

Algebraic Polynomial-based Synthesis for Abstract Boolean Network Analysis

Peter Backeman^{1,2}, Sara-Jane Dunn², Boyan Yordanov², and
Christoph M. Wintersteiger²

¹ Uppsala University, Uppsala, Sweden

² Microsoft Research

Abstract

Function synthesis is the problem of automatically constructing functions that fulfil a given specification. Using templates to limit the form of those functions is a popular way of reducing the search-space while still allowing interesting functions to be found. We present an investigation of restrictions to templates over Boolean functions of polynomial shape, based on their algebraic normal form. These polynomials are then lazily created in such a way that completeness of the search is still guaranteed, while performance is improved. This method is then implemented using an SMT-solver (Z3) and illustrated on a biological problem, the goal of which is to synthesise (Boolean) gene regulatory networks that capture specifications derived from experimental measurements.

1 Introduction

Synthesis techniques aim towards the automated construction of correct-by-design systems from specifications of desired behaviour, often expressed as temporal logic specifications [14]. These techniques have been successfully applied in a number of fields, including software [9, 18] and hardware synthesis [1], but also in biology (e.g., [13, 7, 12]). For biological applications, and more broadly in the field of computational science, the construction of computational models that reproduce known or observed behaviour of a natural system is a major challenge. However, this can be framed as a synthesis problem in which the specification is the experimentally-observed behaviour (as observed in a ‘wet-lab’ setting), together with known or assumed constraints, and the goal is the construction of a model that is consistent with these observations and assumptions.

Abstraction techniques are often applied as part of computational modelling, both to describe the system at a sufficient level of detail, and to increase the runtime performance of synthesis algorithms. For example, the genetic regulation that arises via complex biochemical processes, which governs many cellular processes can be represented by *Gene Regulatory Networks* (GRNs). Further, these networks can be modelled as *Boolean networks* (BNs), under the assumption that each involved gene can be abstracted into *active* or *inactive* states [11]. Considering such a model as a transition system, a Boolean function defines the next state of each component in terms of the states of its regulators (activators and repressors). The resulting models capture an abstract, qualitative representation of the detailed biochemical mechanisms involved, and have proven useful for studying different types of cellular behaviour, such as cell differentiation.

The challenge of constructing a BN for a given biological system lies in identifying the network topology, i.e. the interactions between components which comprise a model that reproduces all observed behaviour. To address this challenge, Dunn *et al.* [6] propose *Abstract Boolean Networks* (ABNs), which allow the construction of models where only some interactions are known, and putative ones are included only as optional interactions. An ABN thus

describes a set of unique, concrete topologies, in which the optional interactions are instantiated as present or absent.

An SMT-based synthesis approach, implemented as part of the RE:IN tool [6], is used to enumerate individual, concrete BNs from the ABN, and to study the constrained interaction network, for example by identifying *required interactions*, which appear in every concrete model that satisfies the specification. While the RE:IN approach reveals important properties, the enumeration of all valid models from the ABN is infeasible due to the usually large number of concrete solutions capable of reproducing the observed behaviour. A naive follow-up approach lies in the enumeration of this large set of solutions and then to search for patterns in the results. Finally, RE:IN does not provide a concise representation of the ABN constrained against the experimental observations, which can be queried efficiently or analysed further, e.g., to develop new biological experiments for validation of the model.

An alternative strategy to the iterative enumeration of concrete BNs from a constrained ABN is to synthesise a template function, parametrised by unknown, optional interactions, that describes all concrete models consistent with the observations. Such a representation of all consistent models is easily interrogated to reveal dependencies between interactions and expose all different mechanisms capable of producing the observed behaviour. If sufficiently concise, this representation also serves as a valuable tool for understanding system properties and guiding further biological studies. For the specific application to ABNs, function synthesis focuses on restricted sets of Boolean functions, through the instantiation of parametrised function templates. However, the problem of choosing appropriate templates is non-trivial. Here, we focus on this problem and propose a lazy algorithm that combines templates in a sound and complete manner.

2 Related Work

Pnueli and Rosner were among the first to suggest a theoretical framework for program synthesis and accompanying algorithms, based on automata theory and temporal logic [14]. Since then, this and similar techniques have been used in many applications, with varying degrees of success. Examples include software synthesis, such as the recent work by Gulwani *et al.* [9] and Solar-Lezama [18], who both employ different algorithms and abstraction techniques based on template instantiation to reduce the vast space of functions that would otherwise have to be investigated. It is interesting to note that in recent years, there has also been an increased interest in software synthesis from *natural language* specifications, such as by Gvero and Kuncak [10] and Raza *et al.* [15].

Hardware synthesis has been a subject of research for many years. Given that some systems are required to be finite-state, or have a very limited number of inputs and output, the resulting synthesis problems are of moderate degree and often practically feasible. To this end, various modelling, verification, and synthesis techniques are applied, many of them based on automata-theoretic principles like the various classes of automata used in synthesis of temporal logic specifications; recent examples include distributed synthesis algorithms by Chatterjee *et al.* [3] and robust system synthesis by Bloem *et al.* [2]. Other formal bases are of course considered as well, e.g., Asarin *et al.* [1] abstract the systems to be synthesised by a finite collection of linear systems. In computational biology there are a number of different approaches to synthesis problems for various specification logics. For instance, Kugler *et al.* [13] consider the synthesis of Live Sequence Charts. As mentioned, Dunn *et al.* [6] consider the synthesis of BNs to model gene regulatory networks, and others like Fisher *et al.* [7] consider techniques tailored toward the efficient reconstruction of BNs from large collections of single-cell experimental data. Köksal *et*

al. [12] propose a modelling and specification language, as well as an embedding thereof into Scala, and they describe an efficient synthesiser able to find a BN for cell fate determination of *C. elegans*, based on wet-lab mutation experiments.

The synthesis algorithm that we propose in Sec. 4.2 is strongly inspired by more general quantifier instantiation techniques in the wider area of automated theorem proving. Our algorithm can be seen as an instance of Model-Based Quantifier Instantiation [8] with automated (and complete) function template refinement [19]. However, we implicitly exploit other heuristic techniques, such as E-matching [4], and their implementation in the Z3 SMT solver [5]. Recently approaches have emerged to support synthesis algorithms that are integrated directly into a theorem prover or SMT solver, e.g., by Reynolds *et al.* [17, 16].

3 Background

Originally introduced by Kauffman [11], Boolean networks (BNs) represent one particular class of gene regulatory network models, where every gene is represented by a Boolean variable indicating a gene’s state as enabled or disabled.

For many (if not most) interactions between genes, it is not known whether they are indeed present in a particular system and if so whether they are positive or negative, and it is very much on the agenda of computational biology to discover and establish the type of these interactions. For our models to encompass such partial knowledge, we add to each interaction a label that is either *optional* or *definite*. Where the presence of an optional interaction is unknown, a definite is assumed to be present. We thus define *abstract* Boolean networks over a set $G := \{g_1, \dots, g_n\}$ (in accordance with the semantics attached to them by Dunnet *al.* [6]), as an extension of non-deterministic finite state machines:

Definition 1 (Abstract Boolean Network). *An Abstract Boolean Network (ABN) is a non-deterministic finite-state machine with*

- *finite state-space* $Q = \mathbb{B}^n = \langle g_1, \dots, g_n \rangle$
- *empty input alphabet,*
- *set of initial states* $Q_i \subseteq Q,$
- *set of final states* $F \subseteq Q,$ *and*
- *transition relation* $\delta : Q \times Q = (q, \langle r_1(q), \dots, r_n(q) \rangle),$ *for all* $q \in Q$ *with an update function* r_i *for each gene* $g_i.$

ABN models allow us to incorporate uncertainty about the precise topology of a BN. Some of the many concrete BNs (CBNs) captured by an ABN may produce behaviour that is consistent with experimental observations of the biological system, while others might not. Therefore a set of experimentally-derived constraints is imposed over the behaviour of ABNs to exclude networks that are inconsistent with observations. We then say that an ABN *satisfies* an observation iff there exists at least one concrete BN that satisfies all such constraints.

Experimental observations are represented as reachability predicates over the states of some or all components at different time steps during the execution of the system. Every concrete execution of an ABN is a sequence of states $q_0, \dots, q_k.$ An observation is a set of concrete traces, which may be specified by a predicate that restricts the set of all traces, e.g., all traces starting in a particular starting state q_0 and that reach a final state $q_f.$

Definition 2 (Observation satisfaction). *An ABN satisfies an observation iff there exists at least one concrete BN, for which there exists at least one trace that satisfies all conditions on initial, final, and other states.*

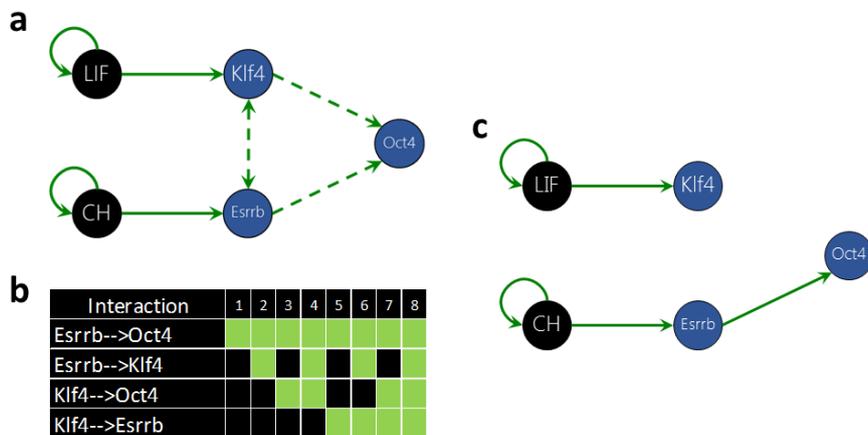


Figure 1: A small example ABN

Example 1. Consider the network in Fig. 1(a). This depicts a small ABN where each circle represents a gene, each solid edge represents a definite interaction, and each dashed edge represents an optional interaction (the bidirectional edge between $Klf4$ and $Esrrb$ represents two optional interactions). Fig. 1(b) shows all unique instantiations of optional interactions forming a CBN consistent with some set of observations (not shown here). Each column represents one CBN, where a green square indicates that the corresponding interaction was included, and a black square indicates that it was not. Fig. 1(c) shows the CBN corresponding to column 1 of Fig. 1(b) where only the first interaction is instantiated.

In previous work, Dunn *et al.* [6] aimed at enumerating and analysing a set of CBNs (as well as finding a *minimal* CBN) consistent with all experimental observations. They enumerated all possible network topologies, then formulated a query that posits that a network satisfies the observational constraints, and dispatched each of those queries to an SMT solver (Z3 [5]). The set of solutions obtained using this approach describes a set of CBNs consistent with all observations.

4 Polynomial-based Synthesis

We start by defining parametrised templates that enable us to transform the problem of synthesising (finding) a function into an equivalent problem of finding parameters or coefficient values. Note that our goal is not to synthesise or find update functions r_i of ABNs. Instead, the goal is to synthesise a concise description of the topology and the properties of ABNs (e.g., observational constraints), also encoded in Boolean functions. Such a template can be used to construct SMT queries to find the coefficients required to model certain properties of a system.

To this end, we observe that all Boolean functions have an Algebraic Normal Form (ANF) polynomial, which provides a canonical representation of functions of n inputs in terms of 2^n coefficients (in the worst case). Further, this representation also allows us to *order* Boolean functions in a manner that enables us to define a class of abstract function refinement procedures. We begin by stating the basic completeness property of ANF.

Theorem 1 (Folklore). *Every Boolean formula f over variables $\{x_1, \dots, x_n\}$ has an equivalent and unique Algebraic Normal Form (ANF) taking the form of*

$$\begin{aligned} f(x_1, \dots, x_n) = & a_0 \oplus \\ & (a_1 \wedge x_1) \oplus \dots \oplus (a_n \wedge x_n) \oplus \\ & (a_{1,2} \wedge x_1 \wedge x_2) \oplus \dots \oplus (a_{n-1,n} \wedge x_{n-1} \wedge x_n) \oplus \\ & \dots \oplus \\ & a_{1,\dots,n} \wedge x_1 \wedge \dots \wedge x_n \end{aligned}$$

for some $a_1, \dots, a_{1,\dots,n} \in \{0, 1\}$ where \oplus is the exclusive-or operator.

We chose this representation of Boolean functions, because it allows us to easily identify and to specify some simple sub-classes of functions; for instance, constants are indeed just a single constant term, linear functions contain only monomials, etc. Mathematically, any other representation of Boolean functions may be equivalent, but we suspect that presenting functions in ANF may help computational biologists to interpret our results. To make this more explicit, we define algebraic polynomials as a symbolic representation of Boolean functions:

Definition 3. *Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. A Boolean formula p over X , i.e., $p(x_1, \dots, x_n)$, is an algebraic normal form polynomial (or simply algebraic polynomial) if it is of the form*

$$p(x_1, \dots, x_n) = c_0 \oplus t_1 \oplus \dots \oplus t_k$$

where $k < 2^n$, each t_i is distinct from all others, and of the form $c_i \wedge \bigwedge_{l=0}^{|X_j|} x_l$, where X_j is the j -th subset of 2^X and x_l is the l -th item in X_j , and $\{c_i \mid 0 \leq i < 2^n\}$ is a set of distinct Boolean constants which we call the ‘coefficients’.

An algebraic polynomial is conveniently described as a subset of the ANF monomials in its representation, and it represents a *set* of Boolean functions (those representable by concrete choices of values for the coefficients c_i). This definition also allows us to *order* functions based on the lexicographical ordering of the coefficients. The maximal polynomial is the one equivalent to the $(2^n - 1)$ -monomial ANF (with all coefficients $c_i = 1$).

Lemma 1. *For every non-maximal algebraic polynomial p , there is a polynomial $p' \succ p$ s.t. every monomial in p is included in p' .*

4.1 Searching for Boolean Functions

Given that we have established an ordering of ANF polynomials, we are now able to define directed search strategies that are designed to exhaust a particular class of functions. The search for a particular function representable within an algebraic polynomial is ultimately performed by checking a propositional formula for satisfiability (in practice by an SMT solver), such that a satisfying assignment to all c_i identifies a specific function.

Example 2. *The algebraic polynomial $c \oplus (c_1 \wedge x_1 \wedge x_2)$ corresponds to the Boolean functions $\{f(x_1, x_2) = 0, f(x_1, x_2) = 1, f(x_1, x_2) = x_1 \wedge x_2, f(x_1, x_2) = 1 \oplus (x_1 \wedge x_2)\}$.*

Looking at Ex. 2, a query could be posted to a SMT solver that checks whether any of the functions corresponding to the given ANF is equivalent to a sought after Boolean function ($\phi(\bar{x})$). Such a query could be on the following form:

$$\varphi = \exists c, c_1. \forall \bar{x}. \phi(x) \Leftrightarrow c \oplus (c_1 \wedge x_1 \wedge x_2)$$

This query will only consider two of the arguments in \bar{x} , this might of course not be enough, in which case a larger ANF must be tried. The key to efficiency lies in knowing which monomials to include in the ANF to make it feasible to find suitable coefficients quickly.

The universal quantifier is necessary to ensure the ANF instantiated with coefficients behaves like the given function. It could be eliminated by using enumeration of the whole domain (or partially, using some kind of bounded model checking), this has not been further developed at this point. However, this does not ensure a quantifier-free formula, since $\phi(\bar{x})$ might contain quantifiers (which is the case in many applications, including ABN synthesis).

Lem. 1 tells us that every Boolean function represented by p has an equivalent representation in some $p' \succ p$. If p is maximal, then all Boolean functions are included. We may therefore traverse any set of algebraic polynomials to search for a particular function, f , that has a representation in p , yet we are still able to guarantee to find any Boolean function if we include (at least) the maximal polynomial. In practice, this is of course (too) expensive. Often, a much smaller set of functions, or a set of smaller functions, is sufficient.

This leads us to a simple, sound, and complete search procedure: start with any algebraic polynomial p and see if it can represent f . If it cannot, then pick the next polynomial $p' \succ p$ according to the ordering and retry. This is repeated until a suitable polynomial has been found. The process is guaranteed to terminate since there is only a finite number of algebraic polynomials over a fixed set of variables, and the maximal algebraic polynomial is guaranteed to be able to represent f .

4.2 FIND and FIX

Let $\varphi(x)$ be a Boolean function and suppose that the problem is to synthesise another function $f_\varphi(x) = \varphi(x)$, i.e., for each assignment to $x \in X$, we want $f_\varphi(x)$ to compute the same result as $\varphi(x)$, but preferably in a smaller or more ‘general’ representation. The idea is to slice the domain of assignments into subsets that are easily solvable, have small representations, and ultimately yield a description of f_φ that pairs predicates (functions that map inputs to 1 or 0), identifying parts of functions, with (function)-terms that efficiently describe the desired functional relationships between variables in φ . Formally, we are looking for a *model*, that is a formula equivalent to $\varphi(x)$ but in a different form or representation.

We propose an iterative procedure: we begin by stating the formula to be modelled, f_φ , and we construct an initial arbitrary model function $m = 0$, for $f_\varphi(x)$. We then search for a counter-example, c , to the current model, m , i.e. an assignment to the variables x such that $m(x) \neq \varphi(x)$. Then a predicate FIND and a function FIX is constructed s.t. $\text{FIND}(c) = 1 \wedge \forall x. \text{FIND}(x) \Rightarrow m(x) \neq \varphi(x)$ and $\forall x. \text{FIND}(x) \Rightarrow (\text{FIX}(x) = \varphi(x))$. A new, improved model is then easily constructed from those parts: $m'_\varphi(x) = \text{ITE}(\text{FIND}(x), \text{FIX}(x), m(x))$, where *ITE* is the if-then-else operator. The algorithm is summarised in Alg. 1.

This procedure can be relaxed in two ways while still maintaining completeness, *approximating* either the FIND or the FIX predicate. This means that in each iteration a counter-example c is generated, w.r.t. the current model m . The task is then to ‘patch’ the current model s.t. $m(c)$ now computes the correct value. We synthesise FIND and FIX, where FIND must identify at least one counter-example c , but is allowed to be imprecise elsewhere. The new model will be correct for c (per construction) and for no other function points will it be incorrect where it previously was correct (FIX being exact). Therefore, the number of inputs for which the new model is correct, will be guaranteed to be larger than for the older model (by at least one). The

```

 $m(x) \leftarrow 0;$ 
while  $\exists x.m(x) \neq \varphi(x)$  do
   $c \leftarrow x$  s.t.  $m(x) \neq \varphi(x);$ 
   $FD \leftarrow fd$  s.t.  $fd(c) \wedge (\forall x.fd(x) \Rightarrow m(x) \neq \varphi(x)) ;$ 
   $FX \leftarrow fx$  s.t.  $\forall x.fd(x) \Rightarrow fx(x) = \varphi(x) ;$ 
   $m(x) \leftarrow ITE(FD(x), FX(x), m(x));$ 
end

```

Algorithm 1: The FIND and FIX loop

algorithm is described in Alg. 2. The reasoning for approximate FIX is similar and summarised by Alg. 3.

```

 $m(x) \leftarrow 0$ 
while  $\exists c.m(c) \neq \varphi(c)$  do
   $FD \leftarrow fd$  s.t.  $fd(c)$ 
   $FX \leftarrow fx$  s.t.  $\forall x.$ 
     $fd(x) \Rightarrow fx(x) = \varphi(x)$ 
   $m(x) \leftarrow$ 
     $ITE(FD(x), FX(x), m(x))$ 
end

```

Algorithm 2: Approximate FIND

```

 $m(x) \leftarrow 0$ 
while  $\exists c.m(c) \neq \varphi(c)$  do
   $FD \leftarrow fd$  s.t.  $fd(c) \wedge$ 
     $\forall x.fd(x) \Rightarrow m(x) \neq \varphi(x)$ 
   $FX \leftarrow fx$  s.t.  $fx(x) = \varphi(x)$ 
   $m(x) \leftarrow$ 
     $ITE(FD(x), FX(x), m(x))$ 
end

```

Algorithm 3: Approximate FIX

4.3 Function Synthesis Reasoning

Combining FIND and FIX with algebraic polynomial function templates leads to a Boolean function synthesis procedure, as outlined in Alg. 4. It first looks for a counter-example ce to the

```

 $m \leftarrow 0 ;$ 
while  $\exists ce . m(ce) \neq \varphi(ce)$  do
   $p \leftarrow findRepresentingPolynomial(ce, p_0) ;$ 
  while  $\exists k . p(k) \wedge (\varphi(k) = 0)$  do
     $p \leftarrow nextRepresentingPolynomial(k, p) ;$ 
     $m \leftarrow ITE(p, 1, m) ;$ 
  end
end

```

Algorithm 4: Approx. FIND, Exact FIX-model search for formula φ

initial model m .¹ $findRepresentingPolynomial(ce, p_0)$ generates an algebraic polynomial that contains ce (starting from an arbitrary but fixed initial polynomial p_0) and $nextRepresentingPolynomial(k, p)$ generates an algebraic polynomial greater than p that contains k .

In $findRepresentingPolynomial$ as well as $nextRepresentingPolynomial$ there is a lot of room for heuristics. A simple way of finding the next representing polynomial is by considering all possible ANFs and selecting the next one according to a lexicographical ordering (which is easy to construct) until a suitable is found. This will often be very slow and, so better heuristics are necessary. Our prototype implementation (FSREIN) implements a method based

¹In essence, this is the query RE:IN uses to enumerate solutions.

on identifying relevant variables from unsatisfiable core reasoning, and only adds monomials over those variables that appear in the core. An important insight here is that any heuristic that always produces larger polynomials is guaranteed to be complete.

5 Comparison

Using the techniques presented in Sec. 4.3 a tool was implemented, FSREIN, which is capable of synthesizing a Boolean function describing all concrete Boolean network, for a given ABN, that are valid with respect to some observational constraints. We present in this section a comparison between this and the state-of-the-art tool for ABN reasoning (RE:IN [6]). Both tools, RE:IN and FSREIN, are sound and complete, and they will find all valid networks given sufficient time. Both tools have exponential worst-case runtime complexity.

Reduced Models. Consider the problem presented earlier in Fig. 1. The ABN shown in Fig. 1(a) is a module of the more extensive gene regulatory network that governs the mouse pluripotent embryonic stem cell (ESC) state dynamics. Experimental measurement of gene expression under different inputs allows us to define experimental observations of the pluripotency network, which we impose as constraints. Using RE:IN to enumerate consistent CBNs yields the table shown in Fig. 1(b). Eight unique CBNs are consistent with the experiments. From the table it is quite easy to see that the interaction $Esrrb \rightarrow Oct4$ is part of every solution, and it is thus a *required* interaction. The output of FSREIN are the two FIND-functions

$$\begin{aligned} & (Esrrb \rightarrow Oct4) \text{ OR} \\ & ((\text{not } Esrrb \rightarrow Oct4) = (Esrrb \rightarrow Klfnn4 \text{ and } Esrrb \rightarrow Oct4)) \end{aligned}$$

where the corresponding FIX functions are both 1, from which it is trivial to see (or to deduce) that $Esrrb \rightarrow Oct4$ is a required interaction.

For another, larger example (16 genes, 8 optional interactions, 2 experiments) a similar result was found. When RE:IN was executed it enumerated 96 solutions, while FSREIN produces the following functions:

$$\begin{aligned} & ((Nanog \rightarrow Sox2) \text{ AND } (Klf2 \rightarrow Oct4)) \text{ OR} \\ & ((Sall4 \rightarrow Sox2) \text{ AND } (Klf2 \rightarrow Oct4)) \end{aligned}$$

The running time for the FSREIN was in this instance three orders of magnitude slower than running the RE:IN-tool, which can of course be a major obstruction, but there is plenty of room to greatly improve the efficiency of the procedure. However, the major improvement is the readability of the model. The FSREIN model is easy to understand and clearly shows the interactions between genes that is in a more ‘natural’ form for computational biologists than the list of counter-examples obtained from RE:IN.

Benchmarks. To compare more quantitatively, we crafted a set of 39 simple benchmark problems on which to test both tools. Fig. 2 summarises the results after running each of the tools, comparing the

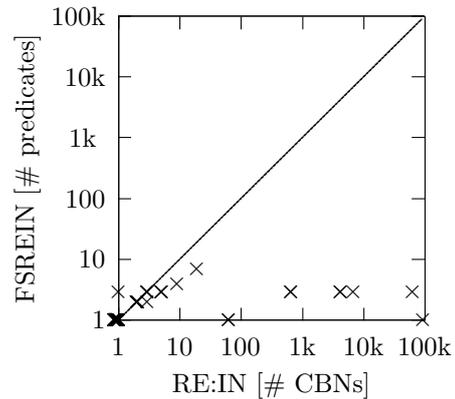


Figure 2: A comparison of the model size produced by RE:IN and FSREIN.

size of the model expressions (the number of enumerated models for RE:IN and the number of FIND and FIX-pairs for FSREIN) for each tool. Observe that FSREIN never produces a model larger than that of RE:IN, but it is interesting to see that in some cases it produces significantly smaller representations.

6 Conclusion

In this paper we present a function synthesis approach based on the FIND and FIX-strategy, using algebraic polynomials to describe sets of Boolean functions to be synthesised. We apply this technique to problems that arise in computational biology; specifically the abstract Boolean network synthesis problem, and show that in small instances it is able to produce functions describing the set of concrete Boolean networks that are consistent with the abstract network, and with experimental observations obtained through wet-lab experiments. The average runtime of our implementations is not better than the state-of-the-art, but it has the benefit of removing the burden of having to analyse a large set of enumerated models to find patterns within the data. Rather, the functions describing the networks are kept in a compact and expressive representation, which is much more amenable to scientific testing and interpretation.

References

- [1] Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, volume 999 of *LNCS*. Springer, 1995.
- [2] Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer, and Robert Könighofer. Synthesizing robust systems. *Acta Inf.*, 51(3-4), 2014.
- [3] Krishnendu Chatterjee, Thomas A. Henzinger, Jan Otop, and Andreas Pavlogiannis. Distributed synthesis for LTL fragments. In *FMCAD 2013*. IEEE, 2013.
- [4] Leonardo de Moura and Nikolaj Bjørner. Efficient e-matching for SMT solvers. In *CADE 2007*, volume 4603 of *LNCS*. Springer, 2007.
- [5] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS 2008*, volume 4963 of *LNCS*. Springer, 2008.
- [6] S.-J. Dunn, G. Martello, B. Yordanov, S. Emmott, and A. G. Smith. Defining an essential transcription factor program for naïve pluripotency. *Science*, 344(6188), 2014.
- [7] Jasmin Fisher, Ali Sinan Köksal, Nir Piterman, and Steven Woodhouse. Synthesising executable gene regulatory networks from single-cell gene expression data. In *CAV 2015*, volume 9206 of *LNCS*. Springer, 2015.
- [8] Yeting Ge and Leonardo de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In *CAV 2009*, volume 5643 of *LNCS*. Springer, 2009.
- [9] Sumit Gulwani. Program synthesis. In *Software Systems Safety*, volume 36 of *NATO Science for Peace and Security Series, D: Information and Communication Security*. IOS Press, 2014.
- [10] Tihomir Gvero and Viktor Kuncak. Interactive synthesis using free-form queries. In *ICSE 2015*. IEEE, 2015.
- [11] S.A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3), 1969.
- [12] Ali Sinan Köksal, Yewen Pu, Saurabh Srivastava, Rastislav Bodík, Jasmin Fisher, and Nir Piterman. Synthesis of biological models from mutation experiments. In *POPL 2013*. ACM, 2013.
- [13] Hillel Kugler, Cory Plock, and Amir Pnueli. Controller synthesis from LSC requirements. In *FASE 2009*, volume 5503 of *LNCS*. Springer, 2009.

- [14] Amir Pnueli and Roni Rosner. A framework for the synthesis of reactive modules. In *Intl. Conf. on Concurrency (Concurrency 1988)*, volume 335 of *LNCS*. Springer, 1988.
- [15] Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. Compositional program synthesis from natural language and examples. In *IJCAI 2015*. AAAI Press, 2015.
- [16] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett. Counterexample-guided quantifier instantiation for synthesis in SMT. In *CAV 2015*, volume 9207 of *LNCS*. Springer, 2015.
- [17] Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstic, Morgan Deters, and Clark W. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In *CADE 2013*, volume 7898. Springer, 2013.
- [18] Armando Solar-Lezama. Program sketching. *STTT*, 15(5-6):475–495, 2013.
- [19] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. Efficiently solving quantified bit-vector formulas. *FMSD*, 42(1):3–23, 2013.