Verification Tool for Systolic Array Design

Fuyau Lin, Timothy Shih, and Nam Ling

Dept. of Computer Engineering
Santa Clara University
Santa Clara, CA 95053

Abstract

The axiomatization of STA defines rules for the systolic array into the language of the predicate calculus. The STA formalism is briefly reviewed and an automated verifier is constructed using Prolog. The verification tool is developed to produce sound and efficient verification process and provide shortcuts to justify systolic array designs. The STA specifications and the corresponding Prolog programs can be written using an almost identical notation.

1 Introduction

The axiomatization is a method for the definition of the semantics of programming languages. It goes back to the work of Hoare. Creating a specification of a systolic array involves the creation of an axiomatic theory in some logic. There are three important choices to be made in the process: the logic, the extralogical symbols, and the axiomatic theory. In our STA formalism, the STA statements of a systolic array design are written in a number of axioms of predicate calculus. The rules of inference of predicate calculus allow for formal reasoning regarding the systolic array design. In this paper, systolic array design probably means the same as systolic array specification and is just another expression for the name of the process we are considering, i.e. for descriptive STA theory building.

Many researchers have turned to automated tools to implement formal verification of hardware as well as systolic array architecture correctness [2, 6, 7, 16, 17, 14]. Prolog, one of the most popular logic programming languages, has served as a useful tool for proving the correctness of hardware [4, 3, 13, 5]. Clocksin [4] and Bosco [3] presented two different approaches by applying the technique of logic programming to problems in the design of CMOS transistor networks and logic simulators, respectively. These approaches describe the system in Horn-clause formulae and higher-order functions which are themselves amenable to formal manipulation.

In this paper, we briefly review the STA formalism and discuss a Prolog-based verifier which automates the verifications. Prolog is adopted for validating the STA formalism and the corresponding verification for systolic array designs due to following factors: ease of representing STA in Prolog terms; strength in symbolic manipulation; "don't know" nondeterminism; ease of meta-linguistic abstraction; and power of the resolution. Since Prolog is also applicable for lower level module and circuit verification (e.g. [13, 5]), this allows the forming of a multilevel formal verification system. Example is discussed in this paper to illustrate our framework.

2 Systolic Temporal Arithmetic

There are two major application of formal theory presentations: modeling the real world and specifying artifact to be built. We can use STA theory to specify aspects of a systolic array structure in terms of mathematical theories introduced by enriching these general theories with axioms constraining the behavior of the particular system being specified.

2.1 The Formalism

STA is a formalism developed to exploit the unique attributes of systolic arrays to provide constructs to simplify description and verification of systolic array architecture. Moreover, it is developed with multi-level abstraction reasoning in mind. The constructs and operators involved are similar to interval temporal logic [18], therefore coherent multi-level reasoning for systolic arrays can be easily developed. Three abstraction mechanisms from interval temporal logic are used to form STA: carrier abstraction, value and operation abstraction, and systolic feature abstraction.

One important feature of STA is the use of temporal variables, each has a value which is a sequential temporal connection of real numbers from the first to the last cycle. Using temporal variables enables
us to describe dynamic arithmetic values and operations effectively. The and operator \( \land \) and temporal operators such as henceforth \( \square \), next \( \bigcirc \), transport next \( \bigcirc^t \), etc., borrowed from interval temporal logic, are used to provide temporal connection of arithmetic entities to indicate arithmetic operations and values at different times. For example, a temporal variable \( u \), where \( u = u_0 \land \bigcirc u_1 \land \bigcirc^2 u_2 \land \bigcirc^3 u_4 \) indicates that \( u_0 \) is in the present cycle, \( u_1 \) in the next cycle, \( u_2 \) in the next next cycle, \( u_3 \) is the next next next cycle, and thereafter. To simplify the writing of some temporal variables, \( u \) can also be written as \( u = u_0 \land u_1 \land u_2 \land \bigcirc u_4 \), the symbol \( \bigcirc^k \) is created and defined as \( \bigcirc^k \equiv \square^k u_0 = \square^k u_2 \land u_4+1 \land \ldots \land u_k \) is adopted. Also, an underscore "\( _n \)" is used to denote an unknown value in the cycle.

### 2.2 STA Formal System

A formal language comprises two parts: its alphabet, which specifies what symbols are to be found in the language, and its syntax, which specifies how these symbols may be put together. An acceptable string of symbols in a language is called a well formed formula or wff of that language. Giving a formal language some semantics is achieved by giving the language an interpretation which assigns a value from some domains of interest to each wff. We list some of the identifiers and predicates used in this paper as follows:

- **STA identifiers** which are used to identify subjects:
  - \( \text{In}(l_j, B_i) \): input port \( l_j \) of component \( B_i \).
  - \( \text{Out}(o_j, B_i) \): output port \( o_j \) of component \( B_i \).
  - \( \text{Mem}(m_j, B_i) \): memory location \( m_j \) of component \( B_i \).
- **STA predicates** which are used in the construction of specifications:
  - \( \text{Ip}(B_i) \): \( B_i \) is an inner-product systolic cell.
  - \( \text{Conn}(X, Y) \): \( X \) and \( Y \) are directly connected.
  - \( \text{Val}(X, m) \): value of the data on the port (in the memory) location \( X \) is \( m \).

We add a deductive apparatus to the formal language, and the result is called a formal system. This deductive apparatus makes no reference to any particular interpretation of the formal language. The two components of a deductive apparatus are: axioms which are wffs written down without reference to any other wffs in the language; and inference rules which allow us to produce wffs in the language as immediate consequences of other wffs. STA temporal variables are operated by a few axioms, rules, and theorems developed for systolic array designs.

- **STA axioms**: Four axioms are used in STA
  - **A.** Let \( R \) be a binary arithmetic, logical, or relational operator which operates on two temporal variables, then quantities of the same cycle are \( R \) with each other.
  - **B.** Let \( R \) be a binary arithmetic or logical operator, then \( x \ R y = \ R z \) if \( x = \ R z \) i.e. non-strict semantic, if one operand of \( R \) is unknown, then the result is unknown.
  - **C.** Quantities that appeared in time before \( 0 \) are purely mathematical results; they have no practical meaning and can be ignored.
  - **D.** Two temporal quantities are equal if and only if their quantities are equal in each cycle.

These axioms can be easily justified by common sense temporal reasoning, properties.

- **STA rules**: Some rules concerning temporal operators (follow directly from temporal logic [1, 8]) that are also useful for systolic array reasoning are stated below:
  - **\( \Box \) rule:**
    \[
    \Box(u_0 \land u_1 \land \bigcirc^2 \ldots) = u_0 \land u_1 \land \bigcirc^2 u_2 \ldots = u_0 \land \bigcirc^2 u_2 \land \bigcirc^3 u_2 \ldots
    \]
    **\( \Box \) rules:**
    \[
    \Box x = x \land \Box \Box x
    \]
  - **\( \Box \) rule:**
    \[
    \Box^k x = \Box \Box^{k-1} x
    \]
    and,
    \[
    \Box^k \Box^l x = \Box^{k+l} x
    \]
- **STA theorems**: Two theorems are derived for reasoning bidirectional systolic arrays (see [11]).

Axioms are built-in theorems of an STA theory. Theorems can only be created by applying inference rules to axioms and other theorems. Axioms, rules, and theorems are useful for simplifications, deductions, and often provide convenient and natural short-cuts in proofs and reasonings for many systolic arrays. Each sentence in STA is written in a declarative manner, in the form of \( Q \Rightarrow P(s) \), which can be mapped to a truth value. Based on the STA language, we add to the normal inference system for predicate calculus the systolic array axioms for the STA theory. With the defined axioms and theorems, we can prove the output specifications as theorems.
3 STA Specification

A formal specification of a systolic array using STA is a mathematical description of its design structure and its intended behavior at the array architectural level. STA specification framework for systolic array consists of:

- **Structure specification**, $\Psi_S$, provides a complete physical description of the design at the array level which includes: Component type specification which describes the type of cells ($PE$s) and modules used to build the array; Structural connectivity specification describes the structural manner in which the cells are connected to form the entire systolic network.

- **Behavior specification** gives a complete functional description of the array which includes: Component function specification, $\Psi_F$, which gives the function of each cell ($PE$) or module used in composing the array; Input behavior specification, $\Psi_I$, which describes the intended dynamic pattern of the input data at each input of the systolic array (temporal variable expressions are usually used to express the input data); and Output behavior specification, $\Psi_O$, which expresses the intended dynamic pattern of the array outputs (or memory contents if these are treated as outputs).

Specifications are written in logic expression format which consist of identifiers, predicates, temporal variable expressions, and high-level constructs. Logic connectives such as $\land$, $\lor$, and $\rightarrow$ are adopted. The verification framework can be expressed by the expression:

$$\Psi_S, \Psi_F, \Psi_I \vdash_{STA} \Psi_O$$

### 3.1 2-D Systolic Array

A two-dimensional systolic array for matrix-matrix multiplication [9], as shown in Figure, is used as an example to briefly illustrate the framework. The array is used to multiply matrix $A$ ($\forall_{i,j}, 1 \leq i, j \leq n$) and the matrix $B$ ($\forall_{i,j}, 1 \leq i, j \leq n$) to produce the resulting product matrix $Y$ ($\forall_{i,j}, 1 \leq i, j \leq n$) where $y_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$. Matrix elements are input in a skewed manner as shown. The specification of this 2-D systolic array is expressed as follows $^1$:

- **Structure Specification** $\Psi_S$: specifies the type of cell used in the array and the local connections in both horizontal and vertical directions.

$$\forall 1 \leq i \leq n, \forall 1 \leq j \leq n \cdot Ip(Y_{ij}) \quad (1)$$

$$\forall 1 \leq i \leq n, \forall 1 \leq j \leq n \cdot Conn(Out(O_{a}, Y_{ij}), In(I_{a}, Y_{j+1})) \quad (2)$$

$$\forall 1 \leq j < n, \forall 1 \leq i \leq n \cdot Conn(Out(O_{b}, Y_{ij}), In(I_{b}, Y_{i+1,j})) \quad (3)$$

- **Component Function Specification** $\Psi_F$: specifies the functions of the cells used in the array.

$$\forall 1 \leq i < n, \forall 1 \leq j < n \cdot Ip(Y_{ij}) \land Val(In(I_{a}, Y_{ij}), a_{in}) \land$$

$$Val(In(I_{b}, Y_{ij}), b_{in}) \land Val(Mem(M, Y_{ij}), Y) \rightarrow Val(Mem(M_{a}, Y_{ij}), f(a_{in}, b_{in}, Y))$$

where $f(a_{in}, b_{in}, Y)$ is a short hand for

$$Val(Mem(M_{a}, Y_{ij}), \land \bigcirc (y + a_{in} \cdot b_{in})) \land$$

$$Val(Out(O_{a}, Y_{ij}), \land \bigcirc a_{in}) \land Val(Out(O_{b}, Y_{ij}), \land \bigcirc b_{in})$$

- **Input Behavior Specification** $\Psi_I$: specifies the inputs from horizontal and vertical.

$$\forall 1 \leq i \leq n \cdot Val(In(I_{a}, Y_{ij}), A_{r=0}^{n-1} \land$$

$$A_{r=0}^{n-1} a_{in} \land D_{0}) \quad (5)$$

$$\forall 1 \leq j \leq n \cdot Val(In(I_{b}, Y_{ij}), A_{r=0}^{n-1} \land$$

$$A_{r=0}^{n-1} b_{in} \land D_{0}) \quad (6)$$

\[1\] The specifications described here are not complete.
There are three kinds of specification clauses. If both the hypotheses and conclusion exist in a logic implication, the resulting clause is an inference rule. If a clause has only the conclusion part, the clause is called a fact. If a clause has only the hypotheses, the clause is a query. The output specification \( \Psi_0 \) to be proved is provided by the user to the verifier as a query, while the other specifications treated as facts or inference rules. Since STA involves some quantifier (e.g. \( \forall 1 \leq i \leq n \)), a fact (or the hypotheses of a rule) has a constraint. If the constraint is satisfied, the fact is valid. And thus it can be used in a deduction step. The inference rules are used in the unfolding process.

### 4.2 The STA Verifier

The STA verifier introduced a new approach to systolic array verification: embedding the STA in a logic programming language, Prolog [18]. STA formulae are terms in Prolog with the help of the operator definitions. All the specifications are instances of an abstract data type \( \text{WF} \). For backwards proof, goals, subgoals, tactics, and heuristics are implemented on top of this abstract data type. Each rule checks that it has received suitable hypotheses, then generates the conclusion, while each tactic checks that it has received a suitable goal then generates the subgoals.

The STA verifier inherits the tradition in favor of unification over one-way matching. An inference rules matches theorems of a given pattern; a tactic matches goals. For backwards proof, an inference rule is a tactic, it takes place by matching a goal with the conclusion of a rule; the hypotheses become the subgoals. Prolog's resolution mechanism allows variables in clauses, whereby one clause stands for a set of ground clauses. For the STA verifier, inference rules are mutant Horn clauses; the composition of rules is resolution. It applies resolution to build proof trees from the inference rules of an STA specification. STA verifier adopts backward chaining and rewriting to perform an induction proof of the goal.

Let \( P(n) \) be the expression \( (G) \) to be proved for a systolic array of size \( n \). For an induction proof, we have to show that \( P(n_0) \) is correct \((n_0 < n)\); assume that \( P(k) \) is correct for \( k \geq n_0 \) then show \( P(k + 1) \) is correct on the basis of the inductive hypothesis (i.e. \( P(k) \rightarrow P(k + 1) \)). Based on the nature of the given specifications, STA verifier takes care of six kinds of goals. Each kind of goal is associated with its own computation rule described as the following:

* constrained goal: \( \text{Constraints such that Goal} \)
  The constraints are solved before the goal.

* conjunction goal: \( \text{GoalA AND GoalB} \)
  Goal A is solved before goal B.
value predicate: \text{val}(\text{Port}, \text{TV}), \text{where TV is not normalized}. If the temporal variable of the port is not normalized, the variable is evaluated.

help predicate: as regular Prolog subgoals. Goals such as array size \text{sa.size}(\text{W}) and number of rows \text{nof.rows}(\text{W}) are checked.

valid fact: \text{Constraints suchthat Goal}. If constraints are satisfied, then the goal is solved.

unfolding rule: \text{Hypotheses implies Goal}. The goal is unfolded to hypotheses then hypotheses are solved in the next iteration.

Each kind of goal is handled by a sub-component in the verifier. Sub-components can call each other recursively until the empty goal is derived or an exception occurred and the verifier aborted by the user.

5 Summary

We have presented an STA theory for systolic array design and the implementation of its verification tool. The STA verifier is a semi-automatic resolution theorem prover using mathematical induction and deduction techniques. A set of rewriting rules (STA rules) are defined in the system as default axioms. All the axioms, theorems, and the specification are represented in Prolog terms which realized the abstract data type \text{WFF}. The STA notation in Prolog is easy to understand and debug. Prolog is adopted due to its popularity and its closeness in representing STA predicate-type notation. The proof is generated by the verifier with the help of the user interaction and heuristics. We discussed a systolic array verification framework. We are seeking for more generic heuristics or tactics for the system, especially those for temporal logic. We believe that the STA theory will provide a simple and efficient device for the formal design and verification of the systolic array architecture.

References


