Synthesis for Unbounded Bit-vector Arithmetic

Andrej Spielmann and Viktor Kuncak

School of Computer and Communication Sciences (I&C)
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract. We propose to describe computations using QFPAbit, a language of quantifier-free linear arithmetic on unbounded integers with bitvector operations. We describe an algorithm that, given a QFPAbit formula with input and output variables denoting integers, generates an efficient function from a sequence of inputs to a sequence of outputs, whenever such function on integers exists. The starting point for our method is a polynomial-time translation mapping a QFPAbit formula into the sequential circuit that checks the correctness of the input/output relation. From such a circuit, our synthesis algorithm produces solved circuits from inputs to outputs that are no more than singly exponential in size of the original formula. In addition to the general synthesis algorithm, we present techniques that ensure that, for example, multiplication and division with large constants do not lead to an exponential blowup, addressing a practical problem with a previous approach that used the MONA tool to generate the specification automata.

1 Introduction

Over the past decades, a number of decision procedures has been developed and integrated into satisfiability modulo theory (SMT) solvers. Among the primary uses of this technology so far has been verification and error finding. Recently, researchers started using this technology for software synthesis [11]. In the line of work on complete functional synthesis, researchers proposed to generalize decision procedures for infinite domains to *synthesis procedures* [7].

The basic idea is to describe fragments of code using formulas in a decidable logic. Such a formula specifies a relation between inputs and outputs. A synthesis procedure then compiles this formula into a program that maps inputs into outputs, and whose behavior corresponds to invoking a decision procedure on that particular constraint. The resulting program is guaranteed to satisfy the specification. Synthesis procedures have been described for, e.g., parameterized Presburger arithmetic [7], using a constructive version of quantifier elimination.

For domains such as integer arithmetic, automata-based methods can have a number of advantages compared to quantifier elimination, including the ability to support operations on unbounded bitvectors. Motivated by these observations, in related previous work [4] researchers considered synthesis of specifications expressed in weak monadic second-order logic of one successor (WS1S), which is equivalent to Presburger arithmetic with bitwise logical operators. In contrast to automata-based approaches to reactive synthesis [1, 2, 5, 8], this approach uses automata to encode relations on integers,

which means that the causality restriction of Church's synthesis problem does not apply. The synthesized function for this problem cannot always be given as a one-pass finite-state transducer. The approach [4] synthesizes a two pass transducer, where the first pass generates a sequence that abstracts the tree of possible executions, whereas the second pass processes this sequence backwards to choose an acceptable sequence of outputs. The previous implementation of this approach used the MONA tool [6] to transform the given specification formula into an automaton accepting a sequence of bits of combined input/output vectors. This implementation therefore suffered from the explicit-state representation used by MONA. The most striking problem is multiplication by constants, where a subformula $x = c * y$ leads to circuits of size proportional to c and thus exponential in the binary representation of c.

To overcome the difficulties with explicit-state representation used in the implementation of [4], in this paper we investigate an approach that directly uses circuit representations for both specifications and implementations. To avoid the non-elementary worst-case complexity [12] of transforming WS1S formulas to automata, we use as our specification language quantifier-free Presburger arithmetic with bitvector operations [9], denoted QFPAbit. We describe a polynomial-time transformation between sequential circuits and QFPAbit. We then present an algorithm for transforming sequential circuit representations of input/output relations into systems of sequential circuits that map inputs into outputs. The worst-case complexity of our translation is bounded by a singly exponential function of the specification circuit size. Building on this general result, we identify optimizations that exploit the structure of specifications to reduce the potential for exponential explosion. Our prototype implementation confirms the improved asymptotic behavior of this synthesis approach, and is available for download from `http://lara.epfl.ch/w/cisy`. Additional details of our constructions are available in the technical report [10].

2 Preliminaries

2.1 Quantifier-Free Presburger Arithmetic with Bit-vector Logical Operators

Presburger Arithmetic with Bit-vector Logical Operators is the structure of integers with addition and bit-vector logical operations acting on the binary two's complement representation of the integers. Let V be a finite set of variables. Let $c \in \mathbb{Z}$, $x \in V$, and $\%$ ranges over $=, \neq, <, \leq, >, \geq$. The following is the grammar of QFPAbitterms and formulas.

$$T := c \mid x \mid T + T \mid cT \mid \bar{\neg} T \mid T \,\bar{\wedge}\, T \mid T \,\bar{\vee}\, T$$
$$F := T \,\%\, T \mid \neg F \mid F \wedge F \mid F \vee F \mid F \rightarrow F \mid F \leftrightarrow F$$

Variables range over the set of integers $\mathbb{Z}$. The bitvector logical operators act on the two's complement encoding of numbers [9]:

$$\langle x_k, ...x_0 \rangle_{\mathbb{Z}} = -2^k x_k + \Sigma_{i=0}^{k-1} 2^i x_i.$$

A property of this encoding is that replicating the most significant bit does not change the value. This justifies our definition of the bit-vector operators because for any two

numbers we can always find encodings that have the same length. By *the* two's complement encoding of a number, we mean its shortest possible encoding. Given a QFPAbit formula F over the set of variables $V = \{x_1, ...x_n\}$, we say that a valuation $val : V \to \mathbb{Z}$ satisfies F if F is true when each occurrence of a variable x_i evaluates to $val(x_i)$. We say that F is satisfiable if there exists a valuation that satisfies F. Note that the identity $-x = \neg\, x + 1$ holds for all x.

We can use QFPAbit formulae to define languages over $\Sigma = \{0,1\}^n$. Let F be a QFPAbit formula over the variables $V = \{x_1, ...x_n\}$. Let $w \in \Sigma^+$ be a word of length m. By $w(j)$ denote the j-th letter of w, indexing from 0, so that the initial letter is denoted $w(0)$. Each $w(j)$ is a vector of dimension n, let $w_i(j)$ denote the the i-th coordinate of $w(j)$. Define a valuation $val_w : V \to \mathbb{Z}$ by $val_w(x_i) = \langle w_i(m-1), ...w_i(0)\rangle_{\mathbb{Z}}$. Thus, in the matrix whose columns are the letters of w, the i-th row represents the encoding of $val_w(x_i)$ with the most significant bit coming first. The language defined by the formula is $L(F) = \{w \in \Sigma^+ | val_w \text{ satisfies } F\}$.

2.2 Sequential Circuits

A combinational boolean circuit K is a pair (G, σ) where G is a finite directed acyclic graph and $\sigma : U \to \{AND, OR, NOT\}$ is a labeling function such that U is the set of vertices of G whose in-degree is greater than zero. We require that whenever $\sigma(x) = NOT$ then x has in-degree of one.

We call the vertices in U the *gates*. We denote the vertices of in-degree zero I and call them *inputs*; we denote the vertices of out-degree zero O, and call them *outputs*.

Given a boolean valuation $i : I \to \{true, false\}$, we define a valuation v on all vertices of G as follows:

$$v(x) = \begin{cases} i(x), & \text{if } x \in I \\ \neg v(\gamma(x)), & \text{if } x \in U \wedge \sigma(x) = NOT \\ \bigwedge_{y \in \Gamma(x)} v(y), & \text{if } x \in U \wedge \sigma(x) = AND \\ \bigvee_{y \in \Gamma(x)} v(y), & \text{if } x \in U \wedge \sigma(x) = OR \end{cases}$$

where $\gamma(x)$ denotes the single neighbor of x connected to it by an edge directed towards x and $\Gamma(x)$ denotes the set of all neighbors of x connected to it by edges directed towards x. We call the values of v on O the output values of K for input i.

The values of the outputs of a *combinational* boolean circuit, defined above, depend only on a single set of inputs and can be represented in a truth table. We next review (clocked) *sequential* circuits, which are equivalent to deterministic finite automata but compactly represent the set of states and the transition function.

A clocked sequential circuit (or SC, for short) is a tuple $(K, M, \mathsf{store}, \mathsf{load}, \mathsf{init})$ where

- K is a combinational boolean circuit with inputs and outputs I and O;
- M is a set of D-type flip-flops;
- $\mathsf{store} : M \to O$;
- $\mathsf{load} : M \to I$;
- $\mathsf{init} : M \to \{true, false\}$.

The load and store functions describe how the data input of each flip-flop is connected to a unique output of K and how the Q-output of each flip-flop is connected to a unique input of K. Such a backward-connected output-input pair will be denoted as a *state variable*. We call the inputs of K that are not in the image of load the *input variables* and call the outputs of K that are not in the image of store the *output variables*.

The SC works in clock pulses. It takes as input a stream that for every clock pulse contains values for all input variables, and produces as output a stream that for every clock pulse contains values of all the output variables, computed by K. In every clock pulse, K is provided with input values and it computes output values. The values for K's inputs corresponding to state variables are loaded from the flip-flops and the values for its inputs corresponding to input variables are provided by the input stream. Some of the values of K's outputs are stored in the flip-flops for the use in the next clock cycle, as determined by store.

The values stored in the flip-flops at the beginning of the first clock cycle are called initial values of the state variables and they are given by init.

Notice that a circuit with n input variables and m output variables can be viewed as a machine that, given a word from $(\{0,1\}^n)^+$, produces a word of the same length in $(\{0,1\}^m)^+$.

We can also use a SC to recognize a language.

Definition 1. *Let C be a SC with one output variable o and n input variables. We say that C accepts the word $w \in \{0,1\}^n$ if the value of o in the last cycle is 1 when the circuit is given w as input, one letter at each clock cycle.*

The language of C is $L(C) = \{w \in \{0,1\}^n | C \text{ accepts } w\}$.

Some of the standard finite state machine operations can be efficiently performed on the sequential circuit representations. Given a SC C with input variables $v_1, ...v_n$, state variables $q_1, ...q_n$ and output o, and a SC C' that uses the same input variables $v_1, ...v_n$ and has state variables $q'_1, ...q'_n$ and output o', we can construct a circuit $\neg C$ by simply appending a NOT gate at o and making the output of the NOT gate the output of $\neg C$. Similarly, we can construct circuits $C \wedge C'$ and $C \vee C'$ by connecting the outputs of C and C' to an appropriate logical gate, whose output will become the output of the composite circuit. It can easily be seen that 1) $L(\neg C) = (\{0,1\}^n)^+ \backslash L(C)$; 2) $L(C \wedge C') = L(C) \cap L(C')$; 3) $L(C \vee C') = L(C) \cup L(C')$.

3 Translations Between QFPAbit and Sequential Circuits

This section establishes correspondence between QFPAbit and sequential circuits by providing translations in both directions that maintain a close correspondence between the accepted languages.

3.1 Reduction from QFPAbit to Sequential Circuits

Since we have already shown how to construct boolean combinations of sequential acceptor circuits, it is enough to find a set of basic QFPAbit formulae out of which all QFPAbit formulae can be built using logical connectives, and then show how these basic formulae can be translated to SCs.

Definition 2. *Let $w \in \Sigma^+$ with $\Sigma = \{0,1\}^n$ as usually. Suppose*

$$w = \begin{pmatrix} w_1(0) \\ \vdots \\ w_n(0) \end{pmatrix} \begin{pmatrix} w_1(1) \\ \vdots \\ w_n(1) \end{pmatrix} \cdots \begin{pmatrix} w_1(m) \\ \vdots \\ w_n(m) \end{pmatrix}$$

Let $S \subseteq \{1, ...n\}$ be non-empty. We define the projection of w onto the coordinates S to be the string $w^S = w^S(0)...w^S(m)$, where $w^S(i)$ is the column vector $(w_j(i))_{j \in S} \in \{0,1\}^{|S|}$. For a language $L \subset \Sigma^+$, we define the projection of L onto the coordinates S to be the language $L^S = \{w^S | w \in L\}$. Note that L^S is a language over the alphabet $\{0,1\}^{|S|}$.

Every QFPAbit formula is a boolean combination of atomic formulae of the form $T_1 \% T_2$ where T_1 and T_2 are terms and $\% \in \{=, \neq, <, \leq, >, \geq\}$. We will now show how to transform any formula F into a new one where the atoms will be of a more restricted form. The new formula will have more variables than F, but when projected onto the variables occurring in F their languages will be the same. We apply the following sequence of transformations:

1. Replace all atomic relations by equalities and strict "less-than" inequalities using the fact that $T_1 < T_2$ if and only if $T_1 + (-1)T_2 < 0$.
2. Remove all instances of multiplication by constants other than -1 and powers of two by exploiting the fact that any term of the form cT is equal to a sum of terms of the form $2^k T$ corresponding to c's two's complement encoding.
3. Remove all instances of multiplication by -1 by replacing every sub-term of the form $(-1)T$ by $\bar{}T + 1$. This equivalence follows easily from the definition of the two's complement encoding.
4. Move all additions to separate conjuncts on the highest level of the formula by replacing every occurence of $T_1 + T_2$ by a fresh variable s and adding conjuncts $s = x + y$, $x = T_1$ and $y = T_2$ to the formula, where x and y are also fresh variables.
5. Move all multiplications by a constant 2^k, which are the only multiplications now left in the formula, to conjuncts on the highest level of the formula by replacing every occurence of $2^k T$ by a fresh variable x and adding $x = 2^k y$ and $y = T$ as conjuncts to the formula, where y is another fresh variable.
6. Replace every additive occurrence of an integer constant c inside a larger term by a fresh variable y_c and add a conjunct $y_c = c$ to the formula.

Let us call the formula that we obtain G. It has size that is polynomial in the size of F and and it consists only of atoms of the following five forms: (i) $T < 0$; (ii) $T_1 = T_2$; (iii) $y = c$; (iv) $x = 2^k t$; (v) $s = x + y$, where x, y, s and t are variables, c is an integer constant and T, T_1, T_2 are terms that contain exclusively variables and bit-vector logical operators.

It is easy to construct SCs for atoms of each of these four forms. For details of these constructions along with circuit diagrams, see our technical report [10]. The general flavor of these circuits is that they compare streams of binary digits. The most complicated case is (iv), where the circuit compares a binary stream to a version of itself

shifted by a constant number of bits. Each of the sub-circuits for cases (i),(ii) and (v) has only a constant number of state variables. In case (iii), the number of state variables is proportional to the logarithm of the constant c and in case (iv) it is proportional to k.

Finally, we compose the partial specification circuits by boolean operations to find a SC for G. The correctness of this synthesis procedure is expressed in the following theorem.

Theorem 1. *Let C_F be the circuit obtained from a QFPAbit formula F using the above synthesis procedure. Let V be the set of variables occuring in F. Then*

$$L(C_F)^V = L(F).$$

Moreover, both the the number of gates of C_F and the running time of the synthesis procedure are polynomial in the number of symbols of F. The number of input variables of C is the same as that of F and the number of C's state variables is proportional to the number of symbols of F.

3.2 Reduction from Sequential Circuits to QFPAbit

Let C be a sequential circuit with an underlying combinational circuit $K = (G, \sigma)$, n input variables $\{v_1, ...v_n\}$, m state variables $\{q_1, ...q_m\}$ and output variables $\{o_1, ...o_l\}$. Let $I : \{q_1, ...q_m\} \to \{0, 1\}$ be the initial assignment of values to the state variables. Let U be the set of all gates of K other than those corresponding to the output variables and state variables of C. We will pretend that the elements of U can be used as identifiers for QFPAbit variables and construct a QFPAbit formula with variables $\{v_1, ...v_n, q_1, ...q_m, o_1, ...o_l\} \cup U$, such that for every satisfying assignment, the two's complement encodings of the values of the variables describes the evolution of the values of the corresponding variables and gates in a run of C. Although the QFPAbit variables have the same names as the variables and gates of the circuit, it should be clear from the context which ones do we mean.

We will refer to the values of the gates and inputs of the automaton in the k-th clock cycle by $q_1(k), ...q_m(k)$,$v_1(k), ...v_n(k)$, $o_1(k)...o_l(k)$ and $x(k)$ for all $x \in U$. In the cycle when the inputs are $q_1(i), ...q_m(i), v_1(i), ...v_n(i)$, the values of all the gates in U will be $x(k)$, the output variables will be $o_1(i), ...o_l(i)$ and the outputs corresponding to state variables at that cycle will be denoted $q_1(i+1), ...q_m(i+1)$, because they serve as inputs for the next cycle. We start the numbering of clock cycles from 0.

We will be abusing notation slightly by writing $\sigma(v)(x_1, ..., x_k)$ for some gate v and boolean values $x_1, ..., x_k$ to mean the application of the boolean function represented by $\sigma(v)$ to $x_1, ..., x_k$. Then for all $j \in \{1, ..., m\}$, $k \in \{1, ..., l\}$, $x \in U$ and all $i \in \{0, ..., N-1\}$ where N is the length of the input word, the run of C on that input word is characterized by the following four equations:

$$q_j(0) = I(q_j) \tag{1}$$

$$x(i) = \sigma(x)(\Gamma(x)(i)) \tag{2}$$

$$o_k(i) = \sigma(o_k)(\Gamma(o_k)(i)) \tag{3}$$

$$q_j(i+1) = \sigma(q_j)(\Gamma(q_j)(i)) \tag{4}$$

where, just like in our definition of a combinational circuit, $\Gamma(v)$ denotes the part of the neighborhood of a gate v connected to it with incoming edges, and $\Gamma(v)(i)$ denotes a vector of values of these nodes in clock cycle i.

We next build a QFPAbit formula for which every satisfying evaluation is such that **the reverse** of the bit-sequences of the values it assigns to the variables conform to the above conditions. Since C treats all numbers as starting with the most significant bit, in our QFPAbit representation this will be reversed and hence $x(0)$, $q_j(0)$ and $o_k(0)$ will refer to the least significant bits of the encoding of the values of the variables.

For any gate v of K, let $\bar{\sigma}_v$ be the formula obtained by applying the bit-vector logical operator corresponding to $\sigma(v)$ to the variables in $\Gamma(v)$. Then the following formula can be used to describe the evolution of the digits of q_j:

$$q_j = 2\sigma_{qj} \bar{\vee} I(q_j)$$

The justification is as follows. Taking the bitwise disjunction of a number with 1 or 0 preserves all the digits except the least significant one, which is set to 1 or 0 respectively. Multiplication by 2 induces a shift to the left of the two's complement encoding of a number. Hence the above formula establishes that every bit of q_j is equal to the next bit of σ_{qj} except for the first (least significant) one, which is equal to $I(q_j)$. This ensures that equations (1) and (4) are satisfied.

Similarly, the formulas $o_j = \bar{\sigma}_{oj}$ and $x = \bar{\sigma}_x$ assert that the reverse binary encodings of o_j and x, for some $x \in U$, correspond to their values in the run of C on the given input as described by equations (3) and (2).

Since the most significant digit in a two's complement encoding can be replicated without changing the value of the represented number, QFPAbit formulas have the property that the last letter of a word in a formula's language can be repeated arbitrarily many times to obtain another word inside the language. In the underlying circuit, this would translate to a "blindness" towards the repetition of the initial input letter, which is a property that not all circuits have. In general we cannot find a formula whose language contains exactly those words whose reverse encodes a run of the circuit.

The way to treat this problem is to construct a formula that contains a clause saying "the variables are only simulating the circuit for a finite number of steps and then are allowed to deviate". That way we obtain a formula for which to every possible satisfying evaluation corresponds a word describing the run of the circuit. However, each such valuation will also represent an infinite number of longer incorrect descriptions of a run of the circuit.

For succintness, let $\Delta_{qj} \equiv 2\bar{\sigma}_{qj} \bar{\vee} I(q_j)$. Let y be a fresh variable and consider the formula

$$F_C \equiv 1 + ((y-1) \bar{\vee} y) = 2y \wedge y > 1 \wedge \left[\bigwedge_{j=1}^{m} (q_j \bar{\wedge} (y-1)) = (\Delta_{qj} \bar{\wedge} (y-1))\right]$$
$$\wedge \left[\bigwedge_{j=1}^{l} (o_j \bar{\wedge} (y-1)) = (\bar{\sigma}_{oj} \bar{\wedge} (y-1))\right] \wedge \left[\bigwedge_{x \in U} (x \bar{\wedge} (y-1)) = (\bar{\sigma}_x \bar{\wedge} (y-1))\right].$$

The subformula $1 + ((y-1) \bar{\vee} y) = 2y \wedge y > 1$ asserts that y is a power of two, say $y = 2^k$, and that k is at least 1. Therefore the two's complement encoding of $(y-1)$ is $\langle 0, ..., 0, 1, ..., 1 \rangle_{\mathbb{Z}}$ with an arbitrary number of zeros and exactly k ones. So the clauses

of the form $(T_1 \bar{\wedge}(y-1)) = (T_2 \bar{\wedge}(y-1))$ assert that the k least significant digits of T_1 and T_2 are the same. The rest of the digits can be arbitrary.

Theorem 2. *For any given satisfying valuation of F_C, $y = 2^k$ for some k. The first k bits, presented in the reverse order, of the bit-sequences corresponding to two's complement encodings of $q_1, ...q_m$ and $o_1...o_l$ describe the evolution of the values of those variables throughout the first k clock cycles of the run of C on the input word given by the reverse two's complement encodings of $v_1, ...v_n$, as specified by the equations (1)-(4).*

Now we show how this translates to acceptor circuits defining a language:

Theorem 3. *Suppose C is a SC with one output o and let $F'_C \equiv F_C \wedge o < 0$. Then $L(F'_C) \neq \emptyset$ if and only if $L(C) \neq \emptyset$*

Proof. *The clause $o < 0$ is true if and only if the first digit of o is one. It follows from the above discussion of F_C that for every word w of length k providing encoding of values for $v_1, ..., v_n$, there exist infinitely many satisfying evaluations for F_C under which $y = 2^k$ and the reverse encoding of the values of the variables describes the run of C on the reverse of w. Now suppose that C accepts w. This happens if and only if in the last, k-th, clock cycle the value of the output bit is one. But this is if and only if the first digit of the value of o in F'_C is one. Therefore the described evaluations satisfy F'_C if and only if C accepts w. This means that the language of C is non-empty if and only if F'_C is satisfiable.* ■

To summarize, we have described polynomial-size translations between QFPAbit and sequential circuits going both ways. For every QFPAbit formula we can construct a sequential circuit recognizing the same language. For every sequential circuit we can construct a QFPAbit formula that contains variables representing inputs, outputs and state variables of the circuit, and it is satisfied only by valuations that assign these variables values whose binary encoding in reverse describes an initial portion of the evolution of the circuit's variables during a run. If the circuit has only one output then it is an acceptor circuit and in this case we can construct a QFPAbit formula which is satisfiable if and only if the language of the SC is non-empty. Moreover, the formula will accept a language such that for every word w in this language, an initial part of w projected onto the input variables and reversed is a word in the language of the SC.

4 From Specification Circuits to Transducer Circuits

Given a specification written as a QFPAbit formula, we have shown how to build a specification circuit of a size linear in the size of the formula. Provided that the variables of the formula, and thus the inputs of the automaton are partitioned into two groups, $\bar{i}$ and $\bar{o}$, interpreted as the inputs and the outputs of the synthesized function, we will now show how to construct a set of circuits that will work as a transducer, i.e. given a word from the "$\bar{i}$-projection" of the language, produce an output word from the "$\bar{o}$-projection" of the language such that together they satisfy the specification, if such an output word exists. The structure of our algorithm is similar to the one presented in [4]. Our use of the word "transducer" does not refer to the traditional notion of Finite State

Transducers, but to a more complicated machine with the following main features. Our transducer reads the whole input twice. The first time from the beginning to the end to generate the exhaustive run of the projection of the specification circuit onto the input variables, and the second time backwards, determining concrete states and output letters within the exhaustive run. In the meantime it uses an amount of memory proportional to the length of the input. This allows us to express functions for which it is not possible to determine the output before reading the entire input, which is needed to obtain complete synthesis for QFPAbit.

In contrast to [4], we will be using sequential circuits instead of automata. This more concrete implementation allows us to perform an optimization that will ensure that the presence of large integer constants in the formula does not necessarily cause a blow-up in the size of the transducer proportional to the value of that constant, as was the case with the previous approach. Moreover, even if a state-space expansion does occur, the size of our circuits is guaranteed to be singly-exponential in the size of the specification formula. No such bound on the size of the automata was provided in [4].

In Section 4.2 we study two more optimization techniques - how to exploit the circumstance when the specification formula is either a conjunction or a disjunction of sub-formulas to build the transducer as a composition of smaller transducers.

Definition 3. *Given a (non-)deterministic automaton $A = (\Sigma_V, Q, init, F, T)$ over variables V and a set $I \subset V$, the projection of A to I, denoted by A^I, is the non-deterministic automaton $(\Sigma_I, Q, init, F, T_I)$ with $T_I = \{(q, \sigma_I, q') \in Q \times \Sigma_I \times Q | \exists \sigma \in \Sigma_V . (q, \sigma, q') \in T \wedge \sigma^I = \sigma_I\}$.*

Since it is natural to view a sequential circuit as a DFA, we also allow ourselves to talk about projections of sequential circuits.

Definition 4. *The exhaustive run ρ of an automaton $A = (\Sigma, Q, init, F, T)$ on a word $w \in \Sigma^*$ is a sequence of sets of states $S_1, ...S_{|w|+1}$ such that (i) $S_1 = init$ and (ii) for all $1 \leq |w|, S_{i+1} = \{q' \in Q | \exists q \in S_i . (q, w_i, q') \in T\}$.*

Suppose the specification circuit is a sequential circuit C with input variables $\bar{i} \cup \bar{o}$, state variables $\bar{q}$ and one output variable determining the acceptance. Here by each of $\bar{i}, \bar{o}$ and $\bar{q}$ we actually mean vectors of variables wide n, l and m bits respectively. We will also be using $\bar{i}, \bar{o}$ and $\bar{q}$ to denote the sets of individual variables comprising each of the vectors.

We now partition the state variables as follows. We let $\bar{s}$ be the largest set of state variables such that the value of each of them in the $(N+1)$-st clock cycle depends only on the values of $\bar{i}$ and the state variables inside $\bar{s}$ in the N-th clock cycle. In particular, they do not depend on the values of $\bar{o}$. We denote all the other state variables as $\bar{r}$ and we will assume that $\bar{r}$ is a vector of width m_1 and $\bar{s}$ is of width m_2.

The set $\bar{s}$ can be determined by exploring the graph of dependencies amongst the variables of $\bar{q}$ and $\bar{o}$. We can determine whether a formula $\varphi(x)$, for example one defining the value of a q_j in the next clock cycle, depends on a variable x, which it contains, by using a SAT-solver to check whether the formula $\varphi(true) \leftrightarrow \varphi(false)$ is valid.

We will now describe the operation of our transducer, which consists of three circuits that we call C', ϕ and τ. Circuit ϕ is a combinational circuit and the other two

are sequential. Their roles are analogous to those of the deterministic automaton A' and functions ϕ and τ in [4]. Our specification circuit C fulfills the responsibility of the specification automaton A used in [4].

C' performs two tasks. First, it runs the part of C that computes the sequence of values of $\bar{s}$ as C consumes $\bar{i}$. In parallel with this, C' also simulates the exhaustive run of the projection of C onto the input variables $\bar{i}$. So running C' with the sequence of values for $\bar{i}$ as input will generate a sequence of values for $\bar{s}$ together with a sequence of sets of possible values for the rest of the state variables, which are $\bar{r}$. We will store this trace in a memory from which it can later be read in the reversed order.

This separation of sets $\bar{s}$ and $\bar{r}$ is one of the main improvements in our approach over previous work. It takes advantage of the simple idea that when projecting a deterministic automaton onto a subset of its input variables, it is possible that the transitions within a subset of the states of the automaton remain deterministic even with the restricted alphabet, and hence that part of the automaton does not need to go through an exponential expansion due to the projection. This optimization applies in particular in the case when the specification formula contains division of a term that is completely determined by $\bar{i}$-variables by a power of 2. An intuitive explanation is the following. The specification circuit for the formula $x = 2^k t$ verifies whether the encoding of x is a copy of the encoding of t shifted to the left by k bits. Therefore it needs k state variables to remember the past k bits of x. The values of these k state variables are independent of t and hence if x is an $\bar{i}$-variable, which means that we are performing division, then these k state variables will belong to $\bar{s}$ and they will not participate in the state-space explosion of C'. On the other hand, this optimization does not apply if x is an $\bar{o}$-variable, i.e. when we are performing multiplication.

The purpose of ϕ is to find inside the last stored set of possible states for $\bar{r}$ one which is, combined with the last stored value of $\bar{s}$, an accepting state of C.

Eventually, we run τ, which reconstructs a whole accepting run of C by tracing backwards through the stored exhaustive run of its projection onto the input variable set $\bar{i}$, using the accepting state determined by ϕ as a starting point. During this backward run it constructs a sequence of $\bar{o}$ letters that is the final output of the transducer.

4.1 Implementation of C', ϕ and τ as Circuits

For C', consider the circuit in the figure in Appendix A, which has state variables $R_1, ...R_{2^{m_1}}$ and $\bar{s}$, and no outputs.

Let C_1 and C_2 denote the sub-circuits of C for computing $\bar{r}$ and $\bar{s}$ respectively. In the figure, we denote the corresponding combinational circuits behind these SCs by K_1 and K_2. We let $C_{\bar{i}}$ be the projection automaton obtained from C_1 by projecting it onto the $\bar{i}$-variables. The intended meaning of the state variables $R_1, ...R_{2^{m_1}}$ of C' is that R_k is set to true if and only if at that point the non-deterministic automaton $C_{\bar{i}}$ could be in the state number k. Since there are exactly 2^{m_1} possible states of $C_{\bar{i}}$, we can make some arbitrary assignment of the possible states of $C_{\bar{i}}$ to the R_k's. Initially, A' is in a state where all variables R_k are 0 except for one, corresponding to the initial state of $C_{\bar{i}}$. The initial value of $\bar{s}$ is also determined by the given initial state of C.

The $\bar{r}_i$ and $\bar{o}_j$ denoted in italics represent constant bit-vectors given as input to each of the 2^{ml} copies of C_1. The indexes are assigned so that $\bar{r}_j$ is the assignment of state

variables of $C_{\bar{i}}$ corresponding to the state which is represented by R_j. Hence each of the C_2-subcircuits produces an outcome $\bar{r}$-state for a given combination of a previous state and values for the $\bar{o}$-variables.

Each of the AND-like-gates with an R_k inscription is understood to have negations at an appropriate combination of its inputs, so that it returns true if and only if its input $\bar{r}$ represents the $\bar{r}$-state corresponding to R_k and also the incoming signal from the state variable R_j is true. This last condition has the effect of considering the output only of those sub-circuits for which the input state $\bar{r}_j$ is actually one of the possible states in the exhausting run of $C_{\bar{i}}$ at the moment.

The last layer of ordinary OR-gates just has the effect that if any of the possible combinations of an active previous state and an $\bar{o}$-letter produces the state corresponding to R_k then R_k is set to one in the next cycle. The main idea of this circuit is that for every state of C that is possible at the present clock cycle, it tries every possible $\bar{o}$-letter to produce the set of all possible states in the next clock cycle.

Now assume that the sequence of states this circuit goes through while reading an input word is saved in a memory from where it can readily be read in the reverse order. Recall that ϕ is supposed to find an accepting state of C amongst the possible states encoded in the last state of C' - that is, in the combination of the "exhaustive state" of $C_{\bar{i}}$ encoded by $R_1, ...R_{2^{m_1}}$ and the deterministic part of the state, $\bar{s}$. A slight divergence between deterministic automata used in [4] and our variant of sequential circuits is that whether the circuit accepts depends not only on the current value of its state variables but also on the value of all its inputs - the circuit accepts simply when it outputs a 1. To account for this, our ϕ circuit has to choose both a state from amongst the states possible in the penultimate clock cycle of the run of C', and a suitable $\bar{o}$-letter, such that the resulting state is accepting. If such state and $\bar{o}$-letter do not exist, the user is notified that for the given sequence of values for the $\bar{i}$-variables there exists no satisfying sequence of values for the $\bar{o}$-variables. The implementation of ϕ is a circuit very similar to that for C', also containing 2^{m_1+l} copies of K_1. However, since it only needs to be run for one clock cycle, it is a combinational circuit rather than a sequential one.

Finally, we use a very similar circuit for the function τ. In each clock cycle, it takes as input a transition $\langle S', \bar{i}, S\rangle$ of C' and a state $\bar{q} \in S$ and generates a state $\bar{q}' \in S'$ and an output symbol $\bar{o}$ such that there is a valid transition in C from $\bar{q}$ to $\bar{q}'$ while reading the letter obtained by combining $\bar{i}$ with $\bar{o}$. This is again implemented by guessing combinations of an appropriate $\bar{o}$-letter and $\bar{r}$-state, so τ consists of 2^{m_1+l} copies of K_1 and some servicing circuitry. The output of τ and also the final output of the transducer is the sequence of $\bar{o}$ letters. Notice that it comes in the reverse order, respective to $\bar{i}$.

4.2 Constructing Transducer as a Composition of Transducers for Sub-formulas

Suppose that the specification formula F on its highest level is a disjunction of sub-formulas $\varphi_1, ...\varphi_k$. Then we can build a transducer for each of them separately and run them in parallel. If for a given input any of the transducers finds an output satisfying the sub-formula corresponding to that transducer, say φ_i, then this output can be taken

to be the global output. If φ_i mentions only a subset of the output variables then values for the remaining ones can be picked arbitrarily.

If the specification formula, on the other hand, is a conjunction of sub-formulas, then we also have to mind dependencies between the variables.

Definition 5. *We say that a QFPAbit formula ψ over variables V uniquely determines a set of variables $\bar{x}$ as a function of a set of variables $\bar{y}$, if for any partial valuation $val_{\bar{y}} : \bar{y} \to \mathbb{Z}$ that only assigns values to the variables of $\bar{y}$, the set of satisfying valuations of ψ that extend val is non-empty and all of them give all the variables in $\bar{o}$ the same values.*

If the specification formula F is a conjunction of sub-formulas $\varphi_1, ...\varphi_k$, we can apply the following reasoning. Suppose that there exists $\bar{o}' \subseteq \bar{o}$ such that some φ_j uniquely determines the values of $\bar{o}'$ as a function of $\bar{i}$. Now suppose that $val : \bar{i} \cup \bar{o} \to \mathbb{Z}$ is a satisfying valuation for F. Then, in particular, it is a satisfiing valuation for φ_j and it assigns $\bar{o}'$ the same values as *any* satisfying valuation of φ_j that gives the $\bar{i}$'s the same values as val.

This means that we can build an independent transducer for φ_j and use its output to fix the values of $\bar{o}'$ in F, allowing us to build a smaller transducer for the rest of the variables. Notice that the values that the transducer for φ_j computes for those variables that have not been proven to be uniquely determined by $\bar{i}$ must be ignored, because their values need not be a part of a satisfying valuation for the rest of F.

In practice, we can use this fact to construct a sequence of transducers with increasing number of $\bar{i}$ variables and decreasing number of $\bar{o}$ variables. We scan through the list of conjuncts of F and whenever we find one, say φ_j, in which some subset of $\bar{o}$ variables is uniquely determined by the $\bar{i}$ variables, we build a transducer for it, reclassify the uniquely determined $\bar{o}$-variables to $\bar{i}$-variables in F and repeat the process, wiring the appropriate outputs of the transducer for φ_j to become the inputs of the next transducer. If it turns out that in a particular conjunct, all the occurring $\bar{o}$-variables are uniquely determined, this whole conjunct can be removed from F.

Notice that for regularly occuring conjuncts of a standard form, like for example equality assertions involving standard arithmetical operations, we will not have to invoke the general transducer-synthesis method described at the beginning of this section. Instead, we can use potentially more efficient pre-computed circuits loaded from a library. This can, for example, be applied in the case when the conjunct asserts that an $\bar{o}$-variable is a constant multiple of a term that is uniquely determined by the $\bar{i}$ variables.

The length of the resulting sequence of transducers is at most quadratic in the number of $\bar{o}$ variables, which can be seen by inspecting the running time of the trivial algorithm that loops through the cojuncts in an arbitrary fixed order and halts when during an iteration examining all the conjuncts it can not reclassify any new $\bar{o}$-variables to $\bar{i}$-variables.

Obviously, this optimization is useful only if the specification formula F is in fact a conjunction containing conjuncts that do have the property of uniquely determining some of the $\bar{o}$-variables as a function of the $\bar{i}$ variables. As discussed in Section 3.1, before building the specification circuit we first pre-process the input formula, so that the formula that is eventually used for building the circuit is

$$G \equiv F' \wedge \varphi_1 \wedge ... \wedge \varphi_n$$

where each of the φ_i has one of the following forms: (i) $x = 2^k t$; (ii) $x = c$; (iii) $s = x + y$; (iv) $T_1 = T_2$, where x, y, t, s are variables, c is an integer constant and T_1, T_2 are terms built out of variables and bit-vector logical operations. F' is a boolean combination of atoms of similar forms, but at the present time we do not have methods for investigating variable dependencies in non-atomic formulas.

On the other hand, for each of the φ_j's we can exactly determine which $\bar{o}$-variables are uniquely determined by the $\bar{i}$-variables. In case (i), if at least one of the variables present is an $\bar{i}$-variable then the other is determied. In case (i), variable x is determined, and in case (iii), if at least two of the variables are $\bar{i}$-variables then the last one is determined. In case (iv), since T_1 and T_2 contain only variables and bit-vector logical operations, the equality holds exactly if the propositional formulas corresponding to T_1 and T_2 evaluate to the same boolean value in every clock cycle. Therefore it is enough to investigate which $\bar{o}$ variables are uniquely determined by the $\bar{i}$ variables in the propositional formula $\hat{T}_1 \leftrightarrow \hat{T}_2$, where $\hat{T}_1$ and $\hat{T}_2$ are propositional formulas obtained from T_1 and T_2 by replacing the bit-vector logical operators by standard boolean operators and treating the QFPAbit variables as propositional variables.

Example. We demonstrate the usefulness of this optimization technique on an example. Let us forget for a moment that our language contains an out-of-the-box plus operator and suppose we would like to synthesize a function for performing addition and outputting the sequence of carry bits at the same time. It can be specified in QFPAbit as follows.

$$(s = x \bar{\oplus} y \bar{\oplus} c) \wedge (c = 2((x \bar{\wedge} y) \bar{\vee} (x \bar{\wedge} c) \bar{\vee} (y \bar{\wedge} c)))$$

where x and y are designated as inputs and s and c are outputs representing the sum and the sequence of carry bits respectively. Clearly, the right-hand conjunct determines c uniquely, given values for x and y. Our prototype implementation is able to detect this and builds a transducer which is a composition of two parts - one for the right-hand conjunct, which computes the value of c given values for x and y, and one for the left-hand conjunct that computes the value of s given values for x, y and c. Due to this factorisation, the total number of gates in all the circuits involved is $7.2\times$ smaller than when we enforce the building of a single monolithic transducer for the whole formula.

To conclude the discussion of this optimization technique, let us look closer at how it applies to those φ_j's that are of form $x = 2^k t$. Because of the way how these conjuncts originate during the pre-processing of the specification formula, often both x and t are output variables. If after inspecting some other conjuncts we manage to specify one of them as an input variable, the other is immediately determined by it and we will be able to remove this conjunct from the formula and construct an efficient transducer for it. We can summarize this in the following lemma.

Lemma 1. *Suppose that the original formula, before pre-processing, contains multiplication by a constant c in a context of the form $T_1[cT] = T_2$ such that either all the $\bar{o}$-variables occuring in T are uniquely determined by the $\bar{i}$-variables, or the $\bar{o}$-variables of T occur nowhere else in T_1 and T_2 and the value of a fresh variable x is uniquely determined in the formula $T_1[x] = T_2$. Then the total size of all the circuits of the transducer obtained by the procedure described in this section will be proportional to the logarithm of c.*

5 Conclusion

We have presented a synthesis procedure that starts from QFPAbit description of an input/output relation, generates a sequential circuit of a polynomial size, and then transforms this circuit into a synthesized system of sequential circuits that maps a sequence of inputs into a sequence of outputs.

The described synthesis procedure improves the previous work by two independent optimizations. We have built a prototype implementation that allowed us to show on examples that these techniques work and are important.

Acknowledgements The idea of replacing synthesis from WS1S with synthesis from QFPAbit as well as a polynomial translation from QFPAbit into circuits originated in a discussion between Barbara Jobstmann and Viktor Kuncak in October 2010. Barbara Jobstmann was also suggesting decomposing specifications and performing synthesis modularly. We thank Aarti Gupta for pointing to the related work in her PhD thesis [3] as well as Sharad Malik and Paolo Ienne for useful discussions.

References

1. R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.
2. J. Buchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138(295-311):5, 1969.
3. A. Gupta. *Inductive Boolean Function Manipulation: A Hardware Verification Methodology for Automatic Induction*. PhD thesis, CMU, 1994.
4. J. Hamza, B. Jobstmann, and V. Kuncak. Synthesis for regular specifications over unbounded domains. In *Formal Methods in Computer-Aided Design (FMCAD), 2010*, pages 101–109. IEEE, 2010.
5. B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *FMCAD*, 2006.
6. N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. In *Proc. 5th Int. Conf. Implementation and Application of Automata*. LNCS, 2000.
7. V. Kuncak, M. Mayer, R. Piskac, and P. Suter. Complete functional synthesis. In *ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI)*, 2010.
8. M. Rabin. *Automata on infinite objects and Church's problem*. Number 13 in Regional Conference Series in Mathematics. American Mathematical Society, 1972.
9. T. Schuele and K. Schneider. Verification of data paths using unbounded integers: Automata strike back. *Hardware and Software, Verification and Testing*, pages 65–80, 2007.
10. A. Spielmann and V. Kuncak. On synthesis for unbounded bit-vector arithmetic. Technical Report EPFL-REPORT-174801, EPFL, 2012.
11. S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In *POPL*, 2010.
12. L. Stockmeyer and A. R. Meyer. Cosmological lower bound on the circuit complexity of a small problem in logic. *J. ACM*, 49(6):753–784, 2002.

A Schema of Circuit C'