algorithm $\mathcal{BC}_k(\mathcal{A})$.

(5.1.2) and (5.1.4) have generalizations to $\mathcal{BC}_k(\mathcal{A})$. We are more interested in the generalization of (5.1.6) and (5.1.7). $\mathcal{A}_{p_k}$ is defined at (5.1.5). We use $\mathcal{BC}_k(\mathcal{A})$ to generalize this to:

5.4.9 *DEFINITION $\widetilde{\mathcal{A}_{p_k}}$:* *Suppose that $\mathcal{A}$ satisfies (5.4.4) and $\{p_k\}_k$ is a sequence of elements of P which converges to $\hat{0}$. If $a_1, \ldots, a_m \in \mathcal{S}(R; X)$ is input data for $\mathcal{A}$ then:*

$$\{\mathcal{S}((\rho_k, \alpha_k); X)(a_1)\}_k, \ldots, \{\mathcal{S}((\rho_k, \alpha_k); X)(a_m)\}_k$$

are strict approximation sequences for $a_1, \ldots, a_m$.[78] *Let $\widetilde{\mathcal{A}_{p_k}}(a_1, \ldots, a_m)$ denote:*

$$\mathcal{BC}_k(\mathcal{A})_R(\mathcal{S}((\rho_k, \alpha_k); X)(a_1), \ldots, \mathcal{S}((\rho_k, \alpha_k); X)(a_m))$$

Now the generalization of (5.1.6):

5.4.10 *PROPOSITION $\widetilde{\mathcal{A}_{p_k}}$-Stability:* *Suppose that $\mathcal{A}$ satisfies (5.4.4) and $a_1, \ldots, a_m \in \mathcal{S}(R; X)$ is input data for $\mathcal{A}$. Suppose that the output of $\mathcal{A}(a_1, \ldots, a_m)$ is $b_1, \ldots, b_n \in \mathcal{S}(R; X)$. Then there is N where for $k \geq N$ the output of $\widetilde{\mathcal{A}_{p_k}}(a_1, \ldots, a_m)$ is $b_{1k}, \ldots, b_{nk} \in \mathcal{S}(R; X)$. For each b_j, the sequence $\{b_{jk}\}_{k \geq N}$ converges to b_j. If b_j comes from an output variable OV then the sequence $\{b_{jk}\}_{k \geq N}$, S_{OV}-converges to b_j.*

[78] See (4.5.7) for the definition of $\mathcal{S}((\rho_k, \alpha_k); X)(a_i)$. See just above (5.1.5) and the end of section (5.1) for how this construction has appeared before.

This shows that the output of algorithms based upon exact computation may be approximated by a variation on the algorithm which utilizes approximate computation.

The low-level computation involved in $\widetilde{\mathcal{A}_{p_k}}$ is approximate computation. The importance of the previous result and the next result – which is the approximate computation analog to (5.1.7) – is that algorithms based on exact computation can be stabilized with approximate computation.

5.4.11 ***THEOREM $\mathcal{A}$-Stabilization with Approximate Computation:*** *Suppose that $\mathcal{A}$ satisfies (5.4.4). Suppose that $p_i \to \hat{0}$ slowly as defined in the footnote to (5.1.7). Write $\widetilde{\mathcal{A}_i}$ in place of $\widetilde{\mathcal{A}_{p_i}}$. Suppose that $a_1, \ldots, a_m \in \mathcal{S}(R;X)$ is input data for $\mathcal{A}$. Assume that for each a_j there is a sequence a'_{j_k} of elements of $\mathcal{S}(R;X)$ which converges to a_j.*

A. There is N which depends on $a_1, \ldots, a_m$ where for $i \geq N$ there is an integer M_i and for $j \geq M_i$ $\widetilde{\mathcal{A}_i}(a_{1j}, \ldots, a_{mj})$ terminates normally. Moreover, the sequence $\{\widetilde{\mathcal{A}_i}(a_{1j}, \ldots, a_{mj})\}_{j \geq M_i}$ converges.

Let $b_{1i}, \ldots, b_{ni}$ denote the limit: $lim_{j \geq M_i} \widetilde{\mathcal{A}_i}(a_{1j}, \ldots, a_{mj})$.

B. The sequence $\{b_{1i}, \ldots, b_{ni}\}_{i \geq N}$ converges to $\mathcal{A}(a_1, \ldots, a_m)$.

As before this gives a stabilization result, but with approximate computation this time:

5.4.12
$$lim_i(lim_j \widetilde{\mathcal{A}_i}(a_{1j}, \ldots, a_{mj})) = \mathcal{A}(a_1, \ldots, a_m)$$

This finishes the handling of approximate computation. We end by outlining how to handle functions with certain kinds of discontinuities. An example of such a function is the $Integer\_Part$ function, i.e. the function which returns the integer part of a real number. Many algorithms use $Integer\_Part$ and similar functions. $Integer\_Part$ is continuous at all real numbers except the integers, a discrete set.

In Section (5.2), S-rewriting only appeared in the treatment of steps of the form:

at step_i: "goto step_j if $Predicate(X_1, \ldots, X_n)$"	Local variable assignments are unchanged. "$Predicate(X_1, \ldots, X_n)$" is evaluated. If: TRUE continue at step_j. FALSE continue at step_$(i+1)$.

In particular, no S-rewriting appeared in the treatment of steps of the form:

at step_i: "$X = Function(Y_1, \ldots, Y_n)$"	$Function(Y_1, \ldots, Y_n)$ is evaluated. The result is assigned to X. Continue at step_$(i+1)$.

By introducing or permitting S-rewriting at $Function$ evaluation steps, we may handle more general functions. For this we must introduce somewhat different and more general hypothesis from HYPOTHESES (5.1.1):

5.4.13 *HYPOTHESES III:*

- *$\mathcal{A}$ is a structured algorithm, (2.3.1).*
- *For the function Function in each "local assignment from computation" step of $\mathcal{A}$, there is a discrete set $S_{Function}$ and an open set $O_{Function}$ where the domain of $Function$ equals $S_{Function} \cup O_{Function}$ and $Function$ has a constructive continuous convergent bound on the set $O_{Function}$.*
- *For the predicate Predicate in each "conditional goto" step of $\mathcal{A}$ there is a discrete set $S_{Predicate}$ where the domain of $Predicate$ equals $S_{Predicate} \cup \Omega(Predicate)$.*

Notice, this allows more general functions than $\mathcal{CCB}$ functions. (5.4.13) allows functions – like $Integer\_Part$ – which fail to have a constructive convergent bound on a discrete set. When (5.4.13) holds we must change the treatment of steps of the form:

at step_i: "$X = Function(Y_1, \ldots, Y_n)$"	$Function(Y_1, \ldots, Y_n)$ is evaluated. The result is assigned to X. Continue at step_$(i+1)$.

in Section (5.2). Follow the description in Section (5.2) up to the point of forming $(w_1, \ldots, w_n)$ which is considered to be an element of $\mathcal{S}(\mathcal{BC}[R, P]; X)$. As in Section (5.2) rename $Function$ to φ. Next S_φ-rewrite $(w_1, \ldots, w_n)$, i.e. form $(S_\varphi \bowtie (w_1, \ldots, w_n))$. If $S_\varphi \bowtie$ causes a change in $(w_1, \ldots, w_n)$ then $(S_\varphi \bowtie (w_1, \ldots, w_n)) = \overrightarrow{[}\ s, \hat{0}\ \overleftarrow{]}$ where $s \in S_\varphi$. In this case execute the function step by assigning X the value: $\overrightarrow{[}\ \varphi(s), \hat{0}\ \overleftarrow{]}$. This is possible since S_φ lies in the domain of definition of φ. If S_φ-rewriting causes no change in $(w_1, \ldots, w_n)$ then consider if $(w_1, \ldots, w_n)$ lies in A' the domain of definition of Φ described in (4.5.10). In this case assign X the value $\Phi(w_1, \ldots, w_n)$. If $(w_1, \ldots, w_n)$ does not lie in A' then the algorithm crashes.

With this variant on the definition of $\mathcal{BC}(\mathcal{A})$, and similarly for $\mathcal{BC}(\mathcal{A})_R$, the main results in section (5.1) still hold. The two extensions mentioned in this section may be combined. In this case algorithms are expected to satisfy:

5.4.14 *HYPOTHESES IV:*

- *$\mathcal{A}$ is a structured algorithm, (2.3.1).*
- *For the function Function in each "local assignment from computation" step of $\mathcal{A}$, there is a discrete set $S_{Function}$ and an open set $O_{Function}$ where the domain of Function equals $S_{Function} \cup O_{Function}$ and Function has a constructive continuous convergent bound and continuous approximate computation on the set $O_{Function}$.*
- *For the predicate Predicate in each "conditional goto" step of $\mathcal{A}$ there is a discrete set $S_{Predicate}$ where the domain of Predicate equals $S_{Predicate} \cup \Omega(Predicate)$.*

5.5 Concluding Remarks

Stabilization techniques for algebraic algorithms and $\mathcal{CCB}$ algorithms have been described. Such algorithms are *structured* algorithms. This section is about the problem of applying our stabilization techniques to a given algorithm which might not originally be presented in a suitable form for stabilization. One must come up with an alternative description of the algorithm where it a structured algorithm. Furthermore, one must alter the algorithm so that functions and predicates have empty or discrete Δ sets. This process is demonstrated by our treatment, 2.6.5, of the Buchberger algorithm. The Buchberger algorithm involved finite sets and set operations. Although such operations are *non-structured* per se, frequently one can *structure* them. The initial concluding remarks are about aspects of *structuring* set operations on finite sets. Then we close with several additional remarks about stabilizing algorithms which might not originally be presented in suitable form for stabilization.

Let us start with an example. Let R' be a set. One often uses finite subsets of R' as a data structure. However, throughout this paper we have been working on $\mathcal{S}(R; X)$ for stabilization. Hence one should introduce finite sequences of elements from a formal copy R of R' instead of finite sets of elements from R'. Of course one should be careful with the fact that sequences are different from sets in treating multiplicities and orderings of elements. For example, in the natural numbers, $\{1, 2\} = \{2, 1\} = \{1, 2, 2\}$ but $(1, 2)$, $(2, 1)$, and $(1, 2, 2)$ are all distinct. Let L be the set of finite sequences of elements from R. Note that $L = \mathcal{S}(R; \{\ \})$. Corresponding to union on sets, one should consider the union-like operation **UNION**: $L \times L \to L$. Namely for $(x_1, \ldots, x_r)$, $(y_1, \ldots, y_s) \in L$, $\mathbf{UNION}((x_1, \ldots, x_r), (y_1, \ldots, y_s)) = (z_1, \ldots, z_t)$ where $\{x_1, \ldots, x_r\} \cup \{y_1, \ldots, y_s\} = \{z_1, \ldots, z_t\}$ and $z_1 = x_1, \ldots, z_r = x_r$, and if any y_i's do not occur among the x_i's then z_{r+1} is the first of the y_i's which do not occur, z_{r+2} is the second of the y_i's which do not occur, etc. **UNION** is not structured. For $a, b, c, d \in R$, with a and b distinct and c and d distinct, $\mathbf{UNION}((a, b), (c, d))$ equals (a, b), (a, b, c), (a, b, d) or (a, b, c, d) depending which of a, b, c, d are distinct. Hence the structure of $\mathbf{UNION}((a, b), (c, d))$ depends on

the specific coefficients a, b, c, d and not just the structure of $((a,b),(c,d))$. Similarly **INTERSECTION**, **SETMINUS**, **SUBSET**, and **SETEQUAL**[79] are not structured. However, they all commonly occur in algorithms. In this case they can be *structured*. *Structuring* the **MEMBERSHIP**[80] function is fundamental to *structuring* these other set functions. **MEMBERSHIP** can be described as a structured algorithm as follows:

MEMBERSHIP:

Initialize:	(x, Y)
step_1:	$Z = Y$
step_2:	$a = \pi_1(Z)$
step_3:	if $ARE\_EQUAL(a, x)$ goto step_7
step_4:	$Z = \pi_{>1}(Z)$
step_5:	if $IS\_EMPTY(Z)$ goto step_8
step_6:	goto step_2
step_7:	stop (1)
step_8:	stop (0)

where π_1 is the projection onto the first component and $\pi_{>1}$ is (the list of the result of) the projection onto the components beyond 1. The functions in this algorithm are structured functions by (2.4.5,e).

Using the **MEMBERSHIP** algorithm as a *subalgorithm* or *subroutine*[81], **UNION** can be described as follows.

UNION:

Initialize:	(X, Y)
step_1:	$Z = X$
step_2:	$b = \pi_1(Y)$
step_3:	$M =$ **MEMBERSHIP**(b, Z)
step_4:	if $ARE\_EQUAL(M, 1)$ goto step_6
step_5:	$Z = G(Z, b)$
step_6:	$Y = \pi_{>1}(Y)$
step_7:	if $IS\_EMPTY(Y)$ goto step_9
step_8:	goto step_2
step_9:	stop (Z)

[79] Like **UNION**, these functions are all defined on L. They correspond to intersection, setminus, subset, and setequal on sets, respectively. The precise definitions are left to the reader.

[80] Given $d \in R$ and $(d_1, \ldots, d_n) \in L$, **MEMBERSHIP**$(d, (d_1, \ldots, d_n))$ is "1" if $d \in \{d_1, \ldots, d_n\}$ and "0" otherwise.

[81] Of course one could simply insert the body itself of the **MEMBERSHIP** algorithm into the algorithm for **UNION**.

Here G is the function defined at (2.4.6,G). G is a structured function by (2.4.5,g).

INTERSECTION, **SETMINUS**, **SUBSET**, and **SETEQUAL** can also be described as structured algorithms in a similar way. The details are left to the reader.

Once one has obtained a structured algorithm, the next problem for the purpose of stabilization is to transform functions and predicates with *non-discrete* Δ sets into structured functions and predicates with *empty* or *discrete* Δ sets. This is not always possible. In the remark after 4.6.20, we discussed how some predicates can be transformed suitably.

Here is another example, the equality test in R. We call this ARE_EQUAL as in the **MEMBERSHIP** algorithm. This predicate has domain equal to $R \times R$. It is easy to see that $\Delta(ARE\_EQUAL) = \{(r, r) \mid r \in R\}$. So if (a copy of) R is not discrete, the Δ set is not discrete. In this case, as in the discussion after 4.6.20, one can decompose $ARE\_EQUAL(a, x)$ into the *structured* function $u = a - x$ and the predicate $IS\_ZERO(u)$ which has the *discrete* Δ set $\{0\}$.

Finally, consider ARE_EQUAL in the **UNION** algorithm. The the domain is just $\{0, 1\} \times \{1\}$, a discrete set. So the Δ set of ARE_EQUAL in the **UNION** algorithm is discrete.

5.6 Further Examples

The body of the paper contains many illustrative examples. This appendix has several additional examples with a different purpose. The examples here illustrate limitations or technical details relevant to the theory in the body of the paper.

We begin with a counterexample. In the introduction we mentioned that spurious convergence may cause anomalous behavior regarding a possible converse to (5.1.2). Here is the example. We are considering a converse to Theorem (5.1.2) in the following sense. Assume that for each $a_i \in \mathcal{S}(R; X)$ there is an approximation sequence $\{a'_{ik}\}_k$ of elements of $\mathcal{S}(\mathcal{BC}[R, P]; X)$. If $\mathcal{A}(a_1, \ldots, a_m)$ does not *terminate normally*, then is there an N where for $k \geq N$ $\mathcal{BC}(\mathcal{A})(a'_{1k}, \ldots, a'_{mk})$ does not *terminate normally*? In general the answer is "No". A *bad* convergent bound prevents the converse. In fact we do not have a counterexample for the usual *good* convergent bounds which define the basic BC-Arithmetic, (4.1.1).

5.6.1 *EXAMPLE Counterexample to the Converse:*

Consider the following algorithm $\mathcal{A}$:

Initialize:	(X)
step_1:	$Y = X$
step_2:	*if* $Y = 0$ *goto step_5*
step_3:	$Y = Y \cdot X$
step_4:	*goto step_2*
step_5:	*stop* (X)

Suppose that $R = \boldsymbol{R}$ *and we use* ***ABSOLUTE VALUE****, (3.1.6). Let* $X = 1/3$ *and consider the approximation sequence* $\{[1/3, 1/(3+k)]\}_k$ *for* $1/3$*. Obviously* $\mathcal{A}(1/3)$ *does not terminate normally.*[82] *For multiplication let us define a new*[83] *convergent bound* $\tilde{\omega}_\times$ *as follows:*

$$\tilde{\omega}_\times([r_1, \epsilon_1], [r_2, \epsilon_2]) = \max(\epsilon_1, \epsilon_2, \omega_\times([r_1, \epsilon_1], [r_2, \epsilon_2])$$

where $\omega_\times$ *is the usual convergent bound for multiplication. I.e.* $\omega_\times([r_1, \epsilon_1], [r_2, \epsilon_2]) = |r_1|\epsilon_2 + \epsilon_1\epsilon_2 + \epsilon_1|r_2|$.

Note that $\tilde{\omega}_\times$ *satisfies the* ***Bound*** *and* ***Convergence*** *conditions of convergent bounds, (4.2.1). Moreover, it is easy to check that with the new convergent bound* $\tilde{\omega}_\times$ *we have* $[1/3, 1/(3+k)]^n = [1/3^n, 1/(3+k)]$*. I.e. the resulting error terms do* ***not*** *depend on* n*.* ***For all*** k*, there exists* n *with* $1/3^n \leq 1/(3+k)$ *so that by* $\{0\}$*-Rewriting,* $[1/3^n, 1/(3+k)]$ *is rewritten to* $[0, 0]$*. This leads to step_5, i.e. stop. Hence* ***for all*** k*,* $\mathcal{BC}(\mathcal{A})([1/3, 1/(3+k)])$ ***does*** *terminate normally.*

5.6.2 ***EXAMPLE Bad Convergence:*** *This example illustrates bad convergence of an approximate computation. More specifically, an approximate computation which is not a continuous approximate computation may not converge when applied to a sequence of elements. Suppose* $\{\rho_k, \alpha_k\}_k$ *is an approximate computation and let* $\{a_i\}_i$ *be a sequence which converges to* $a \in R$*. This example shows that* $\{\rho_k(a_k)\}$ *need not converge.*

Let R *be the real numbers and let* $\rho_k(a)$ *be defined as* $a + 1/(k \cdot a^2)$ *for non-zero* a*. Let* $\alpha_k(a)$ *be defined as* $a + 1/(k \cdot a^3)$ *for non-zero* a*. When* a *is zero both* $\rho_k(a)$ *and* $\alpha_k(a)$ *are zero. Finally, let* $\{a_i\}_i$ *be the sequence* $\{1/i\}_i$*. Then of course:* $a_i \to 0$*. But* $\rho_i(a_i) = i + 1/i$ *so that:* $\rho_i(a_i) \to \infty$.

Acknowledgement

This research was supported in part by the United States Army Research Office through the Army Center of Excellence for Symbolic Methods in Algorithmic Mathematics (AC-

[82] Also, it is easy to check that in the usual BC-Arithmetic (4.1.1), for k large enough, $\mathcal{BC}(\mathcal{A})([1/3, 1/(3+k)])$ does not terminate normally.

[83] A looser bound than $\omega_\times$.

SyAM), Mathematical Sciences Institute of Cornell University, Contract DAAL03-91-C-0027 and the National Security Agency under contract MDA 904-95-H-1035. Both authors wish to express their gratitude to NTT (Nippon Telegraph and Telephone Corporation) Communication Science Laboratories for their support and encouragement of this project.

References

[1] Alefeld, G. and Herzberger, J., *Introduction to Interval Computations, Computer Science and Applied Mathematics, Academic Press* (1983).

[2] Buchberger, B., Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, *Chapter 6 in Multidimensional Systems Theory (N. K. Bose ed.), D. Reidel Publishing Company* (1985), 184-232.

[3] Carathéodory C., *Theory of Functions of a Complex Variable, Vol. I, Chelsea Publishing Co.* (1958).

[4] Hua, L. K., *Introduction to Number Theory, Springer-Verlag* (1982).

[5] Lang, S., *Algebra, Addison-Wesley* (1993).

[6] Shirayanagi, K., An Algorithm to Compute Floating Point Gröbner Bases, *Mathematical Computation with Maple V: Ideas and Applications (Ed. T. Lee), Birkhäuser* (1993), 95-106.

[7] Stoer, J. and Bulirsch, R., *Introduction to Numerical Analysis, Texts in Applied Mathematics* **12**, *Springer-Verlag* (1993).

[8] Winkler, F., A p-adic Approach to the Computation of Gröbner Bases, *Journal of Symbolic Computation* **6** 2&3 (1988), 287-304.