The Power of Polynomials

Work in Progress

Paul Feautrier
ENS de Lyon, LIP, INRIA, CNRS, UCBL
Paul.Feautrier@ens-lyon.fr

ABSTRACT

Every component in the program development chain uses a model to represent and reason about its source. The model must be as expressive as possible without compromising its efficiency and tractability. This paper proposes a slight extension to the polyhedral model by allowing polynomial constraints and relations. Recent mathematical results by Handelman and Schweighofer on the *Positivstellensatz* allow one to devise algorithms similar to familiar emptiness tests or the Farkas algorithm. This paper presents applications of these ideas to three use-cases: dependence tests, scheduling and transitive closure approximation. It then points to unsolved problems and future work.

1. MOTIVATION

Every compiler first starts by building an intermediate representation (IR) for the given source program. Most IRs are syntactical: the Abstract Syntax Tree, for instance, is a data structure which closely represents the input program, while abstracting away details like text layout or syntactical variations. In contrast, the polyhedral model represents the program as it will run, and considers the operations to be executed and their execution order. Since with present day processors the number of operations is to be counted in billions, the set of operations has to be represented in intention, and all operations on this set have to be executed symbolically. Hence the search for representations in which expressive power has to be traded against effectiveness.

The polyhedral model is such a representation: every set – operations, execution order, dependences, memory accesses, memory footprints footprints, schedules – are represented as Z-polyhedra (sets of integer solutions to a system of affine inequalities). This model has met with a fair measure of success; however its expressive power is limited. The time has come to search for more powerful representations.

In this search, one is guided by the following observation: many algorithms of the polyhedral model have been implicitly or explicitly devised in order to avoid using polynomials. A case in point is the recourse to multidimensional schedules when a one dimensional schedule cannot fit. In [8] I proved (see Theorem 1), that a multidimensional schedule can be converted to a unidimensional one by counting clock ticks. This count can be computed using the theory of Ehrhart polynomials [4], and the result is a polynomial or quasi-polynomial schedule. The same remark applies to array and channels: in CRP, I have introduced multidimensional channels in order to avoid linearization functions, which, despite their name, are polynomials [9].

This paper is organized as follows: I will first review recent mathematical results for the PositivStellenSatz, which play the same role for polynomials as Farkas lemma does for affine inequalities. I will then present a very preliminary implementation of the corresponding algorithm. I will apply it to several common use-cases: dependence calculations, scheduling and transitive closure. I will then review related work, and point to extensions and improvements.

2. MATHEMATICAL BACKGROUND

Let $p_1, \ldots, p_n$ be polynomials in d variables. The set:

$$S = \{x \in \mathbb{R}^d \mid p_1(x) \geq 0, p_2(x) \geq 0, \ldots p_n(x) \geq 0\} \quad (1)$$

is a *semi-algebraic set*. Polyhedra are special cases of semi-algebraic sets where all p_is are of first degree. Such sets arise in program analysis and optimization. Important questions are: is S empty, and, given a polynomial (or polynomial *template*) p, is p positive in S? A succession of recent results, starting with a paper by Krivine [14], has provided answers to some of these questions.

THEOREM 1 (HANDELMAN [12]). *Let S be a compact polyhedron as defined by (1), where all p_i are of first degree. A polynomial p is strictly positive in S if and only if it can be represented as*

$$p(x) = \lambda_0 + \sum_{\vec{e} \in \mathbb{N}^n} \lambda_{\vec{e}} p_1^{e_1}(x) \ldots p_n^{e_n}(x), \quad (2)$$

where the $\lambda_{\vec{e}}$ are non-negative and not all of them are zero.

One should notice that Farkas lemma is the special case of this result when p is affine.

THEOREM 2 (SCHWEIGHOFER [16]). *Assume now that some of the p_i in (1) are polynomial of higher degree. A polynomial p is strictly positive in S if it has a representation (2), provided that the p_i of degree one define a compact polyhedron.*

IMPACT 2015
Fifth International Workshop on Polyhedral Compilation Techniques
Jan 19, 2015, Amsterdam, The Netherlands
In conjunction with HiPEAC 2015.

http://impact.gforge.inria.fr/impact2015
This work has been partially supported by the ManycoreLabs project PIA-6394 led by Kalrays.

In the words of Schweighofer, the representation (2) is valid "if there are enough polynomials of degree one" in S.

Their formal similarity notwithstanding, these results have a different status than Farkas lemma. In the later case, one knows exactly to which terms the sum in (2) extends, and if no solution exists, one may assert that p is not positive in S. If no solution is found when applying Theorem 1 or 2, it may be that p is not positive, or that the sum (2) has not been extended far enough. From a pragmatic point of view, if a representation (2) has been found, it can be checked by elementary algebra, and acquires the same status as the elementary identities we all learned in high-school, whether the hypotheses of either theorem are satisfied or not. If no solution is found, one can either conclude conservatively that p may not be positive in S, or try to increase the number of terms in (2), or reformulate the problem, e.g. by trying to prove that $-p$ or $p+a$, a a constant, is positive.

3. IMPLEMENTATION

It is easy to see that, when p and the set of products in (2) are given, the solution can be found by Linear Programming: solving a system of linear equations in positive unknowns, the λs. The usefulness of the method can be augmented by the following trick: instead of a polynomial p, use a polynomial *template*, i.e. a polynomial which depends on additional parameters. To fit into the above method, parameters must occur linearly in p. The parameters can then be considered as additional unknowns in the LP problem, and solved for at the same time. The corresponding algorithm is not only able to prove that a polynomial is positive in a given set, but also to find a polynomial of a given shape that is positive.

Algorithm H

1. Given
   - A set of polynomials $P = \{p_1(x), \ldots, p_n(x)\}$ on d variables $x = (x_1, \ldots, x_d)$, including the trivial polynomial 1.
   - a template $p(\mu, x)$ depending linearly on a set of parameters μ.
   - an "order" M
2. Generate all products of M polynomials from P.
3. Compute the "master equation":
$$E = p(\mu, x) - \sum_{\vec{e} \in \mathbb{N}^n, \sum_{i=1}^n e_i = M} \lambda_e p_1^{e_1}(x) \ldots p_n^{e_n}(x) = 0$$
4. In the fully distributed form of E, each monomial $x_1^{f_1} \ldots x_d^{f_d}$ may occur several times with coefficients linear in the λ and μ. Sum these coefficients and equate the result to zero.
5. Solve the resulting system for the λ and μ by any convenient LP software.

In step 2, since multiplication is commutative, one should be careful to avoid duplicate products. Notice also that since polynomials can be multiplied whatever their degree, this algorithm covers both the Handelman and Schweighofer cases.

A proof-of-concept implementation of this algorithm has been realized, using a home-made algebraic library which is part of the Syntol[1] project and the LP tool PIP[2].

The examples in the following section have been solved by writing, for each use case, a front end which sets up the list of the p_i and the unknown polynomial p, call algorithm H, and edit the result in proper form.

4. USE CASES

4.1 Dependences

Dependence testing requires to decide whether two array accesses touch the same memory cell or not. This is usually formulated as deciding if a set of constraints built from the iteration domains of two (not necessarily distinct) statements, an execution order predicate, and the equality of two subscript vectors is empty or not. If subscripts are affine in the surrounding loop counters, the answer can be obtained by linear programming. However, in some cases, subscripts are polynomials. For instance, a many dimensional array may have been linearized, either by an overeager compiler, or by the programmer because there is no way in C to dynamically allocate a multidimensional array. In other cases, several mathematical objects with disjoint supports may have been compacted in only one array. Consider for instance the following code:

```
for(i=0; i<N; i++)
  for(j=0; j<N; j++)
    A[N*i+j] = 0;
```

Let us prove that this program has no dependence. Consider two iterations (i, j) and (i', j'). For a dependence to exist, these variables must satisfy the following constraints:

$$\begin{aligned} 0 \le i \le N-1 \quad &, \quad 0 \le j \le N-1, \\ 0 \le i' \le N-1 \quad &, \quad 0 \le j' \le N-1 \\ Ni + j \quad &= \quad Ni' + j' \\ i < i' \quad &\vee \quad (i = i' \wedge j + 1 \le j') \end{aligned} \tag{3}$$

The last constraint splits in two cases, and in the second case the problem becomes linear. In the first case, it is possible to apply algorithm H to prove that $Ni' + j' - Ni - j \ge 1$, and hence that there is no dependence. A surprising solution is to show, still using algorithm H, that -1 is a positive combinations of the constraints (3)! In fact, the reader may care to check that:

$$\begin{aligned} -1 \quad = \quad & (N-i-1)(i'-i-1) + i(i'-i-1) + (i-i'-1) \\ & + j' + (N-j-1) + (Ni + j - Ni' - j') \end{aligned} \tag{4}$$

This apparent paradox can be solved by observing that since -1 cannot be positive, it follows that for every value of $i, j, i'j'$ at least one constraint of (3) is unsatisfied, and hence that (3) is empty, which is the desired result. The situation is reminiscent of the Fourier-Motzkin algorithm, where an absurd inequality may be a consequence of an unfeasible initial system. Another observation is that (4) can be rewritten

$$\begin{aligned} Ni' + j' - Ni - j \quad = \quad & 1 + (N-i-1)(i'-i-1) + i(i'-i-1) \\ & + (i'-i-1) + j' + (N-j-1) \end{aligned}$$

[1] `perso.ens-lyon.fr/paul.feautrier/Syntol`
[2] `www.piplib.org`

thus proving that $Ni'+j'-Ni-j$ is always strictly positive.

Observe that, either way, the hypotheses of theorem 2 are not satisfied, since in particular the base set is not compact, due to the presence of the unbounded parameter N. However, this does not detract from the truth of (4). It is true that for each numerical value of N the set is compact, and the surprising fact is that all numerical solutions can be subsumed by one identity.

One can extend this method to compute various dependence approximations: dependence distances, dependence directions, ... simply by adding more constraints to the initial problem.

4.2 Scheduling

A schedule is a non-negative function θ from the set of operations of a program to some ordered set, such that if operation u is the source of a dependence whose destination is v, then $\theta(u) < \theta(v)$. One can say that $\theta(u)$ is the (logical) date at which u is to be executed. In the polyhedral model, a template for θ can be constructed using Farkas lemma, and then the above inequality (the *causality constraint*) can be reduced by another application of Farkas lemma. The resulting system of equations in positive unknowns can then be solved using linear programming. However, it may be that this system is unfeasible. One then has recourse to multidimensional schedules. See [7, 8] for details.

One can apply algorithm H in case either the operation sets or the dependences or the schedule involve polynomials. Consider for instance the simple program:

```
for(i=0; i<N; i++)
  for(j=0; j<N; j++)
    s+= a[i][j];
```

This program clearly runs in time $O(N^2)$ hence a schedule affine in N cannot fit. One can apply algorithm H first at order 2 to build a quadratic template. The dependence of this program splits in two cases, $i < i'$ and $i = i' \wedge j < j'$. Hence algorithm H must be applied twice to solve the causality constraints. The two linear systems thus constructed must be merged and solved in conjunction.

For the program above, one finds the expected result, $\theta(i,j) = N.i + j$. For a slight variation:

```
for(i=0; i<N; i++)
  for(j=0; j<i; j++)
    s+= a[i][j];
```

the result is less satisfactory: $\theta(i,j) = i^2/2 + j$. In fact, this schedule is perfectly valid but not optimal. This is because the solution given by PIP depends on the ordering of the unknowns. Finding better ways of selecting an optimal schedule is left for future work.

When the dependences are expressed as functions, this approach can be greatly simplified. This is the case, for instance, when dealing with systems of affine recurrence equations, or when the dependences result from a dataflow analysis. Consider for instance Example 2.2 in of Achtzinger et. al. paper [1]:

$$a(x,y) = g(a(x,y-1)), a(x-1,x)),$$

in the domain:

$$D = \{2 \leq x \leq n, 4 \leq y \leq n, n-y \leq x\}.$$

Each dependence can be represent as a *source function*. For instance, the source for the second argument of g above at iteration (x,y) is $(x-1,x)$. Hence, the corresponding causality condition, which must be true in D, is

$$d = \theta(x,y) - \theta(x-1,x) \geq 1.$$

The solution then proceeds as before:

- construct a template for θ and substitute it into d,
- express d as a positive combination of products of inequalities defining d,
- solve for the multipliers.

The solution for this problem is:

$$\theta(x,y) = \frac{n + x^2 - 5x}{2} + y - 2$$

and is identical, up to the constant factors, to the result of Achtziger et. al.

It is not clear yet how to generate code from a polynomial schedule, in the manner of CLooG[3]. However, in some cases, the mere *existence* of a schedule is enough to prove, e.g. the absence of deadlocks.

Lastly, it has been shown in [2] that similar techniques can be used to build *ranking functions* and prove program termination. Mutatis mutandis, the present proposal can be used for the construction of polynomial ranking functions.

4.3 The Return of the Transitive Closure

In a previous paper [10], I have shown that a reflexive and symmetric relation can always be represented in one of the two forms:

$$R(x,x') \equiv f(x) \preceq f(x') \quad (5)$$

$$R(x,x') \equiv f(x) = f(x') \quad (6)$$

where f is a function of the universe of R to a set X ordered by $\preceq$. The second formulation applies only when R is an equivalence.

If R is an arbitrary relation, and if:

$$R(x,x') \Rightarrow f(x) \preceq f(x'),$$

then the relation associated to f and $\preceq$ is an upper approximation to the reflexive and transitive closure of R. Assume that we select the rationals with the ordinary order $\leq$ for X and polynomials for f: we are in a position to apply algorithm H to find one or more solutions. Each solution f generates a transitive relation $f(x) \preceq f(x')$, and, since the intersection of several transitive relations is transitive, their conjunction is an improved approximation to the exact result. The only difficulty is that we do not have guidelines for selecting a template for f. A possible approach is to take for f a polynomial of given degree with arbitrary coefficients. Another observation is that the problem is homogeneous, and hence the set of valid solutions is a polyhedral cone. Each ray of this cone will generate an independent f. If the solution cone has lines, the corresponding constraints will be equalities.

This does not apply to the case of an equivalence relation (6) since here the polynomial $f(x') - f(x)$ is not *strictly* positive. This situation occurs most often when R includes

[3] `www.CLooG.org`

a conjunction of equalities, as is the case when R is written as a guard and an assignment $x' := a(x)$ in the manner of the FAST notation [3]. But here a simpler solution is possible. Write R as a substitution $\sigma = [x' \leftarrow a(x)]$. A polynomial f is a possible solution if it is left invariant by σ, i.e. $f(x) - f(x')\sigma = 0$. Equating the coefficient of each monomial in this equation to zero, one gets a system of homogeneous linear equations in the coefficients of f, which can be solved by Gaussian elimination. The solution will usually depend on residual unknowns, which may be given arbitrary values to generate as many independent f.

Consider for instance $R(x, y, i) \equiv \{x' = x + y, y' = y, i' = i + 1\}$. Select for f an arbitrary quadratic polynomial in x, y, i. For brevity, assume some oracle has told one that it is enough to take $f(x, y, i) = \alpha x + \beta i.y + \gamma y$, α, β and γ the unknown coefficients. The difference to be set to zero is:

$$\alpha(x + y) + \beta(i + 1).y - \alpha x - \beta i.y = (\alpha + \beta)y.$$

Hence one solution is $\alpha = 1, \beta = -1, \gamma = 0$ and $f(x, y, i) = x - iy$, giving the constraint $x - iy = x' - i'y'$. Another solution is $\alpha = \beta = 0, \gamma = 1$, giving $y' = y$. If more terms had been included in f, like x^2 or $i.x$, their coefficients would have been found to be zero.

This algorithm can easily be extended to the case of a disjunction of equational constraints. How to handle the full FAST format, with a mixture of inequalities (guards) and equalities is left for future work.

One may wonder what is the status of i, which clearly has been added artificially to help in finding a solution. It is easy to see that firstly $i' - i$ takes only integer values, and that $\{x' - y'(i + n) = x - iy\}$ is the n-th iterate of $S \equiv \{x' = x+y, y' = y\}$. The transitive closure of S is found by eliminating i in integers, and is[4]. $\{y \mid (x' - x), y' = y\}$. Again, exploring and extending these observations is left for future work.

5. RELATED WORK

The problem of testing dependences in the presence of non linear terms has been studied by Pugh and Wonnacott [15] and by van Engelen. et. al. [17]. Pugh and Wonnacott first extend their linear programming tool Omega to handle uninterpreted functions, and then replace any non-linear term in the dependence problem by such functions. While Omega proper is able to decide all problems of Pressburger arithmetic, the adjunction of uninterpreted functions renders its logic undecidable, and one may obtain an "unknown" answer in some cases. van Engelen et. al. convert all non-linear expressions to Chains of Recurrence, which were introduced by E. Zima [18] and are especially convenient to handle polynomials. The formalism is then used to do a range analysis on subscript expressions and decides the existence or absence of a dependence.

Z3 [13] is an SMT solver which has recently be extended to handle polynomial problems in rational and integer variables. Despite the fact that integer problems are undecidable (Hilbert 10th problem), Z3 uses heuristics to get an answer in favorable cases. There is no doubt, for instance, that Z3 would have been able to solve the example in Sect. 4.1. However, Z3, like other SMT solvers, gives a yes-or-no answer and a witness for its decision. Using it for more complex tasks would need tinkering with its source code.

[4]Here, the vertical bar is the divide predicate

The aim of Achtziger and Zimmermann work [1] is to build polyhedral schedules of degree 2. For this special case, they are able to summarize the causality constraints by a set of inequalities involving the vertices of the iteration domains. In this way, the number of constraints to be considered no longer depends on the cardinality of the domains, but they may depends on the parameters of the program. For each values of the parameters, they show how to optimize the latency of a schedule using a numerical method.

Bernstein polynomials provide another method for dealing with arbitrary polynomials in program analysis. A Bernstein polynomial of degree N in $[0, 1]$ is one of the terms of the expansion of $(x + (1 - x))^N$ by Newton's binomial formula. The theory can be extended to Bernstein polynomials in arbitrary polytopes using barycentric coordinates [6]. Since Bernstein polynomials are positive and sum to 1, the value of of a linear combination of Bernstein polynomials lies between its minimal and maximal coefficient, and if all coefficients are strictly positive, the combination is strictly positive in its domain of definition. This property has been exploited by Clauss et. al. [5] for solving problems similar to those in Sect. 4.1. The method has been implemented in the Integer Set Library [5]. At first glance, it seems possible to use similar techniques for scheduling in non-parametric cases. Extending this approach to the parametric case seems quite complex.

Both the computations of Ehrhart polynomials and of Bernstein coefficients need the vertices of the enclosing polyhedron. When the problem has parameters, these vertices may move when parameters change, or even appear or disappear for critical values of the parameters. These phenomena are usually handled by computing *chambers* [6, Sect. 3], subsets of the space of parameters where the enclosing polyhedron does not change its shape. Then, a different solution is computed in each chamber. Since vertices are not used when solving problems by a recourse to Farkas lemma, this refinement has not been used in past research. Whether it will be useful for Handelman or Schweighofer solutions, and whether it is possible to find chambers for semi-algebraic sets are two open problems.

Nearest to the present proposal is the work of Armin Größlinger (see [11] and its references). The aim is similar: allow polynomials in program models, and extend algorithms for dependence testing, scheduling, tiling and code generation to the polynomial case. The main tool is quantifier elimination in the reals by either Cylindrical Algebraic Decomposition (CAD) or Weisspfenning test-points method. While implementing the present solution is simpler than, e.g. CAD, both approaches are plagued by a very high complexity. Time will tell which approach works best on real-life programs.

6. CONCLUSION AND FUTURE WORK

These remarks were intended to show that using polynomials and semi-algebraic sets in program analysis and optimization is feasible and might be useful. One may wonder if polynomials are so frequent as to warrant the trouble. Premature experiments may be biased, as first, programmers were taught to avoid non linear terms, and second, that tools like Polly or Clint ignore pieces of code that contain polynomial terms. Furthermore, polynomials may occur im-

[5]`isl.gforge.inria.fr`

plicitly as the result of an enabling analysis, as for instance solving a recurrence or numbering channel messages.

However, the present proposition raises many questions, the most urgent probably being its complexity. The number of products in the representation (2) increases roughly exponentially with the order. In my pilot implementation, it is difficult to go beyond order 3. It may be that, by using more powerful solvers like CPLEX or Gurobi, this limit might be pushed up by two or three units. However, experimental evidence shows that in practical problems, an overwhelming proportion of products are useless (turn out to have a zero λ multiplier). The first example of section 4.2 at order 2 uses more than a hundred products, only 2 of which are useful. It would be interesting to find heuristics for predicting which products are actually going to occur in the solution. In all the above examples, the running time of algorithm H is negligible; failures are due to memory overflow.

Handelman and Schweighofer theorems are true for the reals and the rationals, but, as far as I know, have no integer extensions. In contrast, most variables in program analysis problems are integral. In many cases, it has been found that ignoring the integrality constraints, or checking them by simple tests like the gcd test, gives acceptable conservative approximations. However, there are exceptions, like dataflow analysis or array shrinking. How to handle this difficulty is left for future work.

While in some cases just solving a polynomial problem is enough, as when testing for dependences or disproving the existence of deadlocks, most often the solution is the input to another step in compilation. For instance, how to generate a program from a polynomial schedule? Is an equivalent of CLooG possible for polynomials?

As a last remark, many other questions may benefit from the use of polynomials. Possible other uses are scheduling under resource constraints, or the construction of polynomial invariants.

Let the power of polynomials be with you.

7. REFERENCES

[1] W. Achtziger and K.-H. Zimmermann. Finding quadratic schedules for affine recurrence equations via nonsmooth optimization. *J. of VLSI Signal Processing*, 25:235–260, 2000.

[2] C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination and complexity bounds of flowchart programs. In *Static Analysis Symposium, SAS 2010*, pages 117–133, Perpignan, Sept. 2010. LNCS 6337.

[3] S. Bardin, J. Leroux, and G. Point. Fast extended release. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 63–66. Springer, 2006.

[4] P. Clauss. Counting solutions to linear and nonlinear constraints throught Ehrhart polynomials: Application to analyze and transform scientific programs. In *ACM Int. Conf. on Supercomputing*, pages 278–285, 1996.

[5] P. Clauss, F. J. Fernandez, D. Garbervetzky, and S. Verdoolaege. Symbolic polynomial maximization over covex sets and its application to memory requirement estimation. *IEEE Trans. VLSI Systems*, 17(8):983–996, 2009.

[6] P. Clauss and V. Loechner. Parametric analysis of polyhedral iteration space. *VLSI Signal Processing*, 19(2):179–194, 1998.

[7] P. Feautrier. Some efficient solutions to the affine scheduling problem, I, one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348, Oct. 1992.

[8] P. Feautrier. Some efficient solutions to the affine scheduling problem, II, multidimensional time. *Int. J. of Parallel Programming*, 21(6):389–420, Dec. 1992.

[9] P. Feautrier. Scalable and structured scheduling. *Int. J. of Parallel Programming*, 34(5):459–487, May 2006.

[10] P. Feautrier. Approximating the transitive closure of a boolean-affine relation. In U. Bondhugula and V. Loechner, editors, *IMPACT 2012*, 2012.

[11] A. Größlinger. *The Challenge of Non-linear Parameters and Variables in Automatic Loop Parallelisation*. PhD thesis, University of Passau, Germany, 2009. `http://nbn-resolving.de/urn:nbn:de:bvb:739-opus-17893`.

[12] D. Handelman. Representing polynomials by positive linear functions on convex polyhedra. *Pacific Journal of Mathematics*, 132(1):35–63, 1988.

[13] D. Jovanovic and L. de Moura. Solving non-linear arithmetic. In *Automated Reasoning - 6th International Joint Conference*, pages 339–354. Springer, June 2012. LNCS 7364.

[14] J.-L. Krivine. Anneaux préordonnés. *J. Anal. Math.*, 12:307–326, 1964.

[15] W. Pugh and D. Wonnacott. Non-linear array dependence analysis. In *Languages, Compilers and Run-time Systems for Scalable Computers*, pages 1–14. Springer, 1996.

[16] M. Schweighofer. An algorithmic approach to Schmüdgen's positivstellensatz. *J. of Pure and Applied Algebra*, 166:307–319, 2002.

[17] R. A. van Engelen, J. Birch, Y. Shou, B. Walsh, and K. A. Gallivan. A unified framework for nonlinear dependence testing and symbolic analysis. In *Int. Conf. on Supercomputing*, St. Malo, France, July 2004.

[18] E. Zima. Simplification and optimization transformations of chains of recurrences. In *Proc. of the Int. Symp. on Symbolic and Algebraic Computing*. ACM, 1995.