On Synthesis for Unbounded Bit-vector Arithmetic

EPFL Technical Report, February 2012

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. Given a QFPAbit formula with input and output variables, we describe an algorithm that generates an efficient linear-space function from a sequence of inputs to a sequence of outputs, without the causality restriction of reactive synthesis. The starting point for our method is a polynomial-time translation of QFPAbit into sequential circuits that check the correctness of the input/output relation. From such circuits, 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

Automatically synthesizing systems from specifications is arguably an easy way to obtain implementations, which are moreover correct by construction. The main topic of this paper is the automated synthesis of a function over unbounded data domains, whose specification is given as a relation between inputs and outputs. We investigated a hardware-oriented approach, where the function is implemented as a chain of sequential circuits. Our study builds on work in the area of synthesizing functions as deterministic finite automata (DFA), which are a natural abstraction for sequential circuits. This branch of research was founded by [5], but automata-based approaches have earlier been essential in reactive synthesis [1,11,7]. We have also built a prototype implementation of our synthesis algorithm, which we applied to examples discussed in the paper.

The aim is to synthesize a function from tuples of input values to tuples of output values. The specification is usually a logical formula stating a relation between the inputs and outputs, or a finite automaton whose language is the set of pairs of corresponding input-output tuples. In our case, the specification is a logical formula in the language of quantifier free Presburger Arithmetic extended with bit-vector logical operators (QFPAbit), a language that was introduced in [12] and whose expressive power is equivalent to that of regular expressions.

As the first step in our synthesis algorithm, we translate the specification formula into a specification circuit. A translation of QFPAbit formulas to alternating finite automata [2,3], was also presented by Schuele in [12]. Alternating finite automata (AFA)

can be seen as another abstraction for sequential circuits, in a sense complementary to DFA. In contrast with a representation as a DFA, the natural representation of a sequential circuit as an AFA does not require an exponential increase in size, but requires that the input sequence is presented in the reversed order. Our translation of QFPAbit to sequential circuits is different from that presented in [12] and optimized for easy implementation. Since the decision problem of emptiness of the language of an AFA was shown to be in PSPACE [6], Schuele's translation from QFPAbit to AFA proves that Satisfiability of QFPAbit is in this complexity class. To demonstrate that QFPAbit is a suitable representation for sequential circuits, we also show how to translate an arbitrary sequential circuit back to a QFPAbit formula (of polynomial size) defining the same language. From this translation follows as our secondary result the fact that QFPAbit is also PSPACE-hard, because of the PSPACE-hardness of AFA-Emptiness. To our knowledge, this is a new discovery.

Once the specification circuit is built, the basic idea of our synthesis algorithm closely resembles the method described in [5]. Given the specification automaton, or in our case a circuit, we simulate the exhaustive run of its projection onto the input variables. The result of this simulation is a sequence of sets of possible states in which the specification circuit could be while reading the given sequence of values for the input variables and *any* sequence of values for the output variables. This information can then be used to trace back through the exhaustive run to find a concrete sequence of states and output letters for the specification circuit ending in an accepting state.

Our main contribution are two optimizations concerning the size of the circuits produced. In a naive implementation, the size of the circuit simulating the exhaustive run would be exponential both in the number of states of the specification circuit, because its state may be any subset of the states of the specification circuit, and in the number of output variables, because for every input letter we have to consider every possible output letter.

Our first optimization is the observation that there may exist a subset of states of the specification automaton such that transitions within this subset are in fact dependend only on the inputs designated as inputs of the function. This part of the specification circuit therefore does not have to undergo the exponential expansion mentioned before. Since the authors of [5] rely on MONA [8] to build their specification circuits, this optimization was not accessible to them because of the explicit enumeration of states that MONA uses. They point out as an exemplary problem that the occurrence of constants in the specification formula leads to a specification automaton with a number of states proportional to the value of the constant and subsequently to an exponential number of states in the exhaustive-run automaton. Due to our optimization, we can avoid this exponential expansion for specifications containing constants in certain contexts, made specific in Section 4.

For example, the exhaustive-run circuit synthesized by our prototype implementation for the specification $in = 2 \times out$, encoding division by 2, needs 7 bits of memory to store its state. The corresponding circuits for specifications $in = 1024 \times out$ and $in = 1048576 \times out$ have only 19 and 30 state-bits respectively, which excludes a proportional dependence on the size of the constant.

The second optimization is finding functional dependencies between the input variables and subsets of the output variables in the specification formula, which allows us to express the resulting function as a composition of several functions with smaller output tuples. In practice this means constructing a longer sequence of smaller circuits. This serves to alleviate the effect of the exponential expansion in the number of output variables for a favorable kind of specification formulas. We provide demonstrative examples in Section 4

To summarize, the contributions of our work are the two optimizations of the automata/circuit-aproach to functional synthesis, and establishing the PSPACE-completeness of the Satisfiability problem of QFPAbit.

The next section contains our definitions of QFPAbit and sequential circuits, and some preliminary results about them. Section 3 describes the reduction constructions from QFPAbit to sequential circuits and vice-versa, proving PSPACE-hardness of QFPAbit-Sat. Section 4 contains detailed description of the main part of our synthesis algotithm along with the optimizations. Section 5 is a summarizing conclusion.

2 Preliminaries

All the languages mentioned from now on will be assumed to be over the alphabet $\Sigma = \{0,1\}^n$ unless otherwise stated. We consider only non-empty words.

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

Presburger Arithmetic with Bit-vector Logical Operators is the first order theory of the integers with addition and bit-vector logical operations acting on the binary two's complement representation of the numbers.

Definition 1. *Let V be a finite set of variables. With $c \in \mathbb{Z}$ and $x \in V$, the set of terms of QFPAbit is defined by the following grammar.*

$$T := c \mid x \mid T + T \mid cT \mid \bar{\neg}T \mid T \bar{\wedge} T \mid T \bar{\vee} T$$

The set of formulae of QFPAbit is defined by the following grammar.

$$F := T \% T \mid \neg F \mid F \wedge F \mid F \vee F \mid F \rightarrow F \mid F \leftrightarrow F$$

where $\%$ ranges over the symbols $=, \neq, <, \leq, >, \geq$.

The assumed interpretation of this language is the set of integers $\mathbb{Z}$ with its usual equality and inequality relations and the usual addition operation. The bit-vector logical operators are acting on the two's complement encoding of numbers, which is defined as follows.

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

An important property of this encoding is that replicating the most significant bit does not change the value. Therefore the representation of a number is not unique. This justifies our definition of the bit-vector operators because for any two numbers we can

always find encodings that have the same length. In the later text, whenever we refer to *the* two's complement encoding of a number, we mean its shortest possible encoding. Another useful property of the two's complement encoding is that it is easy to express the negation of a number: $-x = \neg x + 1$.

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 interpreted with the above semantics and each occurrence of a variable x_i evaluated to $val(x_i)$ is a true statement.

We say that F is satisfiable if there exists a valuation that satisfies F.

Now we describe how we can use QFPAbit formulae to define languages over Σ. Let F be a QFPAbit formula over the variables $V = \{x_1, ...x_n\}$. Let $w \in \Sigma^+$ be a word of length m. Denote $w(j)$ the j-th letter of w (with the first letter denoted $w(0)$). Since 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 least 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

Definition 2. *A (combinational) boolean circuit with n inputs and l outputs is a finite acyclic directed graph with exactly n vertices of in-degree zero and l vertices of out-degree zero. We denote them $v_1, ...v_n$ and $o_1, ...o_l$, respectively. All vertices of a non-zero in-degree have a logical function assigned to them and are called gates. All vertices of in-degree one represent a NOT-gate and vertices of greater in-degrees are either AND- or OR-gates.*

Given boolean values for the inputs, each gate can be evaluated in the natural way according to the logical function it represents.

The values of the outputs of a boolean circuit as defined above depend only on the present inputs and can be represented in a truth table. To obtain a more powerful computational paradigm, we will be working with the following definition clocked sequential circuits, defined as follows. A sequential circuit is equivalent to a deterministic finite automaton, but compactly represents the state space and the size of the transition function.

Definition 3. *A clocked sequential circuit (SC) (see Figure 1) consists of a combinational boolean circuit C and a set of D-type flip-flops. The data input of each flip-flop is connected to a unique output of C and the Q-output of each flip-flop is connected to a unique input of C. Such a backward-connected output-input pair will be denoted as a state variable. Inputs that do not receive their value from an output through a flip-flop will be called input variables.*

The circuit is assumed to work in clock pulses. In every clock pulse, it takes the values of its inputs and computes the output values. Via the flip-flops these values are routed back to the inputs for the use in the next clock cycle.

All the state variables are assumed to be provided with initial values stored in the flip-flops before the first clock cycle. The input variables need to be provided values from outside the system at every clock cycle.

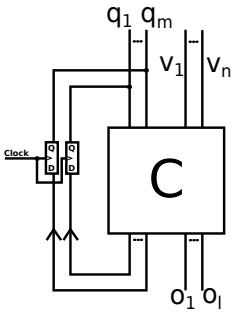

Fig. 1. Schema of a clocked sequential circuit

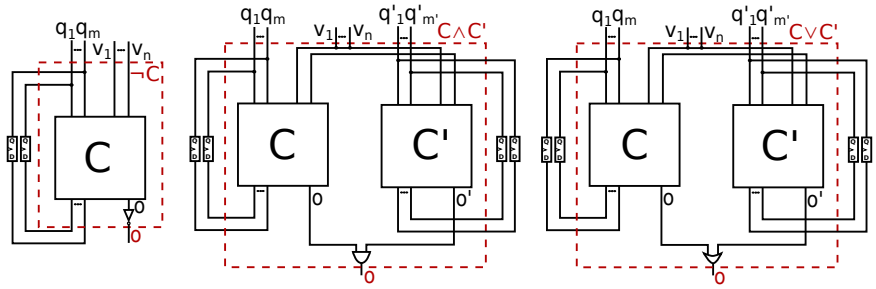

Fig. 2. Schemas of boolean combinations of circuits

Notice that a sequential circuit produces new output values at every clock cycle. Usually we will be interested in the free outputs that do not represent state variables. We will call these outputs output variables. A circuit with n input variables and m output variables can thus 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 SCs to recognize languages.

Definition 4. *Let C be a SC with one output variable o and n input variables. We say that C accepts the word $w \in \Sigma$ 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 $L(C) = \{w \in \Sigma | C \text{ accepts } w\}$.

Boolean operations on acceptor sequential circuits 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 circuits $\neg C$, $C \wedge C'$, $C \vee C'$ as in Figure 2. It can easily be seen that 1) $L(\neg C) = \Sigma^+ \backslash L(C)$; 2) $L(C \wedge C') = L(C) \cap L(C')$; 3) $L(C \vee C') = L(C) \cup L(C')$.

3 The Translations Between QFPAbit and Sequential Circuits

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 5. *We will call two QFPAbit formulae F_1 and F_2 equivalent, if the sets of their variables are V_1 and V_2 respectively, and for any valuation $val : V_1 \cup V_2 \to \mathbb{Z}$, val satisfies F_1 if and only if it satisfies F_2.*

If two formulae are equivalent in this sense then their language is the same.

Definition 6. *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 $\% \in \{=, \neq, <, \leq, >, \geq\}$. I 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. We 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. We remove all instances of multiplication by constants other than -1 and powers of two by using 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. We remove all instances of multiplication by -1 by replacing every sub-term of the form $(-1)T$ by $\neg T + 1$.
4. We 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. We 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. Finally, we replace every 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. We of course exclude the constants 2^k treated in the previous step.

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 and we will avoid discussing the details of these constructions. The circuit functionality mostly boils down to checking equality of two streams of binary values. The most complicated case is (iv), where we have to compare 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 cases (iii) and (iv) the number of state variables is proportional to the logarithm of the constant c and to k respectively.

We can compose the partial specification circuits by boolean operations to find a SC for G. The number of input variables of the circuit will be the same as that of the formula and the number of its state variables will be proportional to the formula's size. The projection of the language of the SC onto the coordinates corresponding to the variables of F will be the same as the language of F. Hence we will then be able to use the circuit to check, for example, the emptiness of the language of F (satisfiability of F).

In our prototype implementation of the algorithm, we have constructed the circuits so that the inputs need to be presented in the order of most significant digit coming first. This is a completely arbitrary choice and it is no more complicated to construct the circuits the other way round, although some of the basic sub-circuits do have to be different in that case.

3.2 Reduction from Sequential Circuits to QFPAbit

Let C be a sequential circuit with 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. We will construct a QFPAbit formula with variables $\{v_1, ...v_n, q_1, ...q_m, o_1, ...o_l\}$, such that for every satisfying assignment, the two's complement encodings of the values of the variables will describe the evolution of the values of the corresponding variables in a run of C. Although the QFPAbit variables have the same names as the variables of the circuit, it should be clear from the context which ones do we mean.

We will refer to the values of the variables of the automaton in the k-th clock cycle by $q_1(k), ...q_m(k)$,$v_1(k), ...v_n(k)$ and $o_1(k)...o_l(k)$. In the cycle when the inputs are $q_1(i), ...q_m(i), v_1(i), ...v_n(i)$, 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 will start the numbering of clock cycles from 0.

By our definition, a Boolean circuit is a directed acyclic graph. Hence, every output vertex is the root of a tree in which the direction of edges goes from children to parents and whose leaves are input vertices. Every inner vertex of the tree is endowed with a boolean logical function, and thus this tree can be interpreted as a propositional formula with the inputs of the circuit regarded as propositional variables. Let $\delta_{q1}, ...\delta_{qm}, \delta_{o1}, ...\delta_{ol}$ denote these formulae for the outputs $q_1, ...q_m, o_1, ...o_l$ respectively.

Then for all $j \in \{1, ..., m\}$,$k \in \{1, ..., l\}$ 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 three equations:

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

$$q_j(i+1) = \delta_{qj}(q_1(i), ...q_m(i), v_1(i), ...v_n(i)); \tag{2}$$

$$o_k(i) = \delta_{ok}(q_1(i), ...q_m(i), v_1(i), ...v_n(i)). \tag{3}$$

We will now build a QFPAbit formula for which every satisfying evaluation is such that the bit-sequences of the values it assigns to the variables conform to the above conditions.

Let $\bar{\delta}_{q1}, ...\bar{\delta}_{qm}, \bar{\delta}_{o1}, ...\bar{\delta}_{ol}$ be the QFPAbit terms obtained from $\delta_{q1}, ...\delta_{qm}, \delta_{o1}, ...\delta_{ol}$ by replacing every logical connective by a bit-vector logical operator. Also, notice that since we chose to interpret the numbers treated by the automaton as being presented with the most significant digit coming first, q_j is negative if and only if $I(q_j) = 1$. Now letting $\psi(q_j)$ be $q_j < 0$ or $q_j \geq 0$ if $I(q_j) = 1$ or $I(q_j) = 0$ respectively, the following formula can be used to describe the evolution of the digits of q_j:

$$((1 \bar{\vee} 2q_j = \bar{\delta}_{qj}) \vee (2q_j = \bar{\delta}_{qj})) \wedge \psi(q_i)$$

The justification is as follows. Multiplication by 2 induces a shift to the left of the two's complement encoding of a number. Hence the equality $2q_j = \bar{\delta}_{qj}$ establishes that every bit of δ_{qj} is equal to the next bit of q_j, apart from the least significant bit that has to be 0, since $2q_j$ is an even number. Taking a bitwise disjunction of a number with 1 forces the least significant bit to be 1 and preserves all other bits. Since the three conditions above do not put any restriction on the last bit of δ_{qj}, we use a disjunction of both the possibilities. Adding the conjunct $\psi(q_j)$ ensures that the most significant digit of q_j has the value $I(q_j)$. This ensures that equations 1 and 2 are satisfied.

Similarly, the formula $o_j = \bar{\delta}_{oj}$ asserts that the binary encoding of o_j corresponds to its values in the run of C on the given input as described by equation 3. Therefore we take the formula

$$F \equiv \left[\bigwedge_{j=1}^{m} ((1 \bar{\vee} 2q_j = \bar{\delta}_{qj}) \vee (2q_j = \bar{\delta}_{qj})) \wedge \psi(q_i)\right] \wedge \left[\bigwedge_{j=1}^{l} o_j = \bar{\delta}_{oj}\right]$$

as a description of the circuit C. For any given satisfying valuation of F, 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 run of C on the input word given by the two's complement encodings of $v_1, ...v_n$.

Now suppose that C has only one output o, so that it is an acceptor circuit defining a language, and consider the formula $F' \equiv F \wedge (o\bar{\wedge}1) \neq 0)$. The clause $(o\bar{\wedge}1) \neq 0$ is true if and only if the least significant digit of o is one. Suppose that C accepts a word w. This means that the output of C when it reads the last letter of w is one. It follows from the above discussion that this happens if and only if there exists a satisfying valuation val of F such that to $v_1, ..., v_n$ val assigns the values specified by w, and to the other variables of F it assigns values corresponding to the evolution of state variables and the output variable of C. Notice that this valuation is also unique. Hence, the language of F is the same as the language of C. It follows as a corrollary that the language of C is non-empty if and only if F' is satisfiable.

To summarize, we have described language-preserving 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 describes 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 that defines the same language.

4 Transducer Construction

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 [5]. 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 a potentially unbounded amount of memory of size proportional to the length of the input. This is a more expressive paradigm, allowing for functions for which it is not possible to determine the output before reading the whole input.

In contrast to [5], 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 [5].

In Section 4.2 we study a second optimization technique. How to exploit the circumstance when the specification formula is a conjunction of sub-formulas to build the transducer as a linear composition of smaller transducers for the conjuncts.

Definition 7. *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 8. *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 $\bar{i}, \bar{o}$, state variables $\bar{q}$ and one output variable determining the acceptance. This corresponds to the schema of Figure 1, only the input variables $\bar{v}$ of the circuit are partitioned into $\bar{i}$ and $\bar{o}$. Here by each of $\bar{i}, \bar{o}$ and $\bar{q}$ we actually mean vectors of variables wide n, l and m bits respectively. Therefore C has 2^m possible states and there are 2^l possible letters in the "output-alphabet". I 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 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. Notice that either of m_1, and m_2 may be 0 and $m_1 + m_2 = m$.

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 combinatorial circuit and the other two are sequential. We chose these names because their roles are analogous to those of the deterministic automaton A' and functions ϕ and τ in [5]. Our specification circuit C fulfills the responsibility of the specification automaton A used in [5].

C' performs two tasks. First, it runs the part of C that computes the sequence of values of $\bar{s}$ as C consumes $\bar{i}$ - remember, $\bar{s}$ depend on $\bar{o}$. 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 undergo 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. Notice that this is a consequence of our arbitrary choice that the numbers will be presented to the circuits starting with the most significant bit. If we were reading the encoding of the numbers in the reverse order, we would have to be storing k bits of t and it would be multiplication rather than division that does not cause state-space explosion. In fact, in the next section we will present another optimization often allowing us to avoid building a transducer for this kind of multiplication sub-formulas altogether, and loading efficient pre-constructed instances from a library.

The purpose of ϕ is to find inside the last 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. This means that τ identifies one particular state (i.e. values for $\bar{r}$) out of the stored set of possible states produced by C' and also an appropriate $\bar{o}$ letter for each clock cycle, so that the whole sequence of combined $\bar{r}$-states and $\bar{s}$-states together with $\bar{i}$ input letters and $\bar{o}$ input letters forms a valid accepting run of C. The constructed sequence of $\bar{o}$ letters 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.

The sub-circuits C_1 and C_2 represent the sub-circuits of C for computing $\bar{r}$ and $\bar{s}$ respectively. We let $C_{\bar{i}}$ be the projection automaton obtained from C_1 by projecting it onto the $\bar{i}$-variables. Notice that the only inputs to C_2 are already $\bar{i}$, so we do not need to *project* C_2. The intended meaning of the state variables $R_1, ...R_{2^{m_1}}$ of C' is that R_i is set to true if and only if at that point the non-deterministic automaton $C_{\bar{i}}$ could be in the state corresponding to i. 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_i's. Initially, A' is in a state where all variables R_i 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-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. Notice that for each sub-circuit C_1, at most one from the corresponding set of AND-gates below it returns a 1.

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, as was mentioned before, assume that the sequence of states this circuit undergoes while reading an input word is saved in a memory from where it can readily be read in a reversed 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 [5] 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 guess both a state from amongst those 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. It is a circuit very similar to that for C', also containing 2^{m_1+l} copies of C. However, since it only needs to be run for one clock cycle, it is a combinatorial circuit rather than a sequential one.

Finally, we use a very similar circuit for the function τ. It should in each clock cycle take 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 $\bar{o}$m 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 C 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 reversed order, compared to $\bar{i}$.

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

Definition 9. *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 on its highest level 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 $val : \bar{i} \cup \bar{o} \rightarrow \mathbb{Z}$ 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 satisfying 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. Methods how this dependency can be decided are described in [10] and [9].

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.*

If the conditions on the context of the multiplication described in this lemma do not hold, we may face constructing circuits whose size is proportional to the value of the constant 2^k.

5 Conclusion

We have presented an automated synthesis procedure for functions specified in QFPAbit and shown that QFPAbit is an adequate language for representing sequential circuits by providing polynomial-time translations between these two language specification paradigms. As a side-result, it follows from out constructions that the satisfiability problem of QFPAbit is PSPACE-complete.

The described synthesis procedure improves the previous work by two independent optimizations. We have built a prototype implementation which allowed us to show on examples that these techniques are working and 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. We thank Aarti Gupta for pointing to the related work in her PhD thesis [4] as well as Sharad Malik and Paolo Ienne for useful discussions.

References

1. J. Buchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138(295-311):5, 1969.
2. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
3. A. Fellah, H. Jürgensen, and S. Yu. Constructions for alternating finite automata? *International journal of computer mathematics*, 35(1-4):117–132, 1990.
4. A. Gupta. *Inductive Boolean Function Manipulation: A Hardware Verification Methodology for Automatic Induction*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1994.
5. 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.
6. T. Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40(1):25–31, 1991.
7. B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *FMCAD*, 2006.
8. N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. In *Proc. 5th International Conference on Implementation and Application of Automata*. LNCS, 2000.
9. J. Lang and P. Marquis. Complexity results for independence and definability in propositional logic. In *Principles of Knowledge Representation And Reasoning International Conference*, pages 356–367. Morgan Kaufmann Publishers, 1998.
10. J. Lang and P. Marquis. Two forms of dependence in propositional logic: controllability and definability. In *Proceedings of The National Conference on Artificial Inteligence*, pages 268–273. John Wiley & Sons LTD, 1998.
11. M. Rabin. *Automata on infinite objects and Church's problem*. Number 13. American Mathematical Society, 1972.
12. 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.

A Schema of Circuit C'