Proving Parameterized Systems: the use of pseudo-pipelines in polyhedral logic

Katell Morin-Allory[1] and David Cachera[2]

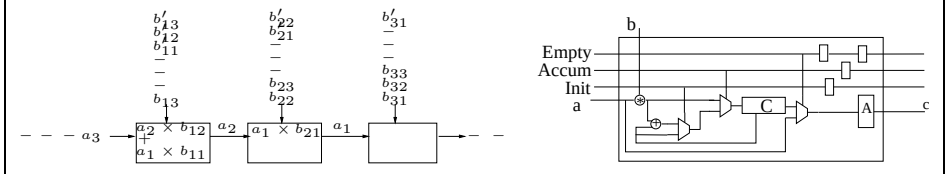
[1] TIMA, 46 avenue Félix Viallet, 38031 Grenoble, France
[2] IRISA - ENS Cachan (Bretagne), Campus de Beaulieu F-35042 Rennes, France

1 Introduction

The polyhedral model mixes recurrence equations over polyhedral domains and affine dependency functions. This model provides a unified framework for reasoning about regular systems composed of both hardware and software parts. Systems are described in a generic manner through the use of symbolic parameters, and structuring mechanisms allow for hierarchical specifications. The ALPHA language [3] and the MMALPHA environment [4] provide a syntax and a programming environment to define and manipulate polyhedral equation systems. High-level system specifications are refined through a user-guided series of automatic transformations, down to an implementable description, from which may be derived C code or a VHDL architecture. For hardware components and interfaces, control signals are generated to validate computations or data transfers. The use of systematic and semi-automatic rewritings together with the clean semantic basis provided by the polyhedral model should ensure the correctness of the final implementation. However, interface and control signal generators are not certified, and hand-made optimisations are still performed to tune the final result. As a consequence, the correctness of control signals has to be checked at the lower level of description, in the presence of *symbolic parameters*. A formal verification tool that benefits from the intrinsic regularity of the model has been developed to (partially) certify low-level system descriptions [2], based on polyhedra manipulation. The present work develops new strategies to prove a wider class of formulae. The basic idea is to detect particular patterns in the definition of signals, that characterise the propagation of known values along spatial or temporal dependencies, and to define a widening operator that allows for the automatic determination of how this propagation can be useful in the proof process.

2 The Polyhedral Model

An example of a modelled system. We introduce the model on the example of a system designed to compute a sequence of matrix-vector products. It consists of a linear array of N cells, N being a symbolic parameter carrying any integer value. The vector coefficients and the N column of the matrix are input sequentially, and each cell computes one coefficient of the output vector. Input vector coefficients and output values are propagated from left to right in the array, through register A. The behaviour of each cell depends on its position in the array and on the time elapsed since the beginning of the computation. Three boolean *control signals* are thus added to precisely control the behaviour of operators and registers: when *Init* is set to true, it initialises register C,



(a) The array and its data inputs, for $N = 3$ and $t = 4$. The $-$ represent insignifi cant values.

(b) One cell

Fig. 1. Structure of an array computing a matrix-vector product.

Accum accumulates the product $a \times b$ in C and *Empty* outputs the value of register C in register A.

Describing this system in the polyhedral model. Each signal is represented by a function called a *polyhedral variable*. The vector of all cells registers A is a mapping from $\mathbb{N} \times [0, N]$ to the boolean set. This mapping is defined by an *affine recurrence equation* composed of three *branches*:

$$A = \begin{cases} \{t, i \mid t = 0; 0 \leq i \leq N\} : & 0 & (1) \\ \{t, i \mid i = 0\} : & a.(t, i \rightarrow t) & (2) \\ \{t, i \mid t > 0; 0 < i \leq N\} : & \text{if Empty.(t, i} \rightarrow \text{t, i) then C.(t, i} \rightarrow \text{t} - 1, \text{i)} \\ & \text{else A.(t, i} \rightarrow \text{t} - 1, \text{i} - 1) & (3) \end{cases}$$

Let us focus on the third branch: $\{t, i \mid t > 0; 0 < i \leq N\}$ denotes a *polyhedral domain, i.e.,* a subset of $\mathbb{Z}^n$ bounded by a finite number of hyperplanes. The dimension of this domain (2 in this example) is also the dimension of variable A. Terms like $(t, i \rightarrow t - 1, i - 1)$ denote *dependency functions, i.e.,* affine mappings between polyhedral domains. We concentrate on *uniform* dependencies, *i.e.* translations by a vector: dependency $(t, i \rightarrow t - 1, i - 1)$ is the translation by vector $(-1, -1)$. The "." notation denotes the composition of functions: $C.(t, i \rightarrow t - 1, i)$ thus represents the mapping $(t, i) \mapsto C(t-1, i)$. Note that we have a *self-dependency* on A in (3). A *polyhedral system* is a set of such affine recurrence equations. Polyhedral systems are *parameterised* with symbolic parameters that are in turn defined on polyhedral domains, and can be seen as additional dimensions on all variables. We only consider systems for which an order in which computations should take place, has been determined, and assume that a particular index (say, the first one, denoted t) is considered as the *temporal index*. Such a system is called a *scheduled system*.

The combination of recurrence equations with polyhedral domains provide a rich mathematical and computational basis for program transformations. RTL descriptions can thus be obtained by derivation from a high-level algorithmic description.

3 Proofs for polyhedral systems

To formally establish properties of systems described in the polyhedral model, such as validity of a given control signal on a given set of time and space indices, we have developed a proof method and a proof tool[1]. Properties of the system are described in

a so-called *polyhedral logic*: a formula is of the form $\mathcal{D} : e \downarrow v$, where $\mathcal{D}$ is a polyhedral domain, e a polyhedral multidimensional expression, and v a boolean scalar value. Proofs for such formulae are constructed by means of a set of inference rules, that are of two kinds: (i) "classical" propositional rules, and (ii) rules specific to the model, based on heuristics using rewritings and polyhedral computations (*e.g.* intersection of polyhedra). The proof tool uses these rules to automatically construct a proof tree, whose root is the initial formula we want to prove. This tool is able to establish simple inductive properties in connection with propagation of boolean values in multidimensional arrays. If formula $\mathcal{D} : e \downarrow v$ is proved, the soundness of the set of rules ensures that the value of e on $\mathcal{D}$ is v. If the proof construction fails on a given node, this node is called a *pending leaf*.

4 Pseudo-Pipelines and Widenings

Since the proof rules described in Section 3 are not complete, we have developed new heuristics to increase the effectiveness of our tool, based on the notion of *pseudo-pipelines*. In a hardware system, pipelined variables are used to transmit values from cell to cell without modifying them. We extend this notion to a less specific one by allowing a more general form of dependencies.

Definition 1 (Pseudo-Pipeline). *A* pseudo-pipeline *is a polyhedral variable X such that one of its branch is defined by an expression e such that:(a) e is in disjunctive (resp. conjunctive) normal form, (b) e contains at least one occurrence of X, (c) each conjunct (resp. disjunct) of e is either a single occurrence of X composed with a dependency d, or a polyhedral expression without any occurrence of X.*

A general form for a pseudo-pipeline is $X = \left\{ \begin{smallmatrix} \mathcal{D}_1 : X.d \wedge e \\ \mathcal{D}_2 : f \end{smallmatrix} \right.$ where e and f are polyhedral expressions. Like pipelines, pseudo-pipelines frequently appear in low-level description of systems, since they are used to compute reduction of boolean operators over a given set of signals, either in a temporal or spatial dimension.

The notion of pseudo-pipeline is a syntactic one. A pseudo-pipeline is characterised by a *propagation direction* d, which corresponds to the self-dependency occurring in its defining expression. The fundamental property of pseudo-pipelines is informally stated as follows: *If a pseudo-pipeline X of propagation direction d is true (resp. false) on a given point z_0, then there exists a domain $\mathcal{D}_{d,z_0}$ on which X is true (resp. false). $\mathcal{D}_{d,z_0}$ is an extension (potentially infinite) of $\{z_0\}$, either in the direction of d, or in the opposite one, depending on the boolean operators and truth values involved.*

This property illustrates the propagation of a value for one instance in a domain. It can be generalised to a whole domain by iteratively computing the image (or preimage) of the domain by the dependency: we widen the domain in the dependency direction. Since the domain $\mathcal{D}_{d,z_0}$ is not strictly a polyhedral domain, we have to extend it by taking its convex hull. The formal definition of our widening operator is:

Definition 2 (widening along a dependency). *Let $\mathcal{D}$ be a domain of dimension n and d a dependency from $\mathbb{Z}^n$ to $\mathbb{Z}^n$. The widening of domain $\mathcal{D}$ by dependency d is the set:*

$$\mathcal{D} \mathbin{\widetilde{\nabla}} d = convex.hull(\{z \mid \exists z_0 \in \mathcal{D}, \exists i \in \mathbb{N},\ z = d^i(z_0)\})$$

The alternative representation of polyhedra, as linear combinations of lines, rays and vertices, allow for a simple computation of convex hulls.

Use of widenings in the proof construction We now show how widenings are used to generate new lemmas. Let $f = \mathcal{D} : e \downarrow v$ be a formula labelling a pending leaf in the proof tree. For all variables occurring in e, a procedure is used to detect if it is a pseudo-pipeline. Let X be such a variable, and d the dependency associated to X in e. In the definition of X, we look for a subdomain $\mathcal{D}_0$ where X is defined by a boolean constant v', and we determine the direction d' of propagation. This direction is given by either d or d^{-1}, depending on the value of v'. The domain $\mathcal{D}_0 \stackrel{\sim}{\nabla} d'$ is then computed and intersected with $d(\mathcal{D})$, the domain on which the dependency d is valid. Let $\mathcal{D}'$ be the resulting domain. All occurrences of $X.d$ defined on $\mathcal{D}'$ may now be substituted by v'. Formula f is thus simplified by this substitution and we get formula $f' = \mathcal{D} : e' \downarrow v'$. Formulae f and f' are semantically equivalent. The proof construction then resumes with formula f' with these new domains and equations.

5 Conclusion

In this paper, we have presented heuristic strategies to generate new lemmas in order to improve the efficiency of proofs for systems described in the polyhedral model. Specifications of the system are described in a polyhedral logic close to the model, and the general proof mechanism relies on proof rules that exploit the expressivity and the computational power of the model. The proposed strategies consist in detecting particular value propagation schemes in the equations defining the variables, and to widen the index domains on which the proof has to be made. The proof rules are implemented within MMALPHAusing the PolyLib [5] . The heuristics greatly improve the effectivity of our verification tool. The proof tool is intended to work at a relatively low description level in the synthesis flow. At this level of detail, there are many signals defined by means of pipelines or pseudo-pipelines. As an example, our heuristics were able to establish the correctness of a hardware arbiter for mutual exclusion.

References

1. D. Cachera and K. Morin-Allory. Proving parameterized systems: the use of a widening operator and pseudo-pipelines in polyhedral logic. Technical report, TIMA, April 2005.
2. D. Cachera and K. Morin-Allory. Verifi cation of safety properties for parameterized regular systems. *Trans. on Embedded Computing Sys.*, 4(2):228–266, 2005.
3. C. Mauras. *Alpha : un langage équationnel pour la conception et la programmation d'architectures systoliques*. PhD thesis, Univ. Rennes I, France, December 1989.
4. D.K. Wilde. A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France, jan 1993.
5. D.K. Wilde. The Alpha language. Technical Report 999, IRISA, Rennes, France, jan 1994.