COMPUTING
SCIENCE

Title: Towards a Framework for Monotonic Approximations

Names: Anirban Bhattacharyya, Maciej Koutny, Brian Randell

TECHNICAL REPORT SERIES

No. CS-TR - 1506 June 2017

TECHNICAL REPORT SERIES

No. CS-TR-1506 **Date June 2017**

Title: Towards a Framework for Monotonic Approximations

Authors: Anirban Bhattacharyya, Maciej Koutny, Brian Randell

Abstract. This report presents an initial framework for modelling and analyzing algorithms that approximate fixed points of vector functions defined over multiple domains. We assume successive approximations of the algorithms form a well partial order, and prove that such (sequential) algorithms terminate with a unique result.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

Title and Authors

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR- 1506

Abstract. This report presents an initial framework for modelling and analyzing algorithms that approximate fixed points of vector functions defined over multiple domains. We assume successive approximations of the algorithms form a well partial order, and prove that such (sequential) algorithms terminate with a unique result.

About the authors

Dr Anirban Bhattacharyya is currently a Guest Member of Staff in the Advanced Model-Based Engineering and Reasoning (AMBER) group, School of Computing Science at Newcastle University. Anirban received a B.Sc. (Hons) in Mathematics from the University of London King's College in 1982, and an M.Sc. in Information Systems Engineering from South Bank Polytechnic in 1984. His PhD was in Formal Modelling and Analysis of Dynamic Reconfiguration of Dependable Systems from Newcastle University in 2013, supervised by Professor John Fitzgerald. For his PhD, Anirban extended the process algebra CCS with a special process (termed a fraction process) and overloaded the semantics of the parallel composition operator in order to model the runtime evolution of a system abstractly.

Maciej Koutny is currently a Professor of Computing Science in the School of Computing Science, Newcastle University. He received his MSc (1982) and PhD (1984) in Applied Mathematics from the Warsaw University of Technology, Poland. In 1985 he joined the then Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. In 1986 he became a Lecturer in Computing Science at Newcastle, and from 1994 to 2000 he held an established Readership at Newcastle University. His research interests centre on the theory of distributed and concurrent systems, including both theoretical aspects of their semantics and application of formal techniques to the modelling, synthesis, and verification of such systems; in particular, model

checking based on net unfoldings. He has also investigated non-interleaving semantics of priority systems, and the relationship between temporal logic and process algebras. He has been working on the development of a formal model combining Petri nets and process algebras as well as on Petri net based behavioural models of membrane systems.

Professor Brian Randell graduated in Mathematics from Imperial College, London in 1957 and joined the English Electric Company where he led a team that implemented a number of compilers, including the Whetstone KDF9 Algol compiler. From 1964 to 1969 he was with IBM in the United States, mainly at the IBM T.J. Watson Research Center, working on operating systems, the design of ultra-high speed computers and computing system design methodology. He then became Professor of Computing Science at the University of Newcastle upon Tyne, where in 1971 he set up the project that initiated research into the possibility of software fault tolerance, and introduced the 'recovery block' concept.

Suggested keywords

Generic framework, Noetherian order approximations, fixed points of vector functions, structured occurrence nets.

Towards a Framework for Monotonic Approximations

Anirban Bhattacharyya, Maciej Koutny, Brian Randell

School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
{Anirban.Bhattacharyya, Maciej.Koutny, Brian.Randell}@ncl.ac.uk

Abstract. This report presents an initial framework for modelling and analyzing algorithms that approximate fixed points of vector functions defined over multiple domains. We assume successive approximations of the algorithms form a well partial order, and prove that such (sequential) algorithms terminate with a unique result.

1 Introduction

Our aim is to investigate schemes for approximating fixed points of vector functions defined over multiple domains. A particular application of such schemes is structured occurrence nets (SONs) [1] and their extensions.

2 Sequential approximation

We first formulate a general statement of the approximation problem we will be addressing, then demonstrate two basic results that will be used in the specific context.

Assume a Cartesian domain $\mathbf{D} = D_1 \times \cdots \times D_n$ consisting of $n \geq 1$ implicitly ordered and indexed sub-domains D_i. Each D_i represents a set of possible approximations of some parameter in a system. For example, D_i can be the set of possible closed interval approximations D_E of a time parameter associated with node i of a SON, where the interval end-points take values from a discrete set E of reals (i.e. for all $e, e' \in E$, there are only finitely many $e'' \in E$ such that $e \leq e'' \leq e'$).

Therefore, a vector $\mathbf{d} = (d_1, \ldots, d_n) \in \mathbf{D}$ can be regarded as an approximation of n different parameters. In the dynamic scenario we consider in this paper, these approximations can be inaccurate initially, but through their repeated re-calculation (or re-evaluation) they can be improved, ending with approximations that cannot be improved further, and (therefore) can be regarded as optimal (i.e. the best). To capture such a scenario, we need additional notions and notations, described as follows:

1. We need to compare approximations of individual parameters. Therefore, we assume there is a well partial (i.e. Noetherian) order $\preceq_i$ for each sub-domain

D_i. That is, there is no infinite $\prec_i$-descending chain $\cdots \prec_i d_2 \prec_i d_1 \prec_i d_0$. Intuitively, $d \preceq_i h$ implies that d is at least as good as h as an approximation of the actual value of the i-th parameter, and that moreover, the initial approximation cannot be improved indefinitely.
In the case of interval approximations D_E, we can define $[e, e'] \prec_E [e'', e''']$ if $e'' \leq e$ and $e' \leq e'''$. Note that $\prec_E$ is a well partial order as E is a discrete set of reals.

2. The orderings $\preceq_1, \ldots, \preceq_n$ give rise to a well partial order $\preceq$ on $\mathbf{D}$, defined component-wise so that, for all $\mathbf{d} = (d_1, \ldots, d_n)$ and $\mathbf{h} = (h_1, \ldots, h_n)$ in $\mathbf{D}$, we have $\mathbf{d} \preceq \mathbf{h}$ if $d_i \preceq_i h_i$ for every $i \leq n$.

3. The improvement of parameter approximations will be represented by the functions
$$f_i : \mathbf{D} \to D_i \quad \text{for every } i \leq n$$
That is, each improvement is 'local' in the sense of providing possibly better approximation for just one parameter. However, at this point, we do not assume that improvements are based on any kind of local information. Instead, we make two basic assumptions about the improvement of approximations implemented by the f_i functions:
$$f_i(\mathbf{d}) \preceq_i d_i \quad \text{for every } \mathbf{d} = (d_1, \ldots, d_n) \in \mathbf{D} \tag{1}$$
$$f_i(\mathbf{d}) \preceq_i f_i(\mathbf{h}) \quad \text{for all } \mathbf{d}, \mathbf{h} \in \mathbf{D} \text{ satisfying } \mathbf{d} \preceq \mathbf{h} \tag{2}$$
Both assumptions are natural: (1) means that applying f_i improves the current approximation of the i-th parameter, and (2) that starting from a better approximation is beneficial.
We lift each f_i to
$$\mathbf{f}_i : \mathbf{D} \to \mathbf{D}$$
so that for every $\mathbf{d} \in \mathbf{D}$, $\mathbf{f}_i(\mathbf{d})$ is $\mathbf{d}$ with its i-th component replaced by $f_i(\mathbf{d})$. Intuitively, $\mathbf{f}_i$ represents the change to a global approximation resulting from an improvement of the approximation of the i-th parameter. We also denote
$$\mathbf{F} = \{\mathbf{f}_1, \ldots, \mathbf{f}_n\}$$

4. A vector $\mathbf{d} = (d_1, \ldots, d_n)$ in $\mathbf{D}$ is defined to be a *stable approximation* if
$$\mathbf{f}_i(\mathbf{d}) = \mathbf{d} \quad \text{for every } \mathbf{f}_i \in \mathbf{F}$$
Intuitively, stable approximations are those that we aim to derive through repeated improvements by applying the functions in $\mathbf{F}$, starting from some initial approximation.

Given an arbitrary initial approximation $\mathbf{d}$, there can be different stable approximations $\mathbf{h}$ satisfying $\mathbf{h} \preceq \mathbf{d}$, and we are interested in finding 'the best' $\mathbf{h}$. An intuitive criterion for selecting 'the best' stable approximation is that it

should not exclude any combination of parameter values that is consistent with the initial approximation $\mathbf{d}$. Intuitively, each improvement function $\mathbf{f}_i$ is assumed to be 'safe', that is, its application does not rule out any previously consistent combination of parameter values. Therefore, if no other means for improving the current approximation of parameters is available, the only way to proceed is to apply one of the functions in the set $\mathbf{F}$.

To prepare the ground for the results concerning parameter approximations, we need a notion capturing *an arbitrary* process of successive improvements resulting from applications of the functions in the set $\mathbf{F}$.

A finite sequence $\sigma = \mathbf{f}_{i_1} \dots \mathbf{f}_{i_k}$ $(k \geq 0)$ of functions in $\mathbf{F}$ can itself be interpreted as a function:

$$\sigma : \mathbf{D} \to \mathbf{D}$$

such that, for every $\mathbf{d} \in \mathbf{D}$:

$$\sigma(\mathbf{d}) = \mathbf{f}_{i_k}(\dots \mathbf{f}_{i_1}(\mathbf{d}) \dots)$$

In other words, such a σ can be seen as a finite improvement process. We say that $\mathbf{h} \in \mathbf{D}$ is *sequentially reachable* from $\mathbf{d} \in \mathbf{D}$ if there is $\sigma \in \mathbf{F}^*$ such that $\mathbf{h} = \sigma(\mathbf{d})$.

It turns out that the process of applying successive improvements is monotonic.

Theorem 1 (monotonicity). *For every finite sequence $\sigma \in \mathbf{F}^*$:*

1. *$\sigma(\mathbf{d}) \preceq \mathbf{d}$, for every $\mathbf{d} \in \mathbf{D}$*
2. *$\sigma(\mathbf{d}) \preceq \sigma(\mathbf{h})$, for all $\mathbf{d}, \mathbf{h} \in \mathbf{D}$ satisfying $\mathbf{d} \preceq \mathbf{h}$*

Proof. Follows from (1) and (2) above, and straightforward induction. □

Moreover, every initial approximation can be successfully turned into a stable approximation.

Theorem 2 (reachability of stable approximations). *For every $\mathbf{d} \in \mathbf{D}$, there exists a unique stable approximation $\mathbf{d}_{stable}$ sequentially reachable from $\mathbf{d}$.*

Proof. We first show that there is at least one stable approximation sequentially reachable from $\mathbf{d}$. Let $\sigma = \mathbf{f}_1 \dots \mathbf{f}_n$, $\mathbf{d}_0 = \mathbf{d}$, and for every $i \geq 1$, $\mathbf{d}_i = \sigma(\mathbf{d}_{i-1})$. Clearly, each $\mathbf{d}_i$ is sequentially reachable from $\mathbf{d}$. By Theorem 1(1), we have:

$$\dots \preceq \mathbf{d}_i \preceq \dots \preceq \mathbf{d}_1 \preceq \mathbf{d}_0$$

and so, as $\preceq$ is a well partial order, there is $k \geq 1$ such that:

$$\mathbf{d}_k = \mathbf{d}_{k+1} = \mathbf{d}_{k+2} = \dots = \mathbf{d}_{stable}$$

We observe that since each $\mathbf{f}_i \in \mathbf{F}$ occurs in σ, $\mathbf{d}_{stable}$ is a stable approximation sequentially reachable from $\mathbf{d}$.

We now show that $\mathbf{d}_{stable}$ is a unique stable approximation sequentially reachable from $\mathbf{d}$. Let $\mathbf{h} = \mathbf{d}_{stable}$ and σ be as above. Suppose that $\mathbf{h}'$ is a stable

approximations sequentially reachable from $\mathbf{d}$. Hence there is $\sigma' \in \mathbf{F}^*$ such that $\mathbf{h}' = \sigma'(\mathbf{d})$. By Theorem 1(1), we get:

$$\sigma(\mathbf{d}) \preceq \mathbf{d} \quad \textit{and} \quad \sigma'(\mathbf{d}) \preceq \mathbf{d}$$

Hence, by Theorem 1(2), we obtain:

$$\sigma'(\sigma(\mathbf{d})) \preceq \sigma'(\mathbf{d}) \quad \textit{and} \quad \sigma(\sigma'(\mathbf{d})) \preceq \sigma(\mathbf{d})$$

Thus, since $\sigma(\mathbf{d})$ and $\sigma'(\mathbf{d})$ are stable:

$$\sigma(\mathbf{d}) \preceq \sigma'(\mathbf{d}) \quad \textit{and} \quad \sigma'(\mathbf{d}) \preceq \sigma(\mathbf{d})$$

As a result, $\sigma(\mathbf{d}) = \sigma'(\mathbf{d})$, and so $\mathbf{h} = \mathbf{h}'$. □

Thus, we have demonstrated that for each initial parameter approximation $\mathbf{d}$, there is 'the best' sequentially reachable stable parameter approximation $\mathbf{d}_{stable}$ that can be used in place of $\mathbf{d}$. Moreover, in the proof of the first part of Theorem 2, we provided a straightforward procedure for deriving $\mathbf{d}_{stable}$. Such a procedure is not unique (of course) and we will now introduce a wide range of procedures of this kind.

Our goal is the development of a strategy for successive applications of the members of $\mathbf{F}$, starting from an initial parameter approximation $\mathbf{d} \in \mathbf{D}$, that leads to the stable approximation $\mathbf{d}_{stable}$ in a finite number of steps. Clearly, not all strategies for applying the functions in $\mathbf{F}$ would normally work. For example, repeatedly applying just one function, say $\mathbf{f}_1$, is unlikely to deliver a stable approximation in all but a very few special cases. However, it turns out there is a wide range of strategies for applying the functions in $\mathbf{F}$ that always succeed, and moreover deliver the deterministic result $\mathbf{d}_{stable}$, even though the process of applying the functions is non-deterministic. To show this, we consider the following meta-algorithm:

```
algorithm seqImprove

input x = d
while x not stable do
   choose f_i
   x = f_i(x)
return x
```

In the above, it is assumed that the procedures for choosing $\mathbf{f}_i$ and applying $\mathbf{f}_i$ are both *effective* (hence, checking for the stability of $\mathbf{x}$ is also effective), and that the algorithm is *fair*. *Fairness* means that if a run of *seqImprove* is non-terminating, then each of the functions $\mathbf{f}_i$ has been selected infinitely many times. Other than that, the selection procedure is completely arbitrary; in particular, it can be non-deterministic. An example of a fair algorithm implementation is *cyclicImprove* which selects the $\mathbf{f}_i$ functions in a fixed cyclic manner as in the

proof of Theorem 2 (of course, the algorithm stops if at any stage a stable approximation $\mathbf{x}$ has been obtained).

Despite allowing a highly unrestricted way of applying the improvement functions in $\mathbf{F}$, it turns out that *seqImprove* always terminates, and so by Theorem 2, delivers the unique stable approximation sequentially reachable from the initial input approximation.

Theorem 3. *seqImprove always terminates.*

Proof. Suppose that *seqImprove* does not terminate for $\mathbf{d} \in \mathbf{D}$. This means that there is an infinite sequence $i_1, i_2, i_3, ...$ of integers from the set $\{1, \ldots, n\}$ such that the infinite sequence:

$$\mathbf{d}_0 = \mathbf{d}, \mathbf{d}_1 = \mathbf{f}_1(\mathbf{d}_0), \ldots, \mathbf{d}_i = \mathbf{f}_i(\mathbf{d}_{i-1}), \ldots$$

is the infinite sequence of the consecutive values of the vector variable $\mathbf{x}$; moreover, none of these values is a stable approximation. It then follows from Theorem 1(1), that

$$\cdots \preceq \mathbf{d}_i \preceq \cdots \preceq \mathbf{d}_1 \preceq \mathbf{d}_0$$

and so, as $\preceq$ is a well order, there is $k \geq 1$ such that

$$\mathbf{d}_k = \mathbf{d}_{k+1} = \mathbf{d}_{k+2} = \cdots = \mathbf{h}$$

Now, since the choice of the function application in *seqImprove* is fair, each $\mathbf{f}_i \in \mathbf{F}$ occurs in the sequence $\mathbf{f}_{i_k}\mathbf{f}_{i_{k+}}\mathbf{f}_{i_{k+2}}\ldots$ at least once. However, this implies that $\mathbf{h}$ is a stable approximation, which yields a contradiction. Hence, *seqImprove* always terminates. □

Therefore, we conclude that any scheme which chooses the functions $\mathbf{f}_i$ in a fair way leads to a terminating algorithm for finding the unique stable approximation sequentially reachable from the initial input approximation. For example, the following algorithm implements the cyclic scheme used in the proof of Theorem 2:

```
algorithm cyclicImprove

// f_1 ... f_n any enumeration of F
input x = d
i = 1
while x not stable do
   i = (i mod n) + 1
   x = f_i(x)
return x
```

2.1 The case of acyclic data dependencies

A SON is a finite directed acyclic graph, each node of a SON has a finite number of parameters, and each parameter takes its value from an interval defined by

end-points taken from a discrete set (see [2]). SONs satisfy conditions 1 – 4 defined in Section 2, and (therefore) both *seqImprove* and Theorem 3 are applicable to SONs.

It is important to investigate how the meta-algorithm *seqImprove* can be optimised for specific domains and improvement functions so that its efficiency can be increased significantly. One approach is to find 'the best' function selection mechanism. This can depend on the *dependencies* between parameters, and hence between their respective domains. For example, each function f_i may depend only on a subset of 'neighbouring' domains. The relationship between the topology of these dependencies (which is related to the SON graph), and the actual definitions of the functions could be a rich field for our investigations.

We start by considering the case where the dependencies between the domains are acyclic. That is, there is an acyclic and reflexive relation $\rightsquigarrow$ on $\{1, \ldots, n\}$ such that, for all $i \leq n$ and $\mathbf{d} = (d_1, \ldots, d_n), \mathbf{h} = (h_1, \ldots, h_n) \in \mathbf{D}$, we have:

$$\mathbf{f}_i(\mathbf{d}) = \mathbf{f}_i(\mathbf{h})$$

whenever $d_i = h_i$ and $d_j = h_j$, for all $j \leq n$ satisfying $j \rightsquigarrow i$. It can be shown that $\mathbf{f}_i(\mathbf{d}) \preceq_i \mathbf{f}_i(\mathbf{h})$ provided that $d_i \preceq_i h_i$ and $d_j \preceq_j h_j$, for all $j \leq n$ satisfying $j \rightsquigarrow i$.

Of course, we could apply *cyclicImprove* in this case, but this would be very inefficient (as in many cases). Instead, we can sort the $\mathbf{f}_i$ functions in a topological order, and then select them following the obtained sequence. Moreover, for each selected $\mathbf{f}_i$, we apply it repeatedly until no further improvement is made, in the following way:

```
algorithm topsortImprove

// f_1 ... f_n any topologically sorted enumeration of F
input x = d
for i = 1 to n do
  continue = true
  while continue do
    y = x
    x = f_i(x)
    if x = y then continue = false
return x
```

In other words, we fully evaluate the parameters in the order determined by the acyclic dependence relation.

We can see that the algorithm will, as a result, reach a stable approximation so that there is no need to select any function after that.

Theorem 4. *For each* $\mathbf{d} \in \mathbf{D}$*, the algorithm topsortImprove always terminates and returns* $\mathbf{d}_{stable}$*.*

Proof. We can assume that the ordering of the domains in $\mathbf{D}$ is exactly the same as the topologically sorted enumeration of $\mathbf{F}$ in the definition of *topsortImprove*. We first observe that, for all $i \leq n$ and $\mathbf{v} \in \mathbf{D}$ there is $k \geq 1$ such that:

- $\mathbf{f}_i^k(\mathbf{v}) \preceq \mathbf{f}_i^{k-1}(\mathbf{v}) \preceq \cdots \preceq \mathbf{f}_i^2(\mathbf{v}) \preceq \mathbf{f}_i(\mathbf{v}) \preceq \mathbf{v}$ (by definition of $\mathbf{f}_i$); and
- $\mathbf{f}_i^{k+1}(\mathbf{v}) = \mathbf{f}_i^k(\mathbf{v})$ (because $\preceq$ is a well partial order on $\mathbf{D}$).

This implies that *topsortImprove* always terminates (because n is finite).

Now suppose that $\mathbf{d}_i$ is the value of $\mathbf{x}$ after the i-th pass of the for-loop. To conclude the proof, by Theorem 2, it suffices to show that $\mathbf{d}_n$ is stable. We observe that, for every $i \leq n$, by the assumed topological ordering of the domains in $\mathbf{D}$ and the fact that the first i parameters of $\mathbf{d}_i, \mathbf{d}_{i+1}, \ldots, \mathbf{d}_n$ are identical, we obtain that the i-th parameter in $\mathbf{d}_i, \mathbf{f}_i(\mathbf{d}_i), \mathbf{f}_i(\mathbf{d}_{i+1}), \ldots, \mathbf{f}_i(\mathbf{d}_n)$ is the same. Thus $\mathbf{d}_n$ is stable, completing the proof. □

If the functions $\mathbf{f}_i$ are idempotent, that is, for every $\mathbf{d} \in \mathbf{D}$:

$$\mathbf{f}_i(\mathbf{f}_i(\mathbf{d})) = \mathbf{f}_i(\mathbf{d})$$

then we can simplify algorithm *topsortImprove* so that each improvement function is applied exactly once, to obtain the following:

```
algorithm simpletopsortImprove

// f_1 ... f_n any topologically sorted enumeration of F
input x = d
for i = 1 to n do
  x = f_i(x)
return x
```

Theorem 5. *If all functions in* $\mathbf{F}$ *are idempotent then for each* $\mathbf{d} \in \mathbf{D}$*, the algorithm simpletopsortImprove always terminates and returns* $\mathbf{d}_{stable}$*.*

Proof. If all functions in $\mathbf{F}$ are idempotent then the body of the `for` loop in algorithm *topsortImprove* becomes functionally equivalent to the body of the `for` loop in algorithm *simpletopsortImprove* (by definition of idempotency), which implies *topsortImprove* and *simpletopsortImprove* become functionally equivalent (by their definitions),[1] which implies algorithm *simpletopsortImprove* always terminates and returns $\mathbf{d}_{stable}$ (by Theorem 4). □

3 Future Work

The framework will be extended to represent parallel approximation algorithms. We will also use the framework to examine the relationship between the topology of parameter/domain dependencies and function definitions in order to develop efficient SON-based approximation algorithms.

[1] Notice that *simpletopsortImprove* is more efficient than *topsortImprove* as the former always involves a single call of each $\mathbf{f}_i$, whereas the latter involves one or two calls of each $\mathbf{f}_i$. This justifies having a separate algorithm for idempotent $\mathbf{f}_i$ functions.

References

1. M.Koutny, B.Randell: Structured Occurrence Nets: A Formalism for Aiding System Failure Prevention and Analysis Techniques. Fundamenta Informaticae 97(1-2), 2009.
2. A.Bhattacharyya, B.Li, B.Randell: Time in Structured Occurrence Nets. Proceedings of PNSE'16, `http://ceur-ws.org/Vol-1591`, 2016.