Coho Solver: A Efficient and Robust Linear Program Solver for Projectagons

Chao Yan, Mark Greenstreet and Marius Laza

Abstract. We present a novel implementation of the Simplex algorithm for solving linear programs that arise when manipulating "projectagons". A projectagon is a compact representation of a high-dimensional geometric object by its projections onto two-dimensional subspaces. These two-dimensional projections lead to linear programs where each inequality constraint has one or two non-zero coefficients. In this paper, we breifly describe the use of projetagons in the verification of VLSI circuits. We then present an efficient and robust implementation of Simplex for the linear programs that arise from projectagons. We show how this solver can be used to implement projectagons and illustrate these ideas with a small example.

Keywords. Linear Programming, Reachability Analysis, Interval Computation, Formal Verification.

1. Introduction

We consider the solution of a class of linear programs that have a simple, geometrical interpretation. Consider the representation of a d-dimensional object by its projection onto a set of two-dimensional subspaces. This is a generalization of the classical "mechanical drawings" (without idioms for representing interior surfaces) for representing three-dimensional objects. We refer to such objects as *projectagons*. For example, figure 1 shows how a three-dimensional object (the "anvil") can be represented by its projection onto the xy, yz, and xz planes. The high dimensional object is the largest set of points that satisfies the constraints of each projection. The projection polygons can be non-convex; thus, non-convex, high-dimensional objects can be represented by projectagons.

This work was supported in part by grants from Intel, SUN Microsystems, and Canadian NSERC..

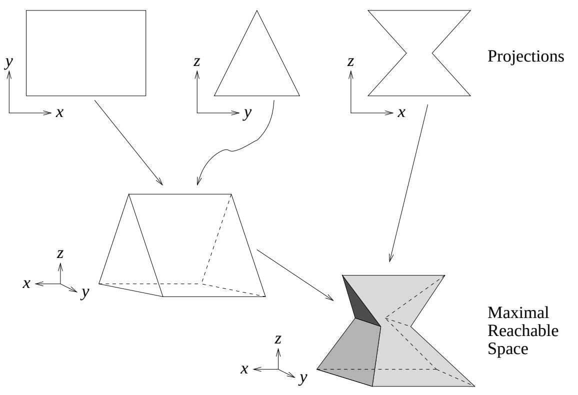

FIGURE 1. A three dimensional "projectagon"

Earlier [GM98, GM99], we proposed using projectagons for verifying digital circuits using detailed circuit models (i.e. non-linear, ordinary differential equations). Preliminary studies (e.g. [Gre96]) suggested that the two-dimensional projections could effectively capture the input/output relations used to reason about digital circuits. The verification methods proposed in [GM98, GM99] made extensive use of linear programming. When we tried to implement these methods, we found that the linear programs were often highly ill-conditioned, and standard methods for solving linear programs such as Simplex [Van01, chap. 2] and interior point methods [Meh92][NW99, chap. 14] failed.

This paper presents a novel implementation of the Simplex algorithm that is robust and efficient for solving linear programs that arise in the manipulation of projectagons. This algorithm exploits the special sparsity structure that arises from the projectagon representation. To put our algorithm in context, we give a brief description of the COHO verification tool and its use of projectagons in section 2. Section 3 describes the special structure of the linear programs for projectagons and how we exploit this structure in our implementation of Simplex. A key operation in COHO is computing the projection of the feasible region for a linear program onto a two-dimensional subspace; section 3.4 describes how we perform this projection operation. Section 4 presents a simple example of reachability analysis using COHO with our new linear program solver.

2. Coho

COHO [GM98, GM99] is a tool for reachability analysis based on projectagons. Our primary motivation for developing COHO is for circuit verification. Thus, this section briefly describes the circuit verification problem and then summarizes how COHO approaches this problem. The COHO algorithm involves solving a large number of linear programs. These linear programs are the motivation for the linear program solver described in the remainder of this paper.

2.1. Circuit Verification

Given a circuit with d nodes, we can define a function, $v : \mathbb{R}^+ \rightarrow R^d$, where the value of $v(t)_i$ is the voltage on node i at time t. Let $\mathsf{in}(t)$ be a function for the voltages applied to the inputs of the circuit. An ordinary differential equation that models the circuit has the form:

$$\tfrac{d}{dt}v \quad = \quad f(v, \mathsf{in}). \tag{1}$$

To verify that the circuit has correct digital behaviour, we need to show properties such as:

> *The data input to a flip-flop has settled to a value between $V_{0,\min}$ and $V_{0,\max}$ or to a value between $V_{1,\min}$ and $V_{1,\max}$ by $t_{set\text{-}up}$ before the rising edge of the clock and remains in that interval until at least time t_{hold} after the rising edge of the clock.*

These properties can be expressed as geometrical constraints on the values that v and in may take on (see [Gre98]). In particular, we specify the allowed behaviour of the inputs using geometric constraints [Bro89, GH97]. This allows us to model the circuit and its inputs with a differential inclusion [JPA84]:

$$\tfrac{d}{dt}(v, \mathsf{in}) \quad \in \quad F(v, \mathsf{in}). \tag{2}$$

Let d_{in} be the number of inputs to the circuit. A specification of the desired behaviour can be interpretted as describing a region, $Q \subseteq \mathbb{R}^{d+d_{in}}$. A circuit satisfies its specification if all points reachable by trajectories that satisfy the circuit's differential inclusion are contained in Q. Thus, the circuit verification problem becomes a problem of finding the reachable set for a differential inclusion.

2.2. Coho's Approach to Computing Reachability

Any method for computing reachability for differential inclusions must first address the issue that the reachability problem is undecidable [AM95, HKPV95]. COHO addresses this challenge by *overapproximating* the reachable region. Thus, COHO may reject a circuit that operates correctly, but it will not incorrectly verify a flawed design. The second issue that must be addressed is how to represent reachable regions of $\mathbb{R}^d$ and how to compute their evolution over time. COHO uses projectagons to represent reachable regions, and we describe how to compute the evolution of projectagons in the remainder of this section.

The key observation is that the faces of an object represented by a projectagon correspond to edges of its projection polygons. Assuming that the model has derivatives that are bounded, the extremal trajectories are those emanating from faces. The projectagon representation allows many operations on faces including calculations of their trajectories to be carried out as simple operations on polygon edges. Given an object represented by a projectagon, we must determine how this object evolves according to the model for the system. COHO does this by considering each face of the projectagon in turn. Because faces correspond to edges of the projection polygons, this involves considering each edge of each projection polygon. COHO moves each edge/face forward in time with an overapproximation obtained by a simple Euler integrator. Although the Euler integrator is less accurate than higher-order methods, its simplicity allows us to guarantee that our results are overapproximations as desired.

COHO advances the reachable region by performing the following operations at each time step (see [GM99] for details):

1. **Bound the convex hull of the current projectaton.**
   The projectagon obtained from the convex hulls of each of the projection polygons contains the convex hull of the original projectagon. COHO bloats this hull by a small amount to account for outward flows during the current time step. This hull is represented by a linear program. We'll call this bloated hull H.
2. **Derive a linear program for the neighbourhood of each face.**
   Recall that for each face, their is a corresponding edge of a projection polygon. Let e be such an edge. COHO constructs a rectangle, E with edges parallel or perpendicular to e, with the same bloating distance as used when forming H. The intersection of E and H is a linear program for a slab containing the current face. We'll call this linear program G.
3. **Linearize the model in this neighbourhood.**
   The model uses the linear program G to compute a matrix A, and vectors b and u such that
   $$F(x,t) \quad \subseteq \quad A \cdot x + b + \mathsf{cube}(u), \tag{3}$$
   where $\mathsf{cube}(u) = \{z \mid \forall i.\ |z_i| \leq u_i\}$. The $\mathsf{cube}(u)$ term allows a nonlinear relation to be approximated by a linear one plus an error term.
4. **Determine the step size.**
   Given the linearized models for each face and the bloating distance used when forming G, COHO can determine a time step such that all trajectories starting on faces of the original projectagon are guaranteed to stay inside their respective slabs (i.e. the feasible region of G for that face).
5. **Compute a time-advanced the face.**
   Given a linear ODE model (from step 3) and a time step (from step 4), it is straightforward to construct linear time forward and time backward operators. It is also straightforward to derive a linear program for the convex hull of the unbloated face. COHO applies the time forward operator to the linear program for the convex hull of the face to obtain the convex hull for the face at the end of the time step. the slab around the face, just omitting the bloat. Let F be the linear program for this time advanced face.
6. **Project the time advance face back to the original two-dimensional subspace.**
   The feasible set for F is a convex polyhedron, a subset of the state space $\mathbb{R}^{d+d_{in}}$. As described in section 3.4, COHO computes the projection of this polyhedron onto the two-dimensional subspace for the original polygon.

7. **Construct the time-advanced projection polygons.**
When the time-forward images of all of the edges of the polygon have been computed, COHO takes their union to determine the boundary of the polygon at the end of the time step. Typically, this increases the number of edges in the polygon. Thus, COHO performs overapproximating simplifications to keep the number of edges in the polygon tractable. Conversely, COHO automatically breaks long edges into shorter segments to avoid excessive errors in the linearization step.

As seen from this short description, COHO makes extensive use of linear programs when computing reachable sets for systems modeled by non-linear ODEs. When we first implemented COHO [GM99], we found that these linear programs are often highly ill-conditioned. While we were able to analyse a few simple systems, this ill-conditioning prevented further application of the approach. More recently, we noted that the linear programs arising in COHO have a special structure that could be used to obtain efficient and robust solution. This paper reports on our successful deployment of this approach.

3. Linear Programs in Coho

The linear programs in COHO express a conjunction of the half-spaces corresponding to the projection polygon edges. These linear programs can be written as

$$\begin{array}{l} \min\limits_{x} d^T \cdot x, \text{ s.t.} \\ \quad P \cdot E^{-1} \cdot x \quad \leq \quad c \end{array} \tag{4}$$

where $P \cdot x \leq c$ are the constraints such that x is a point of the projectahedron, d is the cost vector, and E is the forward time step operator for linearized model. Rows of P correspond to projection polygon edges; therefore, each row has either one or two non-zero elements. Because the time steps are relatively small, E is well-conditioned, and E^{-1} is easily computed. We refer to a linear program in the form from equation 4 as a "COHO linear program".

The dual [PS82] of a COHO LP is a standard form linear program. We write this program as

$$\begin{array}{lrcl} \min\limits_{u} & -c^T \cdot E \cdot u, \text{ s.t.} & & \\ & P^T \cdot u & = & d \\ & u & \geq & 0 \end{array} \tag{5}$$

Such a linear program can be solved using the Simplex algorithm. The Simplex algorithm repeatedly selects a subset of the columns of P called the "basis". Let $P_{\mathcal{B}}^T$ denote P^T restricted to this basis. Simplex solves for $u_{\mathcal{B}} = P_{\mathcal{B}}^{-T} \cdot d$ and determines if the basis can be modified so as to improve the cost. When used in COHO, a direct implementation of Simplex frequently encounters ill-conditioned bases and fails.

We attempted to use interior point algorithms [NW99, Meh92]. However, the inherent ill-conditioning of interior point methods compounded the problem of badly conditioned linear programs and prevents their successful application. Instead, we modified the Simplex algorithm to be robust with guaranteed error bounds. Briefly,

1. **Our method uses interval arithmetic.**
   While this does not improve the conditioning of the linear programs, it allows badly conditioned bases to be detected and handled safely. Interval arithmetic also provides guaranteed upper bounds on the error of the computed results.
2. **We developed a specialized linear system solver.**
   We present a linear time linear system solver for our problems. This allows the Simplex tableau to be created lazily after each pivot from the original coefficients of P thereby avoiding the error accumulation that is typically the bane of interval arithmetic.
3. **We modified the pivot selection rules.**
   Interval comparisons can produce "ambiguous" results such as "might be less than". We modified the rules for pivot selection and identifying optimal bases to be sound with interval comparisons.
4. **We implemented a specialized method for projecting the feasible polyhedron onto a two-dimensional subspace.**

The following subsections describe each of these in further detail.

3.1. Interval Arithmetic

To implement a proof-of-concept version of our algorithm, we wrote a simple, interval arithmetic package in Java. Our implementation uses the property that the IEEE floating point standard requires rounding to the nearest representable number. Thus, the double precision result is within $\frac{1}{2}$ulp (unit of least precision) of the exact value. Interval are combined in arithmetic operations in the obvious way.

3.2. A Specialized Linear System Solver

Traditional implementations of Simplex exploit the property that each basis is a rank-1 update of its predecessor. This allows the updated tableau to be computed in $O(n(n-m))$ time where n is the number of constraints in the standard form LP, and m is the number of variables. However, this approach accumulates error from one step to the next. Using interval arithmetic, the error bounds would quickly become larger than the values themselves. Instead, we exploit the structure of the P matrix to obtain a linear time algorithm for solving the linear systems that arise in the Simplex algorithm. This linear time algorithm for linear systems arising from projectahedra is the main contribution of this paper.

As noted earlier, each row of P corresponds to an edge of a projection polygon and thus has either one or two non-zero elements. Accordingly, each column of matrix $P_{\mathcal{B}}^T$ has either one or two non-zero elements. Now, consider the rows of $P_{\mathcal{B}}^T$. By the pivot selection rules, $P_{\mathcal{B}}^T$ will never have a row that is all zeros. If a row has exactly one non-zero element, then the value of the corresponding variable can be determined directly and the row and column are eliminated. Such elimination can be continued until every row has at least two non-zero elements. These eliminations preserve the property that every column has at most two non-zero elements, and the matrix remains square. Thus, at the end of this phase, every row and every column of the matrix has exactly two non-zero elements. This matrix has many wonderful properties; so, we will call this matrix W.

Assume that W is $n \times n$. Consider an undirected graph with vertices $v_1 \dots v_n$ such that there is an edge between vertices i and j iff W has a column that has non-zero elements in rows i and j. By the structure of W, every vertex in this graph has degree 2, and the graph can be decomposed into disjoint, simple cycles. The linear systems corresponding to these cycles can be solved independently. For simplicity of presentation, we will assume that W has a single cycle and note that the generalization to multiple cycles is straightforward.

We can now permute the rows and columns of W such that $W_{i,j} \neq 0$ iff $j = i$ or $j = (i \bmod n) + 1$. We can scale the rows so that every diagonal element of the matrix is 1. The resulting matrix has the form shown below:

$$W = \begin{bmatrix} 1 & -\alpha_1 & 0 & \dots & 0 \\ 0 & 1 & -\alpha_2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1 & -\alpha_{n-1} \\ -\alpha_n & 0 & \dots & 0 & 1 \end{bmatrix}. \tag{6}$$

Now, to solve a linear system of the form $Wu = d$, we observe that

$$u_1 = \frac{\sum_{j=1}^{n}(d_j \beta_{j-1})}{1 - \beta_n} \tag{7}$$

$$u_{i+1} = \frac{1 - \sum_{j=1}^{i}(d_j \beta_{j-1})}{\beta_i} \tag{8}$$

where $\beta_k = \prod_{i=1}^{k} \alpha_i$.

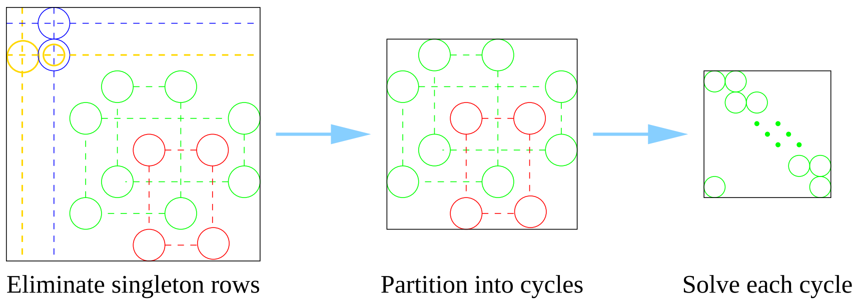

FIGURE 2. COHO's Linear System Solver

Figure 2 summarizes the flow of our linear system solver. It is straightforward to show that all of the steps described above for solving a linear system can be performed in $O(n)$ time where n is the number of equality constraints in the standard form LP. Furthermore, the only step that introduces cancellation is the denominator $1 - \beta_n$ in equation 7. Thus, we can rapidly identify ill-conditioned linear systems.

In the event that basis is poorly conditioned, we can use equation 7 to solve each variable separately. This results in an $O(n^2)$ algorithm but avoids accumulation of error. In our implementation, we use the linear time method first, and if the error exceeds a preset bound, solve again with the more precise $O(n^2)$ algorithm. In practice, the $O(n)$ algorithm is used almost all of the time. Thus we obtain high accuracy and low run-time.

In some cases, the basis is too badly conditioned for the algorithm to be able to meaningfully identify the next pivot to take. Note that changing any column in the ill-conditioned cycle changes the value of β_n and can remedy the problem. This situation is quite rare; so, we simply discard one of the columns from the standard form linear program and restart over. This may result in a sub-optimal solution to the standard form LP and therefore a super-optimal solution to COHO-LP. This preserves our guarantee of overapproximation (as described in section 2.2). To avoid excessive growth in the reachable region, we test the discarded constraint at the end of the LP solution. If it is clearly violated, then we bring that column back into the LP and reject a different one. We continue in this fashion until we find a good quality optimum.

3.3. Pivot Selection

The Simplex algorithm works by finding a feasible basis (one where all variables in equation 5 are positive) and pivoting to bases that progressively lower the cost until the optimal basis is determined. These pivots are determined from inequalities involving the results of solving linear systems as described above. Using interval arithmetic, the outcome of such comparisons can be ambiguous: if the intervals for x and y overlap, then the ordering of x and y cannot be determined. Thus, bases can be encountered that *might* be feasible and/or optimal, and pivots can be considered that *might* be favorable.

When selecting a pivot, our algorithm first checks to see if there is any pivot that is clearly favorable. This involves checking each column that is outside of the current basis until a clearly favorable pivot is found. A pivot is "clearly favorable" if the interval for the incremental cost resulting from the pivot is strictly negative. Each such check requires solving one linear system. Thus, the total time is $O(n(m-n))$ where n is the number of constraints and m is the number of variables. If a clearly favorable pivot is found, it is taken, and the other columns don't need to be evaluated at that step. If no clearly favorable pivot is found, then possibly favorable pivots are considered. A pivot is "possibly favorable" if the lower bound for its incremental cost interval is negative. At this point, our algorithm branches: all possibly favorable pivots are explored.

This branching pivoting precludes the use of standard anti-cycling algorithms [Bla77]. Instead, we maintain a hashtable of all bases that we have visited.

If a pivot leads to a basis that we've already seen, it is ignored. By branching, COHO is guaranteed to visit the truly optimal basis.

Similar issues arise when testing bases for optimality. Mathematically, a basis is optimal if and only if it is feasible for both the standard- and COHO-form LPs. If COHO encounters a clearly optimal basis, it is done. Otherwise, each time COHO encounters a basis that is possibly optimal, it checks to see if the corresponding basis for the COHO-form LP is possibly feasible. If so, the basis is flagged as possibly optimal. If no clearly optimal basis is found, COHO uses the set of possibly optimal bases to determine an interval that it guaranteed to contain the actual cost at optimality. We designed the rest of COHO to only depend on obtaining valid bounds for the cost and not to require the optimal vertex itself.

3.4. Computing Projections

As described in section 2.2 (step 6), a frequent operation is to find the projection of the polyhedron for a linear program onto a two-dimensional subspace. Let x and y be the basis vectors for such a subspace. COHO's projection algorithm starts by finding the vertex of the polyhedron for which the x component is maximized. Clearly, this vertex is a vertex of the projection polygon as well. Then, COHO rotates the optimization direction until this vertex is no longer optimal. This rotation can be determined by solving two linear systems using the solver described above. The linear program can be solved for this new search direction to obtain the next vertex of the projection. Furthermore, at the critical value where the current vertex becomes non-optimal, the search direction is orthogonal to an edge of the the projection polygon from the previous optimal vertex to the new optimum. Thus, the projection can be determined by solving a sequence of linear programs.

4. An Example

In the section we present a simple example to demonstrate a reachability calculation using Coho. Our model is a Van der Pol oscillator [HS74, pp. 217f] where we've added the z variable to provide three dimensions and use Coho's projection capability. The model is:

$$\begin{array}{rcl} \dot{x} & = & -y - x^3 + x \\ \dot{y} & = & x - y^3 + y \\ \dot{z} & = & 2x^2 - 2z \end{array} \tag{9}$$

The initial region is the hypercube with the diagonal

$$[(1.0, -0.05, 0.9),\ (1.2, 0.05, 1.1)] \tag{10}$$

We used Coho with projections onto the x-y and x-z planes to find an invariant set for this model. Figures 3 and 4 show the bounds that Coho computes for this example. Note that in both projections, the region at the end is contained in the initial region. This establishes the invariance of the computed region.

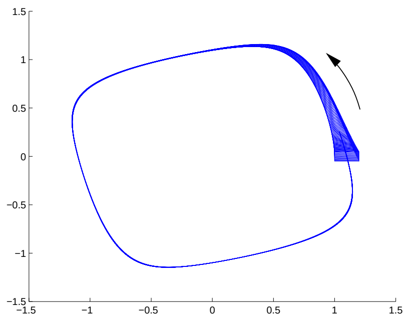

FIGURE 3. x-y projection of the three-dimensional Van der Pol Oscillator

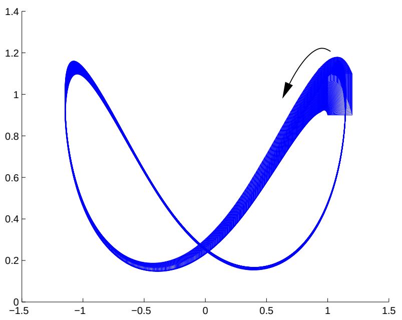

FIGURE 4. x-z projection of the three-dimensional Van der Pol Oscillator

5. Conclusions and Future Work

Reachability plays a central role in analysing and verifying complex systems modeled by differential equations. For realistic models, exact solutions to reachability problems are not computable and approximations must be used. If the approximation is guaranteed to be an overapproximation, then the method is sound for verifying safety properties. Efficient and robust optimization methods play a central role in computing such approximations.

This paper presented an efficient and robust linear programming solver and demonstrated its use for reachability analysis. This solver uses interval arithmetic to provide guaranteed error bounds and a novel linear-time, linear system solver that exploits the structure of linear programs corresponding to the "projectagons" that we use to represent high-dimensional geometrical objects. We demonstrated the use of this approach on a simple non-linear example: a three-dimensional Van der Pol oscillator.

Obviously, this is just the start. Based on our experience with a few small examples, we believe that we have solved the problems of ill-conditioning that have hindered the use of projection based methods for reachability calculations in the past. Our top priority is to use our approach on more examples, especially non-linear circuits for high-speed digital applications. We are also interested in example from hybrid systems and control as well as models of biological processes.

We believe that our linear program solver has the attractive property that the maximum error in the cost is bounded by the diameter of the reachable region times the square root of the machine precision. This conjecture is supported by our numerical data, but we have not at this point completed a proof.

The use of interval arithmetic in our methods has the advantage that it is supported by the IEEE floating point standard and therefore by the floating point hardware units on most computers. Arbitrary precision rational arithmetic provides another, and perhaps complementary way to solve ill-conditioned linear problems reliably. In particular, one could use our interval methods to identify a small set of possibly optimal bases, and then use arbitrary precision rational arithmetic to select the truly optimal one from this set. This is an area for future investigation.

While we have developed our linear program solver specifically for Coho, we note that efficient representation of high-dimensional geometric objects is important in many domains. The critical feature of our algorithm is that it exploits structure arising from the use of two-dimensional projections. Thus, it is likely that our algorithm or ideas within it could be used for other geometrical problems that are constructed from two-dimensional subproblems.

Projectagons provide an attractive way to represent high dimensional objects. Projectagons can represent non-convex objects and operations on projectagons are efficient in terms of both time and space. Exploring the issues of projectagon geometry is another area for future work. For example, the intersection of two or more projectagons is obtained by the intersection of their projection polygons. On the other hand, projectagons are closed under neither complement nor union, and we don't know how to calculate the smallest overapproximation of the union. Such operations would be useful for implementing frontier based methods for reachability that could offer significant improvements of efficiency. Likewise, we do not know of an efficient algorithm for determining whether or not a projectagon is empty.

References

[AM95] Eugene Asarin and Oded Maler. On the analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138:35–65, 1995.

[Bla77] R.G. Bland. New finite pivoting rules for the simplex method. *Math. Oper. Res.*, 2:103–107, 1977.

[Bro89] R.W. Brockett. Smooth dynamical systems which realize arithmetical and logical operations. In Hendrik Nijmeijer and Johannes M. Schumacher, editors, *Three Decades of Mathematical Systems Theory: A Collection of Surveys at the Occasion of the 50th Birthday of J. C. Willems*, volume 135 of *Lecture Notes in Control and Information Sciences*, pages 19–30. Springer, 1989.

[GH97] Mark R. Greenstreet and Xuemei Huang. A smooth dynamical system that counts in binary. In *Proceedings of the 1997 International Symposium on Circuits and Systems*, volume II, pages 977–980, Hong Kong, June 1997. IEEE.

[GM98] Mark R. Greenstreet and Ian Mitchell. Integrating projections. In Thomas A. Henzinger and Shankar Sastry, editors, *Proceedings of the First International Workshop on Hybrid Systems: Computation and Control*, pages 159–174, Berkeley, California, April 1998.

[GM99] Mark R. Greenstreet and Ian Mitchell. Reachability analysis using polygonal projections. In *Proceedings of the Second International Workshop on Hybrid Systems: Computation and Control*, pages 103–116, Berg en Dal, The Netherlands, March 1999. Springer. LNCS 1569.

[Gre96] Mark R. Greenstreet. Verifying safety properties of differential equations. In *Proceedings of the 1996 Conference on Computer Aided Verification*, pages 277–287, New Brunswick, NJ, July 1996.

[Gre98] Mark R. Greenstreet. A hybrid-systems view of discrete computation. In *Proceedings of the* 37^{th} *IEEE Conference on Decision and Control*, pages 2107–2112, Tampa, Florida, December 1998.

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[HS74] Morris W. Hirsch and Stephen Smale. *Differential Equations, Dynamical Systems, and Linear Algebra.* Academic Press, San Diego, CA, 1974.

[JPA84] Arrigo Cellina Jean-Pierre Aubin. *Differential Inclusions: Set-Valued Maps and Viability Theory.* Springer, 1984.

[Meh92] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 1992.

[NW99] J. Nocedal and S. Wright. *Numerical Optimization*, pages 395–417. Springer Series in Operations Research, Springer Press, 1999.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[Van01] Robert$\tilde{\text{J}}$. Vanderbei. *Linear Programming: Foundations and Extensions.* Springer, second edition, 2001.

Chao Yan
2366 Main Mall
Vancouver,BC,V6T 1Z4
e-mail: chaoyan@cs.ubc.ca

Mark Greenstreet
2366 Main Mall
Vancouver,BC,V6T 1Z4
e-mail: mrg@cs.ubc.ca

Marius Laza
2366 Main Mall
Vancouver,BC,V6T 1Z4
e-mail: mariusl@schemasoft.com