Interval Analysis, Constraint Propagation and Applications

Christophe Jermann, Yahia Lebbah, Djamila Sam-Haroud

SIXTH PART

Interval analysis, constraint propagation, applications

Chapter 1

Interval analysis, constraint propagation, applications

Foreword

This part of the book gathers a selection of papers from the third international workshop on INTerval analysis, Constraint Propagation, and applications (IntCP 2006) held at Cité des Congrès, Nantes (France) on September 25th, 2006.

The two most appealing features of interval analysis and numerical constraint propagation, when used to solve numerical problems, are completeness and rigor. Completeness means the ability to find all solutions, whereas rigor is the ability to control the rounding errors due to floating-point computation. Completeness and rigor are essential in numerous applications such as engineering design, robotics, control, logistics, manufacturing, chemical and biological sciences and computer-aided design to name a few. The problems in these domains involve equations, inequalities, differential equations and sometimes an objective function on variables taking their values in the set of real numbers.

The multidisciplinary nature of constraint propagation and the general framework offered by constraint programming give rise to a unique combination of theoretical and experimental research providing bridges between separate, but often complementary areas. However, while interval based constraint propagation solvers have proved particularly efficient in solving challenging instances of nonlinear numerical problems, they do not yet have enough appeal in many practical areas. One of the reasons

---

Chapter written by Christophe JERMANN and Yahia LEBBAH and Djamila SAM-HAROUD.

is that they generally provide representation of the solution set that are either prohibitively verbose or poorly informative. Another reason is that they are sometimes too inefficient, especially to address real-time applications like interactive control or animation. Recent advances have shown that these limitations are not intrinsic since constraint propagation can be considerably improved using techniques from interval analysis and local/global optimization.

The goal of IntCP 2006 was to emphasize the multidisciplinary nature of these researches and reach out a number of communities which are increasingly interested in interval analysis and constraint propagation. Nine papers were accepted for presentation during the workshop (http://liawww.epfl.ch/Events/IntCP2006). Their content reflects the trends both towards the combination of techniques from different areas, and towards more demanding real life applications: nonlinear differential equations, advanced relaxations for global optimization, interval disjunction for continuous shaving based reduction and for search, continuous MaxCSP, reduction methods for robotics, and hybrid systems. Our selection contains three representative papers chosen from these nine presentations. The first paper on radio antennas deployment, introduces an interesting approach to model and solve a numerical sub-problem within a constraint programming framework. The second paper shows that numerical injectivity can be handled rigorously via interval analysis. The third paper presents an interval constraint propagation framework to handle hybrid systems where occur both discrete and continuous changes.

We hope that this workshop has helped in growing the maturity of interval analysis and constraint propagation. We would like to thank the program and referee committees who worked under tight deadlines.

Christophe Jermann,
Yahia Lebbah,
Djamila Sam-Haroud

Members of the program committee and referees

P. Barahona	D. Daney	S. Ratschan
M. Ceberio	L. Jaulin	N. Sahinidis
M. Christie	C. Jermann	D. Sam-Haroud
J. Cruz	Y. Lebbah	J. Vehi

Chapter 2

Modeling and solving of a radio antennas deployment support application

2.1. Introduction

The new theatre of operations requires an ever growing number of telecommunication systems to share a limited frequency spectrum. As a consequence, the task of assigning frequencies becomes more and more difficult for operators taking care of the deployment of mobile units.

Deployment support of radio antennas is a crossbreeding between a classical radio link frequency assignment problem [AAR 03] (RLFAP) and a location analysis problem. In a classical RLFAP, the position of all antennas is known beforehand, and determines statically the set of constraints that apply on frequencies used for the communication links. This contrasts with our problem that we call LocRLFAP (for location and RLFAP), where the links to be established between different antennas is fixed, but the position of some antennas has to be found in a way that optimizes the overall frequency usage while respecting all physical constraints.

Outline of the chapter: Section 2.2 gives two finite and mixed continuous domain models of the application, and compares their performance. Section 2.3 introduces a new continuous Euclidean distance global constraint. Section 2.4 presents how this constraint can fit into the applications' model and what computational and qualitative improvements this global constraint enables.

Chapter written by MichaelHEUSCH.

2.2. Two simple models for the application

After having introduced a simple discrete model of our problem, we motivate why we wish to shift it to mixed domains. We then describe the search algorithm we use and analyse its performance on our experiments.

2.2.1. *A first finite domain model*

Call $S_0, \ldots, S_n$ the set of sites that host antennas and f_{ij} the discrete frequency used to establish a link from S_i to S_j. For two sites S_i and S_j positioned at (x_i, y_i) and (x_j, y_j), the square of the Euclidean distance between them is given by $dist(S_i, S_j) = (x_i - x_j)^2 + (y_i - y_j)^2$.

The set of available frequencies is discrete and discontinuous. Discrete constraints applied solely to frequencies are the following.

– On each site, the frequency used by a transmitter must be at an absolute value distance from the frequencies of the receivers:

$$\forall (i,j,k) \ / \ i \neq j, i \neq k, \quad |f_{ij} - f_{ki}| > \Delta_{tr}$$

– An absolute value constraint applies on all bidirectional links:

$$\forall (i,j) \in E_d, \quad |f_{ij} - f_{ji}| = \Delta_b$$

We now list all Euclidean distance based constraints.

– For security and interference reasons, a minimum distance must be enforced between any two sites:

$$\forall (i,j) \ / \ i \neq j, \quad dist(S_i, S_j) \geq d_m \tag{2.1}$$

– If there is a link from S_i to S_j, reachability of the link implies a maximum distance constraint:

$$\forall (i,j) \ / \ i \neq j, \quad dist(S_i, S_j) \leq d_M \tag{2.2}$$

– Two levels of compatibility are imposed between links established from remote or near sites $\forall (i,j,k,l)$, $i \neq j$, $i \neq k$, $j \neq l$:

$$dist(S_i, S_j) \geq d_l \ \vee \ |f_{ik} - f_{lj}| > \Delta_l \tag{2.3}$$

$$dist(S_i, S_j) \geq d_L \ \vee \ |f_{ik} - f_{lj}| > \Delta_L \tag{2.4}$$

The purpose of the optimization problem is to minimize the maximal frequency used by all links while respecting all operational constraints.

2.2.2. *Shifting the model to mixed domains*

Eclair [LAB 98], the finite domains constraint solver we use to tackle this problem, allows us to model all these discrete constraints with constructs that ensure equivalence with the mathematical model. Euclidean distance constraints can be discretized and decomposed using square and linear term constraints. Position variables are given by domain bounds, and the filtering algorithms that deal with the absolute value constraints take advantage of enumerated domain variables.

However, as Euclidean distance constraints are fundamentally continuous, it is interesting to see the advantage of reformulating this problem into a mixed model. When doing this with interval constraints, the mathematical model can be preserved and all that changes is the consistency algorithm used to solve the constraints on continuous variables. We achieve this with a generic HC4[BEN 99] local consistency algorithm to propagate Euclidean distance constraints, whereas the other constraints remain handled by the discrete constraint solver. The only mixed discrete-continuous constraints of our model are disjunctions as in [2.4]. Both model components (discrete and continuous) obey a fixed point semantics which is governed by synchronization of these disjunctions.

2.2.3. *Description of the search algorithm*

After testing chronological order, minimum domain variable selection, and the heuristics proposed in [BES 96] we developed a new strategy. For frequency variables we choose the variable with smallest domain and the greatest number of attached constraints (in case of ties); for postitions we choose the variable with the largest domain first. This heuristic proved to perform the best on our examples. Also, as our model is mostly under-constrained on positions, one can slightly improve the search algorithm when exploring the positions subtree by using a heuristic introduced by Gelle [GEL 03]. This value selection heuristic refines the dichotomic search principle we used previously. In addition to splitting each domain in two parts at each choice point, it first explores the value at the middle of the domain. By orienting the search towards a point at the center of the box, one hopes to select a region where one is most likely to find a solution.

2.2.4. *Analysis of the performance on progressive deployment problems*

We compared both models performance on a set of six test instances[1] that represent networks with between 5 and 10 sites [2]. One familiar with the RLFAP shall notice that the LocRLFAP is much more difficult to solve: not only does it contain a great number of disjunctions, but the number of constraints also grows much faster for a same number of sites.

We consider scenarios where part of the sites locations is relaxed. We did not study all 2^n possible cases but rather a chronological progressive relaxation from the problem where all sites are fixed. We first loosen up S_0, then S_0 and S_1, etc. up to the case where all sites are freed. When sites 0 to i out of 10 sites are relaxed we call the scenario $L_{10}^{0..i}$. We give the computational results of the instances with 9 and 10 sites on Tables 2.1 and 2.2. Remarkably, the difficulty to solve the LocRLFAP

	discrete model		hybrid model	
Instance	Time$_{ms}$	Fails	Time$_{ms}$	Fails
$L_9^{\emptyset}$	3120	9192	5430	9196
L_9^{0}	830	3270	1890	3265
$L_9^{0..1}$	930	3319	1990	3306
$L_9^{0..2}$	1410	6754	3880	6725
$L_9^{0..3}$	1420	6768	3870	6733
$L_9^{0..4}$	430	655	760	644
$L_9^{0..5}$	440	646	750	634
$L_9^{0..6}$	140	370	360	422
$L_9^{0..7}$	70	140	285050	75737
$L_9^{0..8}$	90	148	240	168

Table 2.1. *Analysis of the solving of the instance with 9 sites*

varies strongly and non-monotonously according to the number of relaxed site positions. On the example with 9 sites, one can observe differences of up to three orders of magnitude between the easiest and the most difficult instances. Except from $L_9^{0..7}$, the difficulty is generally analogous between the discrete and hybrid models. For this particular case, almost all backtracks (precisely 75146 out of 75737) take place in the continuous part of the search tree; this suggests that the local-consistency enforced by HC4 on the continuous distance constraints is not sufficient to efficiently solve the

1. For confidentiality reasons, we could not use real numerical data. However, Thales has provided us with a slightly simplified testbed that it considers realistic and representative of the LocRLFAP's difficulty.

2. All experimentations are performed on a 1.6Ghz Pentium M-735 laptop with 512Mb RAM.

	discrete model		hybrid model	
Instance	Time$_{ms}$	Fails	Time$_{ms}$	Fails
$L_{10}^{\emptyset}$	86.540	156.487	136.350	156.489
L_{10}^{0}	2.894.490	8.422.085	7.034.800	8.397.394
$L_{10}^{0..1}$	465.510	641.425	893.360	585.919
$L_{10}^{0..2}$	210.940	253.606	358.930	253.204
$L_{10}^{0..3}$	143.240	162.252	238.950	161.624
$L_{10}^{0..4}$	11.720	16.146	17.110	12.639
$L_{10}^{0..5}$	11.670	16.040	17.630	12.756
$L_{10}^{0..6}$	3.330	8.529	8.590	9.058
$L_{10}^{0..7}$	-	-	4.420	4.324
$L_{10}^{0..8}$	-	-	-	-
$L_{10}^{0..9}$	-	-	-	-

Table 2.2. *Analysis of the solving of the instance with 10 sites*

problem. In six instances out of ten, the hybrid model reaches the optimum with a few backtracks less, but the gap isn't sufficient to make the computations faster. This difference in the number of fails could be explained by the fact that in the discrete case, one sometimes gets near to a real-number points solution, but that one is brought to reject it due to the restriction to integer solutions. With an equivalent number of backtracks, the hybrid resolution is between one and three times slower. If one excludes the outlier $L_9^{0..7}$ and the instance where all sites are fixed[3], solving times are on average 2.4 times slower. These experimentations generally show us that a naive hybridization of our application's model is of little help, as it allows solving only one additional test case while it basically always hampers the computation times.

2.3. Introducing the *distn* constraint

Euclidean distance constraints are of fundamental incidence in the core of our problem formulation. A constraint that considers all constraints of the problem globally rather than considering each member of a clique distance constraints separately can therefore be advantageous to model our application more efficiently.

Euclidean Distance constraints appear in many application domains of CP, ranging from deployment problems to robotics, spatial databases and chemistry. 2B-consistency type methods [BEN 99] attain a poor filtering [BAT 05] on Euclidean distance constraints, and stronger consistency techniques [LHO 93, FAL 94] not applicable on

3. For $L_9^{\emptyset}$, the LocRLFAP has no real justification and one could as well choose a model without disjunctions.

problems involving many such constraint instances. Different means have therefore been considered [PES 99, LEB 02, KRI 02, BAT 05, HEU 03] to enhance the usability of constraint solvers. None of these is however adapted to variable distances.

We use a continuous n-ary global constraint proposed in [HEU 03, HEU 06] that considers the interdistance relations between n points. The syntax of this constraint is given by: $\texttt{distn}([\texttt{P}_1, \ldots, \texttt{P}_\texttt{n}], \texttt{V}_\texttt{i,j})$ where the $P_i = (X_i, Y_i)$ is the Cartesian product of domain variables, $V_{i,j}$ is a symmetric $n \times n$ matrix of nonnegative variables, constrained to be equal to the Euclidean distance between the P_i. Domains may be discrete or continuous. This constraint holds iff for all $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$ and $v_{i,j}$ we have $dist(p_i, p_j) = v_{i,j}$. To use this constraint to model plain (non variable) minimal distance constraints, one can simply use an upper bound equal to $+\infty$ for all domains of the $V_{i,j}$. Conversely, it suffices to set a lower bound equal to 0 for all domains of the $V_{i,j}$ to model solely maximal distance constraints.

The continuous filtering of the constraints is done by an extension to an algorithm developed to solve Circle Packing problems [MAR 05]. It defines forbidden regions with a geometric reasoning and approximates them by representing domains with polygons. This allows us to reinforce the consistency level by considering n constraints simultaneously. Also, all computations are made reliable by using interval arithmetic [NEU 90] extensively, thus no feasible point is lost. An iteration of the algorithm considers each polygon in turn and operates in polynomial time. For efficiency reasons, we don't use a fixed point semantics and the algorithm is stopped after a fixed number of iterations. When working with discrete variables, floating point polygons are reduced to integer bounds and the algorithm is restarted until a fixed point is reached. We refer the reader to [HEU 06] for more details.

2.4. Modeling the application with the *distn* constraint

We show how we can use *distn* to improve both our simple discrete and hybrid constraint models. We then analyze the numerical results of our solving of the LocRLFAP on instances with 9 and 10 sites and finally we examine what qualitative improvements this further provides.

2.4.1. *Revised model of the application*

One weakness of the previous model is that distance constraints involved in the disjunctions [2.1] and [2.1] are disconnected from those appearing in the clique of inter-distance constraints [2.4]. For all $i \neq j$, $i \neq k$, $j \neq l$ we have:

$$\mathtt{dist}(\mathtt{S_i}, \mathtt{S_j}) \geq \mathtt{d_L} \ \vee \ |\, \mathtt{f[i,k]} - \mathtt{f[l,j]} \,| > \Delta_\mathtt{L}$$
$$\mathtt{dist}(\mathtt{S_i}, \mathtt{S_j}) \geq \mathtt{d_l} \ \vee \ |\, \mathtt{f[i,k]} - \mathtt{f[l,j]} \,| > \Delta_\mathtt{l}$$
$$\mathtt{dist}(\mathtt{S_i}, \mathtt{S_j}) \geq \mathtt{m}$$
$$\mathtt{dist}(\mathtt{S_i}, \mathtt{S_j}) \leq \mathtt{M} \quad \text{if there is a link from } \mathtt{S_i} \text{ to } \mathtt{S_j}$$

Writing the whole deployment problem with a disconnected set of elementary Euclidean distance constraints doesn't manage to handle the domain's semantics. As a consequence, the filtering achieved on the variables giving the sites' positions is too low to enable an efficient solving of the given model. We address this problem by using the global constraint we introduced.

The *distn* constraint can be used for the application while maintaining a mathematical equivalence between the mathematical and the constraint model. It enables to give a tighter formulation to both the reachability and minimal interdistance constraints. Moreover, it enhances the integration of the constraint in the disjunctive nature of the model, as it allows us to link the inter-distance constraints with the "distant compatibility" constraints.

For each couple (i, j), $i \neq j$ let's introduce a variable $\mathtt{v[i,j]}$ that expresses the Euclidean distance between S_i and S_j. The minimum and maximum distance constraints of [2.1] and [2.2] can be expressed simply by setting the distance variable's domain to $\mathtt{Domain(v[i,j])} = [\, \mathtt{m_{i,j}}, \mathtt{M_{i,j}} \,]$ with for all $i \neq j$, $\mathtt{v[i,j]} = \mathtt{v[j,i]}$. It now remains to connect the distance constraints appearing in the hybrid disjunctions [2.4] to the global constraint. One can do this by gathering the whole set of distance constraints of the application and writing for all $i \neq j$, $i \neq k$, $j \neq l$:

$$\mathtt{v[i,j]} \geq \mathtt{d_l} \ \vee \ |\, \mathtt{f[i,k]} - \mathtt{f[l,j]} \,| > \Delta_\mathtt{l}$$
$$\mathtt{v[i,j]} \geq \mathtt{d_L} \ \vee \ |\, \mathtt{f[i,k]} - \mathtt{f[l,j]} \,| > \Delta_\mathtt{L}$$
$$\mathtt{distn}([\mathtt{S_1}, \ldots, \mathtt{S_n}] \,,\ \mathtt{v[i,j]})$$

This allows us to have a bidirectional communication between the distance constraints appearing in the distance constraints and those required by reachability and minimum distance constraints.

– If an absolute value constraint is violated, the new lower bound for the minimal inter-distance will directly influence the whole lot of distance relations in *distn*.

– If the whole set of distance constraints considered globally in *distn* doesn't allow instantiating the left branch of a disjunction, the corresponding absolute value constraint is immediately enforced.

2.4.2. *Numerical results when solving the LocRLFAP with distn*

The solving times and number of backtracks obtained for instances with 9 and 10 sites using this new model are given on Tables 2.3 and 2.4.

	discrete model		hybrid model	
Instance	Time$_{ms}$	Fails	Time$_{ms}$	Fails
$L_9^{\emptyset}$	3120	9192	5540	9196
L_9^{0}	1570	3284	2150	3278
$L_9^{0..1}$	1580	3403	1910	2744
$L_9^{0..2}$	1280	1851	2040	2096
$L_9^{0..3}$	1300	1763	2100	2022
$L_9^{0..4}$	2140	716	2180	698
$L_9^{0..5}$	2690	707	2550	702
$L_9^{0..6}$	780	266	530	389
$L_9^{0..7}$	410	117	640	215
$L_9^{0..8}$	690	117	830	237

Table 2.3. *Solving of the 9 sites instances with the discrete and hybrid models using distn.*

	discrete model		hybrid model	
Instance	Time$_{ms}$	Fails	Time$_{ms}$	Fails
$L_{10}^{\emptyset}$	85.620	156.487	138.370	156.489
L_{10}^{0}	351.020	852.318	150.170	87.144
$L_{10}^{0..1}$	124.620	115.307	123.340	73.451
$L_{10}^{0..2}$	239.700	147.809	325.530	163.440
$L_{10}^{0..3}$	237.950	131.582	337.970	147.716
$L_{10}^{0..4}$	28.310	16.830	28.420	13.758
$L_{10}^{0..5}$	29.960	16.698	29.660	13.760
$L_{10}^{0..6}$	11.360	8.052	3.210	1.931
$L_{10}^{0..7}$	-	-	6.310	4.264
$L_{10}^{0..8}$	-	-	218.220	100.298
$L_{10}^{0..9}$	-	-	369.300	140.592

Table 2.4. *Solving of the 10 sites instances with the discrete and hybrid models using distn.*

On the example with 9 sites, when comparing these results with those obtained with elementary constraints, one can notice that *distn* allows us to divide the number of backtracks by two on average for the nine deployment instances, both in the hybrid and discrete models, but the cost of calling the global constraint impedes that computations get 2 (resp. 1.2) slower on the discrete (resp. hybrid) models. One no longer observes the irregularity we had on $L_9^{0..7}$ and we get a speedup of two orders of magnitude on this instance, therefore when summing all results for hybrid instances, one divides the number of backtracks by 8 and gains a time factor of 20.

On the examples with 10 sites, when we compare the seven instances of deployment that we manage to solve in both models, the hybrid model requires 63% less

backtracks than the discrete model, but does solve only 3% faster. However, on three other instances, the hybrid model manages to solve the scenario in less than 10 minutes, whereas the discrete model doesn't manage to obtain even a feasible solution in more than one hour. Comparing again these last results with those obtained with elementary constraints, one observes that *distn* allows to divide the number of backtracks by 18.8 (resp. 7.4) in the hybrid case (resp. discrete) and to divide the computation times by 8.6 (resp. 3.7) on an average of seven instances.

2.4.3. *Qualitative analysis of the results*

Although it requires greater computational effort, with a qualitative point of view, relaxing the problem from the RLFAP model to the LocRLFAP one enables to save up to 63% of used frequencies. The most interesting conclusion on our new hybrid model is its twofold advantage:

– It allows us to obtain solutions saving frequencies compared to a discrete model: we get a result better by 15% on L_{10}^{0}. We can not claim anything for the three last instances that we did not manage to solve in a discrete space.

– We manage to solve the whole of our instances with this model while no solution is found in more than one hour in a discrete search space

2.5. Conclusion

We have defined a simple constraint model for a mobile radio antennas deployment support application and evaluated it on several test instances, both on discrete and on mixed finite and continuous domains. They enable to solve the smallest models satisfactorily but are unable to scale to the greater ones, even with an improved search algorithm. We have remodeled the application by proposing a novel continuous global constraint maintaining variable Euclidean distance constraints. In particular we have seen in how far this constraint addresses the disjunctive nature of the deployment problem. The association of a discrete-continuous hybrid search space with this advanced model and a new variable selection heuristic enable to solve our complete set of test problems. These results have more generally shown the interest that both continuous global constraints and the hybridization of finite domain and interval constraints can present to improve the solving performance of complex combinatorial problems where a complete discretization is customarily used. Moreover, our tests have highlighted some scenario where a modeling by hybrid constraints enables a qualitative advantage in the sense that better optima become reachable than in a fully discretized model.

Acknowledgements: This work has greatly benefited from discussions with Frédéric Benhamou, Frédéric Goualard and Juliette Mattioli.

2.6. Bibliography

[AAR 03] AARDAL K. I., VAN HOESEL C. P. M., KOSTER A. M. C. A., MANNINO C., SASSANO A., "Models and Solution Techniques for the Frequency Assignment Problem", *4OR*, vol. 1, num. 4, p. 261–317, 2003.

[BAT 05] BATNINI H., Contraintes Globales et Techniques de Résolution pour les CSPs Continus, PhD thesis, University of Nice Sophia-Antipolis, december 2005, (in french).

[BEN 99] BENHAMOU F., GOUALARD F., GRANVILLIERS L., PUGET J.-F., "Revising Hull and Box Consistency", *Proc. of ICLP-99*, 1999.

[BES 96] BESSIÈRE C., RÉGIN J.-C., "MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems", *Proc. of CP-96*, 1996.

[FAL 94] FALTINGS B., "Arc Consistency for Continuous Variables", *Artificial Intelligence*, vol. 65, num. 2, p. 363–376, 1994.

[GEL 03] GELLE E., FALTINGS B., "Solving Mixed and Conditional Constraint Satisfaction Problems", *Constraints*, vol. 8, num. 2, p. 107-141, Kluwer Academic Publishers, 2003.

[HEU 03] HEUSCH M., "distn: An Euclidean Distance Global Constraint", *Proc. of CP 2003*, Page975, 2003, Doctoral Program.

[HEU 06] HEUSCH M., Modeling and solving of a radio antennas deployment support application by constraint programming over finite and continuous domains, PhD thesis, University of Nantes, january 2006, (in french).

[KRI 02] KRIPPAHL L., BARAHONA P., "PSICO: Solving Protein Structures with Constraint Programming and Optimization", *Constraints*, vol. 7, num. 3-4, p. 317–331, Kluwer Academic Publishers, 2002.

[LAB 98] LABURTHE F., SAVÉANT P., DE GIVRY S., JOURDAN J., ECLAIR: A Library of Constraints over Finite Domains, Report num. ATS 98-2, Thomson-CSF LCR, Orsay, France, 1998.

[LEB 02] LEBBAH Y., RUEHER M., MICHEL C., "A Global Filtering Algorithm for Handling Systems of Quadratic Equations and Inequations", *Proc. of CP-02*, 2002.

[LHO 93] LHOMME O., "Consistency Techniques for Numeric CSPs", *Proc. of IJCAI-93*, p. 232-238, 1993.

[MAR 05] MARKÓT M. C., CSENDES T., "A New Verified Optimization Technique for the "Packing Circles in a Unit Square" Problems", *SIAM J. Optimization*, vol. 16, p. 193-219, 2005.

[NEU 90] NEUMAIER A., *Interval Methods for Systems of Equations*, Cambridge University Press, 1990.

[PES 99] PESANT G., BOYER M., "Reasoning about Solids Using Constraint Logic Programming", *J. Autom. Reason.*, vol. 22, num. 3, p. 241–262, Kluwer Academic Publishers, 1999.

Chapter 3

Guaranteed numerical injectivity test via interval analysis

3.1. Introduction

The purpose of this paper is to present a new method based on guaranteed numerical computation able to verify that a function $f : \mathcal{X} \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ satisfies

$$\forall x_1 \in \mathcal{X}, \forall x_2 \in \mathcal{X}, x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2). \tag{3.1}$$

To our knowledge, there does not exist any numerical method able to perform this injectivity test and moreover, the complexity of the algebraic manipulations involved often makes fail the formal calculus (especially when the function is not polynomial). Presently, in the context on structural identifiability, Braems and al. have presented in [BRA 01] an approximated method that verifies the injectivity around ε namely ε-injectivity. It consists in verifying the following condition

$$\forall x_1 \in \mathcal{X}, \forall x_2 \in \mathcal{X}, |x_1 - x_2| > \varepsilon \Rightarrow f(x_1) \neq f(x_2), \tag{3.2}$$

which can be view as an approximation of the condition (3.1).

Note that, many problems could be formulated as the injectivity verification of a function. For example, concerning the identification of parametric models, the problem of proving the *structural identifiability* amounts to check injectivity [E.W 90, BRA 01]. Other applications can be cited: For instance, consider the robotic arm with two degrees of freedom ($\theta_1 \in [0, \frac{\pi}{2}], \theta_2 \in [-\pi, \pi]$) represented in Figure 3.1(right).

Chapter written by Sébastien LAGRANGE and Nicolas DELANOUE and Luc JAULIN.

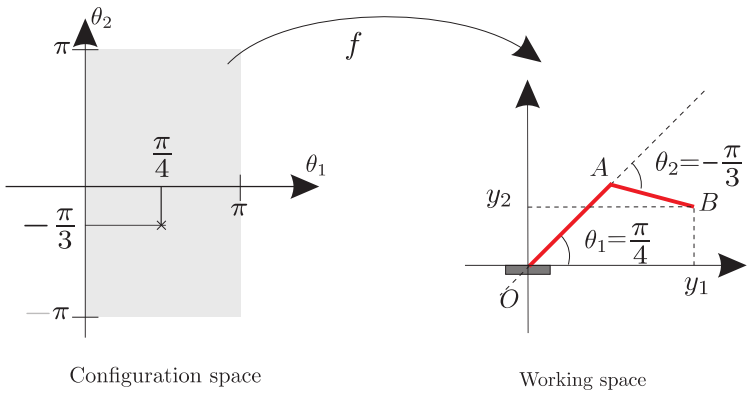

Figure 3.1. *A point in the configuration space and its corresponding robot configuration.*

Each point (θ_1, θ_2) of the *configuration space* is associated with a robot position (y_1, y_2) by the function

$$f : (\theta_1, \theta_2) \rightarrow \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 2cos(\theta_1) + 1.5cos(\theta_1 + \theta_2) \\ 2sin(\theta_1) + 1.5sin(\theta_1 + \theta_2) \end{pmatrix} \tag{3.3}$$

(See Figure 3.1). Now, a basic question is to know whether several pairs (θ_1, θ_2) lead to identical position (y_1, y_2) of the robot ending. This problem amounts to test the function f (defined in (3.3)) for injectivity.

This paper provides an efficient algorithm, based on interval analysis, able to check that a differentiable function is injective. The paper is organized as follows. Section 3.2 presents interval analysis. In Section 3.3, a new definition of partial injectivity makes possible the use of interval analysis techniques to test injectivity and to get a guaranteed answer. Section 3.4 presents an algorithm able to test a given differentiable function for injectivity. Finally, in order to show the efficiency of the algorithm, two illustrative examples are provided. A solver called ITVIA (Injectivity Test Via Interval Analysis) implemented in C++ is made available at `http://www.istia.univ-angers.fr/~lagrange/`.

3.2. Interval analysis

This section introduces some notions of interval analysis to be used in this paper. A vector interval or a *box* $[\mathbf{x}]$ of $\mathbb{R}^n$ is defined by

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x}\}, \tag{3.4}$$

where $\underline{x}$ and $\overline{x}$ are two elements of $\mathbb{R}^n$ and the partial order $\leq$ is understood componentwise. The set of all bounded boxes of $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$ as in [JAU 01].

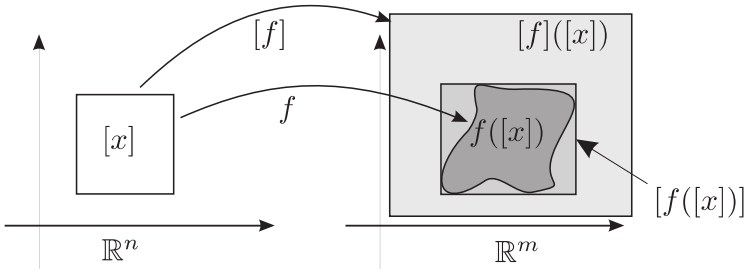

Figure 3.2. *Inclusion function $[f]$ of a function f.*

Remark 1 *By extension, one defines an* interval matrix $[M] = [\underline{M}, \overline{M}]$ *as the set of the matrices of the form :*

$$[M] = \{M \in \mathbb{R}^{n\times m} \mid \underline{M} \leq M \leq \overline{M}\} \tag{3.5}$$

and $\mathbb{IR}^{n\times m}$ denoted the set of all interval matrices of $\mathbb{R}^{n\times m}$. The properties of punctual matrices can naturally be extended to interval matrices. For example, $[M]$ is full column rank if all the matrices $M \in [M]$ are full column rank.

To *bisect* a box $[x]$ means to cut it along a symmetry plane normal to a side of maximal length. The length of this side is the *width* of $[x]$. A bisection of $[x]$ generates two non-overlapping boxes $[x_1]$ and $[x_2]$ such that $[x] = [x_1] \cup [x_2]$. The *hull box* $[\mathcal{X}]$ of a bounded subset $\mathcal{X} \in \mathbb{R}^n$ is the smallest box of $\mathbb{IR}^n$ that contains $\mathcal{X}$.

Interval arithmetic defined in [MOO 66] provides an effective method to extend all concepts of vector arithmetic to boxes.
Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a vector function; the set-valued function $[f] : \mathbb{IR}^n \to \mathbb{IR}^m$ is a *inclusion function* of f if, for any box $[x]$ of $\mathbb{IR}^n$, it satisfies $f([x]) \subset [f]([x])$ (see Figure 3.2). Note that $f([x])$ is usually not a box contrary to $[f]([x])$. Moreover, since $[f([x])]$ is the hull box of $f([x])$, one has

$$f([x]) \subset [f([x])] \subset [f]([x]). \tag{3.6}$$

The computation of an inclusion function $[f]$ for any analytical function f can be obtained by replacing each elementary operator and function by its interval counterpart [MOO 66, NEU 90].

Example 2 *An inclusion function for $f(x_1, x_2) = x_1^2 + \cos(x_1 x_2)$ is $[f]([x_1], [x_2]) = [x_1]^2 + \cos([x_1][x_2])$. For instance, if $[x] = ([-1, 1], [0, \frac{\pi}{2}])$ then the box $[f]([x])$ is computed as follows:*

$$\begin{array}{lll} [f]([-1,1],[0,\tfrac{\pi}{2}]) & = [-1,1]^2 + \cos([-1,1] \times [0,\tfrac{\pi}{2}]) & = [0,1] + \cos([-\tfrac{\pi}{2},\tfrac{\pi}{2}]) \\ & = [0,1] + [-1,1] & = [-1,2]. \end{array}$$

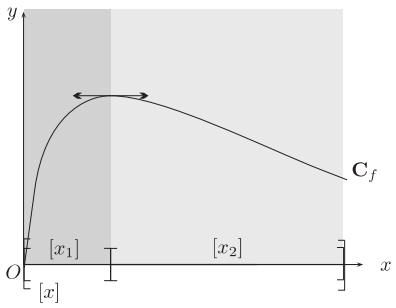

Figure 3.3. *Despite the fact that $f_{|[x_1]}$ and $f_{|[x_2]}$ are injection, f is not an injection.*

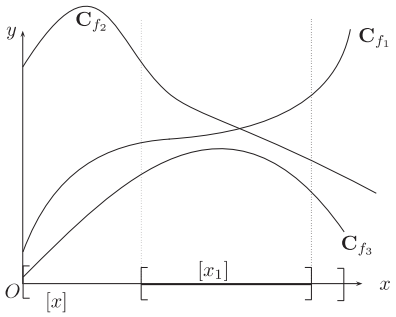

Figure 3.4. *Graphs of functions f_1, f_2 and f_3.*

3.3. Injectivity

Recall that this paper proposed to build an effective method to test differentiable function $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$ for injectivity. The main idea of the divide-and-conquer algorithm to be proposed is to partition $\mathcal{X}$ into subsets $\mathcal{X}_i$ where f restricted to $\mathcal{X}_i$ (denoted $f_{|\mathcal{X}_i}$) is an injection. However, as illustrated in Figure 3.3, injectivity is not "preserve by the union operation" *i.e.*

$$\left(f_{|\mathcal{X}_1} \text{ is an injection and } f_{|\mathcal{X}_2} \text{ is an injection}\right) \not\Rightarrow f_{|\mathcal{X}_1 \cup \mathcal{X}_2} \text{ is an injection.}$$

Thus, the injectivity cannot directly be used in our algorithm. That is why we are going to consider a concept akin to injectivity, namely *the partial injectivity*, that will be preserved by the union operation. The following subsections present the fundamental results that we will be used in the algorithm able to test function for injectivity.
First, we introduce the definition of the partial injectivity and give some illustrative examples. Then, we propose theorem which give a sufficient condition to test function for partial injectivity.

3.3.1. *Partial Injectivity*

Let us introduce the definition of *partial injectivity* of a function.

Definition 1 *Consider a function $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$ and any set $\mathcal{X}_1 \subset \mathcal{X}$. The function f is a partial injection of $\mathcal{X}_1$ over $\mathcal{X}$, noted $(\mathcal{X}_1, \mathcal{X})$-injective, if $\forall x_1 \in \mathcal{X}_1, \forall x \in \mathcal{X}$,*

$$x_1 \neq x \Rightarrow f(x_1) \neq f(x). \tag{3.7}$$

Remark 3 *Trivially, if f is $(\mathcal{X}, \mathcal{X})$-injective then f is an injection over $\mathcal{X}$.*

Example 4 *Consider the three functions of Figure 3.4. The functions f_1 and f_2 are $([x_1], [x])$-injective (although f_2 is not $[x]$-injective) whereas f_3 is not.*

Proposition 5 *Consider a function $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$ and $\mathcal{X}_1, \ldots, \mathcal{X}_p$ a collection of subsets of $\mathcal{X}$. We have*

$$\forall i, 1 \leq i \leq p, f \; is \; (\mathcal{X}_i, \mathcal{X}) - injective \; \Leftrightarrow f \; is \; (\bigcup_{i=1}^{p} \mathcal{X}_i, \mathcal{X}) - injective. \tag{3.8}$$

Proof. ($\Rightarrow$) One has $\forall x_i \in \mathcal{X}_i, \forall x \in \mathcal{X}$, $x_i \neq x \Rightarrow f(x_i) \neq f(x)$. Hence $\forall \check{x} \in (\cup_i \mathcal{X}_i), \forall x \in \mathcal{X}$, $\check{x} \neq x \Rightarrow f(\check{x}) \neq f(x)$, *i.e.* f is $(\cup_i \mathcal{X}_i, \mathcal{X})$-injective. ($\Leftarrow$) Trivial. ■

Remark 3 and Proposition 5 ensure the correctness of the divide-and-conquer algorithm to be present in Section 3.4.

3.3.2. *Partial Injectivity Condition*

In this paragraph, a fundamental theorem, which gives a sufficient condition of partial injectivity, is presented. First, let us introduce a generalization of the Mean Value Theorem[1].

Theorem 6 (Generalized Mean Value Theorem) *Consider a differentiable function $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$. Let ∇f be its Jacobian matrix and $[x] \subset \mathcal{X}$. One has*

$$\forall x_1, x_2 \in [x], \exists J_f \in [\nabla f([x])] \; such \; that \; f(x_2) - f(x_1) = J_f \cdot (x_2 - x_1), \tag{3.9}$$

where $[\nabla f([x])]$ denotes the hull box of $\nabla f([x])$.

Proof. According to Mean-Value Theorem applied on each components $f_i : \mathbb{R}^n \to \mathbb{R}$ of f ($1 \leq i \leq m$) and since the segment $seg(x_1, x_2)$ belongs to $[x]$, we have

$$\exists \xi_i \in [x] \; such \; that \; f_i(x_2) - f_i(x_1) = \nabla f_i(\xi_i) \cdot (x_2 - x_1). \tag{3.10}$$

Taking $J_{f_i} = \nabla f_i(\xi_i)$, we get

$$\exists J_{f_i} \in \nabla f_i([x]) \; such \; that \; f_i(x_2) - f_i(x_1) = J_{f_i} \cdot (x_2 - x_1). \tag{3.11}$$

---

1. Let $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}$, $f \in \mathcal{C}^1$. If $x_1, x_2 \in \mathcal{X}$ such that the segment between x_1 and x_2, noted $seg(x_1, x_2)$, is included in $\mathcal{X}$. Then, there exists $\xi \in seg(x_1, x_2)$ such that

$$f(x_2) - f(x_1) = \nabla f(\xi) \cdot (x_2 - x_1).$$

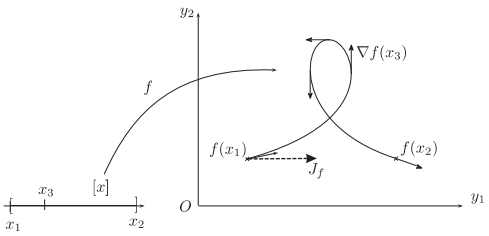

Figure 3.5. *Graph of* $f : \mathbb{R} \to \mathbb{R}^2$.

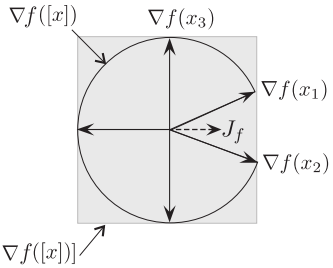

Figure 3.6. *Illustration of the set* $[\nabla f([x])]$.

Thus

$$\exists J_f \in \left(\nabla f_1\left([x]\right), \ldots, \nabla f_m\left([x]\right)\right)^T \; such\ that\ f(x_2) - f(x_1) = J_f \cdot (x_2 - x_1). \tag{3.12}$$

i.e., since $\left(\nabla f_1\left([x]\right), \ldots, \nabla f_m\left([x]\right)\right)^T \subset [\nabla f([x])]$ (see (3.6)),
$\exists J_f \in [\nabla f([x])]\ such\ that\ f(x_2) - f(x_1) = J_f \cdot (x_2 - x_1)$. ■

Example 7 *Consider the function*

$$f : \left\{ \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R}^2 \\ x & \to & (y_1, y_2)^T \end{array} \right. . \tag{3.13}$$

depicted in Figure 3.5. Figure 3.6 represents the set $\nabla f([x])$ *of all derivatives of* f *(drawn as vectors) and its hull box* $[\nabla f([x])]$. *One can see that the vector* J_f *defined in (3.9) belongs to* $[\nabla f\left([x]\right)]$ *(but* $J_f \notin \nabla f\left([x]\right)$*) as forecasted by Theorem 6.*

Now, the following theorem introduces a sufficient condition of partial injectivity. This condition will be exploited in next section in order to design a suitable algorithm that test injectivity.

Theorem 8 *Let* $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$ *be a differentiable function and* $[x_1] \subset [x] \subset \mathcal{X}$. *Set* $[\tilde{x}] = \left[f^{-1}\left(f\left([x_1]\right)\right) \cap [x]\right]$. *If the interval matrix* $[\nabla f\left([\tilde{x}]\right)]$ *is full column rank then* f *is* $([x_1], [x])$*-injective.*

Proof. The proof is by contradiction. Assume that f is not ($[x_1]$,$[x]$)-injective then

$$\exists x_1 \in [x_1], \exists x_2 \in [x]\ \ such\ that\ x_1 \neq x_2\ and\ f\left(x_1\right) = f\left(x_2\right). \tag{3.14}$$

Now, since $f(x_1) = f(x_2)$, one has $x_2 \in f^{-1}\left(f\left([x_1]\right)\right) \cap [x]$ and trivially $x_1 \in f^{-1}\left(f\left([x_1]\right)\right) \cap [x]$. Therefore, since $\left(f^{-1}\left(f\left([x_1]\right)\right) \cap [x]\right) \subset \left[f^{-1}\left(f\left([x_1]\right)\right) \cap [x]\right] =$

$[\tilde{x}]$ (see Equation (3.6)), one has $x_1, x_2 \in [\tilde{x}]$.
Hence, (3.14) implies

$$\exists x_1, x_2 \in [\tilde{x}]\,,\ such\ that\ x_2 \neq x_1\ and\ f(x_1) = f(x_2)\,. \tag{3.15}$$

To conclude, according to Theorem 6, $\exists x_1, x_2 \in [\tilde{x}]$, $\exists J_f \in [\nabla f([\tilde{x}])]$ such that

$$x_1 \neq x_2\ and\ 0 = f(x_2) - f(x_1) = J_f \cdot (x_2 - x_1), \tag{3.16}$$

i.e. $\exists J_f \in [\nabla f([\tilde{x}])]$ such that J_f is not full column rank and therefore the (interval) matrix $[\nabla f([\tilde{x}])]$ is not full column rank. ■

3.4. ITVIA Algorithm

This section presents the Injectivity Test Via Interval Analysis (ITVIA) algorithm designed from Proposition 5 and Theorem 8. ITVIA (defined in Algorithm 2) uses the divide-and-conquer strategy to check a given differentiable function $f : [x] \subset \mathbb{R}^n \to \mathbb{R}^m$ for injectivity. Algorithm 1 is a sub-algorithm of 2 that checks (a sufficient condition of) partial injectivity.
- Algorithm 1 checks if the interval matrix $\left[\nabla f\left(\left[f^{-1}\left(f\left([x_1]\right)\right) \cap [x]\right]\right)\right]$ is full rank. In the positive case, according to Theorem 8, the function f is $([x_1], [x])$-injective. Therefore, Algorithm 1 can be viewed as a test for partial injectivity.
- Algorithm 2 divides the initial box $[x]$ into a paving $\{[x_i]\}_i$ such that, for all i, the function f is $([x_i], [x])$-injective. Then, since $[x] = (\cup_i [x_i])$ and according to Proposition 5, f is $[x]$-injective.

In Algorithm 1, a set inversion technique [GOL 05, JAU 01] is first exploited to characterize a box $[\tilde{x}]$ that contains $[f^{-1}(f([x_1])) \cap [x]]$. Secondly, an evaluation of $[\nabla f]([\tilde{x}])$ is performed in order to test its rank[2]. Thus, since $[\nabla f([\tilde{x}])] \subset [\nabla f]([\tilde{x}])$ and according to Theorem 8, one can test whether f is $([x_1], [x])$-injective.
Algorithm 2 creates a paving of the initial box $[x]$ such that, for all i, the function f is $([x_i], [x])$-injective. Therefore, if the algorithm terminates, then f is an injection.
By combination of these two algorithms, we can prove that a function is injective over a box $[x]$. A solver, called ITVIA, developed in C++ is made available and tests the injectivity of a given function $f : \mathbb{R}^2 \to \mathbb{R}^2$ (or $f : \mathbb{R} \to \mathbb{R}^2$) over a given box $[x]$.

3.5. Examples

In this section, two examples are provided in order to illustrate the efficiency of the solver ITVIA presented in previous section. We are going to check the injectivity of two functions $f : \mathbb{R}^2 \to \mathbb{R}^2$ over a given box $[x]$.

---

2. Several techniques exist to test an interval matrix for full ranking. If it is square, the simplest way consists in verifying that the determinant (which is an interval) not contains zero. Otherwise (*i.e.* $f : \mathbb{R}^n \to \mathbb{R}^m$), the Interval Gauss Algorithm could be used [NEU 90].

Algorithm 1 Partial_Injectivity_Test

Require: $f \in \mathcal{C}^1$, $[x]$ the initial box and $[x_1] \subset [x]$.
Ensure: A boolean :
- true : f is $([x_1],[x])$-injective,
- false : f may or not be partially injective.

Initialization : $\mathcal{L}_{stack} := \{[x]\}$, $[\tilde{x}] := \emptyset$.
while $\mathcal{L}_{stack} \neq \emptyset$ **do**
  Pop $\mathcal{L}_{stack}$ into $[w]$.
  if $[f]([w]) \cap [f]([x_1]) \neq \emptyset$ **then**
    if width$([w])$ > width$([x_1])$ \\ To avoid useless splitting of $[w]$ *ad infinitum* **then**
      Bisect $[w]$ into $[w_1]$ and $[w_2]$.
      Stack $[w_1]$ and $[w_2]$ in $\mathcal{L}_{stack}$.
    else
      $[\tilde{x}] = [[\tilde{x}] \cup [w]]$.
    end if
  end if
end while
if $[\nabla f]([\tilde{x}])$ is full rank **then**
  Return true \\ "f is $([x_1],[x])$-injective"
else
  Return False \\ "Failure"
end if

Algorithm 2 Injectivity_Test_Via_Interval_Analysis

Require: f a $\mathcal{C}^1$ function and $[x]$ the initial box.
Initialization : $\mathcal{L} := \{[x]\}$.
while $\mathcal{L} \neq \emptyset$ **do**
  Pull $[w]$ in $\mathcal{L}$.
  if Partial_Injectivity_Test$(f,[x],[w])$ = False **then**
    Bisect $[w]$ into $[w_1]$ and $[w_2]$.
    Push $[w_1]$ and $[w_2]$ in $\mathcal{L}$.
  end if
end while
Return "f is injective over $[x]$".

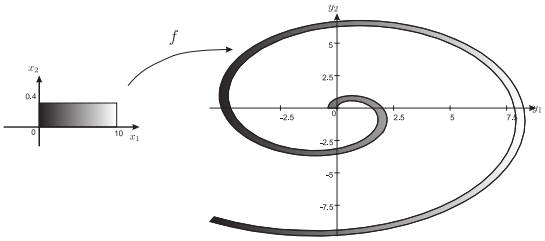

Figure 3.7. *Graph of the function f defined in (3.17)*

Figure 3.8. *Bisection of $[x]$ obtained by ITVIA for the function f defined in (3.17). All the grey boxes have been proved partially injective.*

3.5.1. *Spiral function*

Consider the function f, depicted in Figure 3.7, defined by

$$f: \begin{array}{ccc} \mathbb{R}^2 & \to & \mathbb{R}^2 \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \to & \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 \sin(x_1) + x_2 \frac{x_1 \sin(x_1) - \cos(x_1)}{\sqrt{x_1^2+1}} \\ x_1 \cos(x_1) + x_2 \frac{\sin(x_1) + x_1 \cos(x_1)}{\sqrt{x_1^2+1}} \end{pmatrix} \end{array} \tag{3.17}$$

and test its injectivity over the box $[x] = \left([0, 10], [0, \frac{4}{10}]\right)^T$. After less than $0.1\,\mathrm{sec}$ on a Pentium $1.7GHz$, ITVIA proved that f is injective over $[x]$. The initial box $[x]$ has been divided in a set of sub-boxes where f is partially injective. Figure 3.8 shows the successive bisections of $[x]$ made by ITVIA.

3.5.2. *Ribbon function*

Consider the ribbon function f (depicted in Figure 3.9) defined by

$$f: \begin{array}{ccc} \mathbb{R}^2 & \to & \mathbb{R}^2 \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} & \to & \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} \frac{x_1}{2} + (1 - x_2)\cos(x_1) \\ (1 - x_2)\sin(x_1) \end{pmatrix} \end{array} \tag{3.18}$$

and get interest with its injectivity over the box $[x] = ([-1, 4], [0, \frac{1}{10}])^T$. Since the ribbon overlapping, one can see that f is not injective over $[x]$. After 3 seconds, the solver ITVIA is stopped (before going to end). It returns the solution presented in Figure 3.10. The function f has been proved to be a partial injection on the gray domain over $[x]$, whereas the white domain corresponds to the indeterminate domain where ITVIA was not able to prove the partial injectivity. Indeed, the indeterminate domain corresponds to the non injective zone of f where all points are mapped in the overlapping zone of the ribbon.

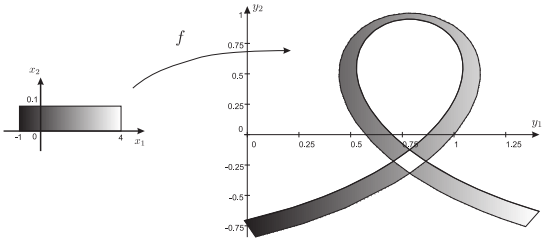

Figure 3.9. *Graph of the function f defined in (3.18).*

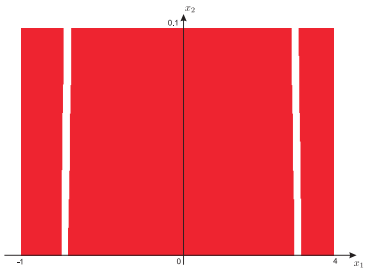

Figure 3.10. *Partition of the box $[x]$ obtained by ITVIA for the function f defined in (3.18). In gray, the partial injectivity domain and, in white, the domain where the f is not proved partially injective.*

3.6. Conclusion

In this paper, we have presented a new algorithm, based on interval analysis, able to test differentiable functions for injectivity. In case of functions $f : \mathbb{R} \rightarrow \mathbb{R}^2$ and $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, a C++ solver is available. From a given function f and a given box $[x]$, the solver divides $[x]$ in two domains : a partially injective domain and a indeterminate domain (where the function may or not be injective). Of course, when the indeterminate domain is empty, the function is proved injective over $[x]$.

3.7. Bibliography

[BRA 01] BRAEMS I., JAULIN L., KIEFFER M., WALTER E., "Guaranteed numerical alternatives to structural identifiability testing", *Conference on Decision and Control.*, 2001.

[E.W 90] E.WALTER, PRONZATO L., "Qualitative and quantitative experiment design for phenomenological models, a survey.", *Automatica*, vol. 26, p. 195-213, 1990.

[GOL 05] GOLDSZTEJN A., "A Right-Preconditioning Process for the Formal-Algebraic Approach to Inner and Outer Estimation of AE-solution Set.", *Reliable Computing*, 2005.

[JAU 01] JAULIN J., KIEFFER M., DIDRIT D., WALTER E., *Applied Interval Analysis*, Springer, 2001.

[MOO 66] MOORE R., *Interval Analysis*, Prentice-Hall, Englewood Cliffs, 1966.

[NEU 90] NEUMAIER A., *Interval Methods for Systems of Equations*, Camb. Univ. Pres, 1990.

Chapter 4

An Interval-based Approximation Method for Discrete Changes in Hybrid cc

4.1. Introduction

Hybrid systems are systems consisting of continuous and discrete changes. Figure 4.1 (a) illustrates a simple hybrid system in which a particle falls down by gravity and bounces off the ground. While the particle's falling is modeled by differential equations, the bouncing is governed by equations describing discrete changes.

Hybrid cc (*Hybrid concurrent constraint programming*) [GUP 95], a compositional and declarative language based on constraint programming, has been proposed as a high-level development tool of hybrid systems for simulation, animation and design. Below is a description of the hybrid system in Figure 4.1 (a) (See Section 4.2 for the syntax and implementation):

```
y = 10, y' = 0,              // initial conditions
hence {
    cont(y),                 // height is continuous
    if y > 0 then y'' = -10, // free fall
    if y = 0 then
        y' = -0.5 * prev(y') // bounce
}
```

In our experience of modeling a number of hybrid systems using Hybrid cc, however, we have encountered several difficulties. Figure 4.1 (b) shows a further execution

---

Chapter written by Daisuke ISHII, Kazunori UEDA and Hiroshi HOSOBE.

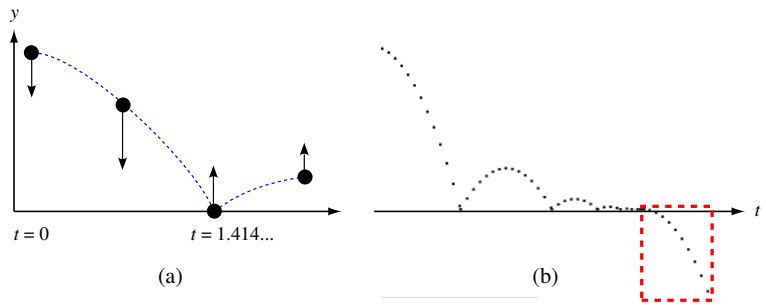

Figure 4.1. *(a) A bouncing particle and (b) An unexpected result*

result of the above example. As the distance between the particle and the ground gets closer, the particle moves in an unexpected way[1]. The problem is due to the computation of discrete changes in the current implementation of Hybrid cc (Section 4.3). After the particle loses energy and its height stays within ϵ (to tolerate numerical errors), the implementation ignores discrete changes, and as the result the particle goes underground.

The contribution of this paper is to develop a method for obtaining (guaranteed) approximate solutions of discrete changes in hybrid systems. As we discuss in Section 4.3, our goal is to bring correct techniques to a number of issues such as reliable simulation, reachability analysis and backward computation in hybrid systems with Hybrid cc. Our approach is based on interval arithmetic (Section 4.4). The proposed method detailed in Section 4.5 aims to enclose trajectories of hybrid systems by tight intervals or boxes without losing any solutions. We experimented and evaluated the effect of the method (Section 4.6).

4.2. An overview of Hybrid cc

4.2.1. *The Hybrid cc language*

The formal operational semantics of Hybrid cc is described in [GUP 95]. The basic syntax for processes in Hybrid cc is shown in Table 4.1, where we denote processes by A and constraints by C. Computation in Hybrid cc is performed by processes interacting with the *constraint store*. A process **if** C **then** A triggers discrete changes when an *ask condition* C is entailed by the current constraint store. If C is entailed, the process is reduced to A. Otherwise, the process is ignored. If the entailment of C cannot be determined at the moment, the process is suspended.

1. This kind of phenomenon that causes an infinite number of discrete changes in a finite length of time is known as *Zeno behavior*.

C	Add the constraint C
if C **then** A	If C holds, reduce to A
A, A	Parallel composition
hence A	Execute A at every instant after now
cont(C)	Declare that C is continuous over time
prev(C)	The value of C before entering a phase

Table 4.1. *The basic syntax of Hybrid cc processes*

4.2.2. *Implementation of Hybrid cc*

The implementation of the Hybrid cc interpreter is detailed in [CAR 98]. The interpreter handles a Hybrid cc process by alternating *point phases* and *interval phases* as follows:

Step 1 Computation starts from a point phase.

Step 2 In the point phase, the interpreter performs reductions of processes such as the propagation of arithmetic constraints. The computation of the point phase ends at a stable point, where all constraints have been propagated and all possible reductions have been completed.

Step 3 After the point phase, the computation proceeds to an interval phase. Each expression **hence** A, which is passed from Step 2, is reduced to A and processed. All the processes are reduced until a stable point is reached as in Step 2. The resulting constraint store consists of constraints on the derivatives of variables (i.e., ODEs), conditions for discrete changes and constraints to be reduced in the future.

Step 4 Integration of the arithmetic constraints that were told in the previous step is processed until one of the conditions changes its status.

Step 5 If there are no processes left in the constraint store, the computation terminates. Otherwise, go to Step 2.

The arithmetic constraint solvers in the implementation handle nonlinear equations (NLEs) and ordinary differential equations (ODEs). In the implementation, the following approaches are adopted:

– The computation by the solvers is based on interval arithmetic to model uncertainty in the parameters.

– The NLE solver takes an arithmetic constraint which can be rewritten in the implicit form $f(x) = 0$. Solving of the constraint is done by interval pruning of each variable in the constraint. In the implementation, the following four pruning operators are adopted: (1) use of indexicals, (2) splitting of intervals, (3) the Newton-Raphson method, (4) the Simplex method (optional).

– The ODE solver uses the Runge-Kutta-Fehlberg method which numerically integrates with adaptive step-sizes. The solver reduces equations of arbitrary form to an explicit form by propagating equations in each step of integration.

– The ODE solver stops the integration at the *breakpoint* at which any state of constraints in the constraint store is changed. To take care of an overshoot from the breakpoint, the solver backtracks until an exact solution within a constant ϵ is found.

4.3. The objective of the paper

We are developing systems for handling physical simulations and animations based on Hybrid cc. For example, we want to simulate a billiard table on which a number of balls roll and collide with other balls or table edges. The goal is to obtain trajectories of the balls from a description written in Hybrid cc that directly reflects the laws of elementary physics.

However, the current implementation of Hybrid cc has several limitations in the reliability aspect as follows:

– Although the implementation supports interval arithmetic, it does not use interval arithmetic to handle discrete changes such as **if** C **then** A, based on interval arithmetic.

– The ODE solver detects discrete changes during the integration with an ad hoc method. To work around computation errors, the detection allows for a tolerance ϵ, but this may result in qualitatively different trajectories.

The goal of this paper is to propose a reliable method for computing trajectories in Hybrid cc. Our method obtains valid and tight interval enclosures of discrete changes, corresponding to Step 4 in Section 4.2.2. By combining the proposed method and other known techniques for reliable computing, we will be able to guarantee the accuracy of trajectories, as well as to verify the reachable area of objects. We also expect to apply the method to problems such as backward computation of trajectories.

4.4. Background of interval arithmetic

To describe our method based on interval arithmetic [MOO 66], the following notions and definitions are used.

4.4.1. *Basic notions of interval arithmetic*

$\mathcal{F}$ denotes the set of machine-representable floating-point numbers, $\mathcal{I}$ denotes the set of intervals over $\mathbb{R}$ whose bounds are in $\mathcal{F}$, and I denotes an interval in $\mathcal{I}$. $\mathrm{lb}(I)$ denotes the lower bound, $\mathrm{ub}(I)$ denotes the upper bound, and $\mathrm{w}(I)$ denotes the width of I. $\mathcal{D}$ denotes the set of boxes over $\mathbb{R}^n$ whose bounds are in $\mathcal{F}$, and D denotes a box in $\mathcal{D}$. Given a real r and a subset A of $\mathbb{R}^n$, $\bar{r}$ denotes the smallest interval in $\mathcal{I}$ containing r, and $\Box A$ the smallest box in $\mathcal{D}$ containing A. If g is a function, G denotes the *interval extension* of g.

4.4.2. *ODE solving based on interval arithmetic*

There have been several techniques for solving ODEs based on interval arithmetic or constraint propagation; see [LOH 92, NED 99, DEV 98, CRU 03] for example. A *solution* of an ODE system $\mathcal{O}$ with the initial value $u(t_0) = u_0$ is a function $s^*(t) : \mathbb{R} \to \mathbb{R}^n$ satisfying $\mathcal{O}$ and the initial conditions $s^*(t_0) = u_0$. Also the solution of $\mathcal{O}$ is denoted as the function $s(t_0, u_0, t) : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ such that $s(t_0, u_0, t) = s^*(t)$. An *interval solution* of $\mathcal{O}$ is an interval extension S of s. A box B is an *a priori bound* of a solution s with respect to $\mathcal{O}$, $[t_0, t_1]$ and D, if for all t in $[t_0, t_1]$, $S(t_0, D, t) \subseteq B$.

LEMMA.– *(Continuity of the solution of ODEs) Let f be continuous over an open set E over $\mathbb{R} \times \mathbb{R}^n$ with the property that for every (t_0, u_0) in E, the initial value problem $\{u' = f(t, u), u(t_0) = u_0\}$ has a unique solution $u(t) = s(t_0, u_0, t)$. Let $T(t_0, u_0)$ be the maximal interval in which $u(t) = s(t_0, u_0, t)$ exists. Then s is continuous over $\{(t_0, u_0, t) \mid (t_0, u_0) \in E, t \in T(t_0, u_0)\}$.*

4.5. The proposed method

Two algorithms TRACE and PRUNEANDMERGE compose our method. By processing Step 2 and Step 3 in Section 4.2.2 using interval arithmetic, we obtain an ODE system $\mathcal{O}$ which has an initial value D_0 and an initial time T_0. The main TRACE algorithm computes trajectories with respect to $\mathcal{O}$. TRACE simulates the continuous evolution of the system until an ask condition C is entailed. PRUNEANDMERGE algorithm computes the precise enclosure of the point of a discrete change.

4.5.1. *Assumptions on the proposed method*

We consider problems that satisfy the following assumptions:

– Any ODE system $\mathcal{O}$ has the form $u' = f(t, u)$ (f is continuous) and a unique solution.

– Any ask condition C is an arithmetic constraint of the form $f_d = 0$, where f_d is a continuous function $f_d : \mathbb{R}^n \to \mathbb{R}$ and is invariant over time.

Also, in this paper, we investigate the method under the following additional assumptions that simplify the problems:

– We consider problems with a single variable and assume they cause only one discrete change at one time. It is a topic of future study to improve the method in this paper to handle more general cases.

– The solution $s(t_0, u_0, t)$ of $\mathcal{O}$ either increases or decreases monotonically over any interval T_d whose width is equal to $\mathrm{w}(T_0)$ and which T_d includes t_d satisfying $f_d(s(t_0, u_0, t_d))$.

4.5.2. TRACE *algorithm*

We give the TRACE algorithm in Figure 4.2, and illustrate its basic idea in Figure 4.3. TRACE computes a trajectory with respect to $\mathcal{O}$ and ensures a set of step solutions S. The main idea is to obtain the "lower bound" of the trajectory, which is computed from the initial value D_0 at $\mathrm{lb}(T_0)$, and to shift it by $\mathrm{w}(T_0)$ over time. At the beginning of the loop (line 6), TRACE obtains the one-step solution of the trajectory consisting of an a priori bound $B_{t,h}$ and a tight solution D_{t+h}. Interval solutions of ODEs can be obtained using existing techniques, e.g. [NED 99]. TRACE preserves several a priori bounds of the width $\mathrm{w}(T_0)$ in a buffer Q (line 12), and obtains the step solution by taking the union of elements in Q (line 15). Figure 4.3 (a) illustrates the idea. Note that we assume the step size h is equal to $\mathrm{w}(T_0)$. Although boundary trajectories do not enclose an envelope created by shifting an extremal value in a trajectory (consider Figure 4.3 (b)), TRACE can safely enclose the envelope since a priori bounds are preserved in Q.

TRACE detects a time point of a discrete change by evaluating the ask condition C in each step of the loop (lines 7 and 9). We can use the existing interval-based method like [Van 97] to evaluate ask conditions. TRACE preserves step solutions in the buffer Q, from B_l in which C is first entailed to a step where C is overshot (line 8). Figure 4.3 (c) illustrates the idea. As a result, Q encloses an area of a discrete change, and we obtain an output B_{result} by passing Q to the PRUNEANDMERGE algorithm. Below is a soundness theorem of the method:

THEOREM.– *Assume the assumptions in Section 4.5.1 hold. Assume there exists* $t \in \mathbb{R}$ *such that for all* $u \in D_0$, $f_d(s(t_0, u, t)) > 0$ *holds. Assume there exists* $t' \in \mathbb{R}$ *such that for all* $u \in D_0$, $f_d(s(t_0, u, t')) < 0$ *holds. Then, for all* $u \in D_0$, *there exists* $t_d \in \mathbb{R}$ *between* t *and* t' *such that the ask condition* $f_d(s(t_0, u, t_d)) = 0$ *is entailed.*

PROOF.– For each u in D_0, $f_d(s(t_0, u, t)) > 0$ and $f_d(s(t_0, u, t')) < 0$ hold. Since f_d is continuous, there exists a solution $s_d \in \mathbb{R}^n$ such that $f_d(s_d) = 0$ holds between $s(t_0, u, t)$ and $s(t_0, u, t')$. Since s is continuous by Lemma in Section 4.4.2, and also since s increases or decreases monotonically, there exists a unique time t_d between t and t' such that $s(t_0, u, t_d) = s_d$ holds. ∎

4.5.3. PRUNEANDMERGE *algorithm*

We give the PRUNEANDMERGE algorithm in Figure 4.4. Figure 4.5 illustrates an application of the algorithm to an expanded example in which a 2-dimensional variable over the x and y axes and a nonlinear condition C are used. Note that Figure 4.5 (a) corresponds to Figure 4.3 (c), and is overwritten by the computation result of PRUNEANDMERGE. We apply a *branch and prune* algorithm to the time component of an area of a discrete change based on the following hull consistency (line 1):

Require: an ODE system $\mathcal{O}$, initial time T_0, initial value D_0, step size h, an ask condition C
Ensure: a set S of step solutions, an area of a discrete change B_{result}

1: $t := \mathrm{lb}(T_0)$
2: $D := D_0$
3: $Q :=$ a buffer to preserve a trajectory for several steps
4: $B_l := \emptyset$
5: **loop**
6: $(B_{t,h}, D_{t+h}) :=$ do a step computation w.r.t. $\mathcal{O}$, t, D, h
7: **if** $C(D_{t+h})$ is entailed **then**
8: $B_l := B_{t,h}$
9: **else if** $C(D_{t+h})$ is overshot **then**
10: break
11: **end if**
12: put $(B_{t,h}, D_{t+h})$ to the tail of Q
13: remove a redundant element from the top of Q
14: $D := D_{t+h}$
15: $S := S \cup \{$the sum of solutions in Q for $\mathrm{w}(T_0)\}$
16: $t := t + h$
17: **end loop**
18: $B_{result} :=$ PRUNEANDMERGE$(Q, \mathcal{O}, C, \mathrm{w}(T_0))$
19: return B_{result}

Figure 4.2. TRACE *algorithm*

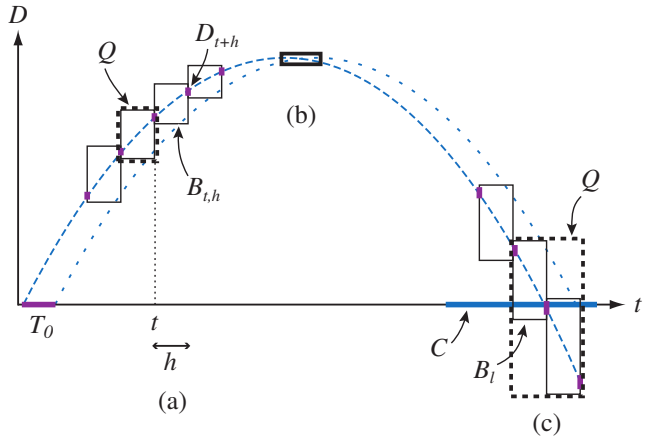

Figure 4.3. *Solving of an ODE system with an ask condition by* TRACE

Require: a buffer of solutions Q, an ODE system $\mathcal{O}$, an ask condition C, width w
Ensure: a box B_{result}
1: B := $\Box$\{branch and prune Q w.r.t. $\mathcal{O}, C$\}
2: B' := enlarge B for w
3: B_{result} := prune B'_D w.r.t. C
4: return B_{result}

Figure 4.4. PRUNEANDMERGE *algorithm*

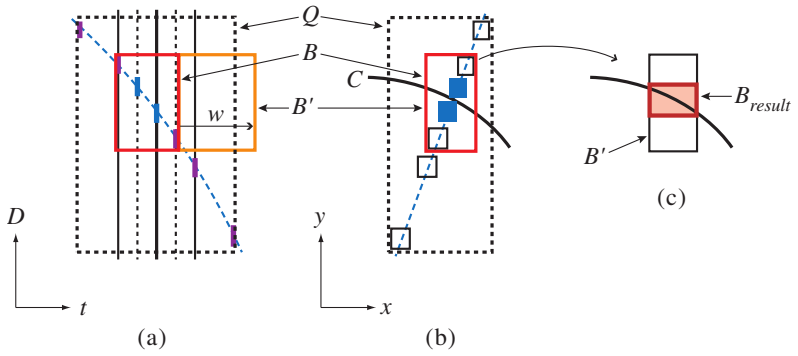

Figure 4.5. *Pruning of an area of a discrete change by* PRUNEANDMERGE

DEFINITION.– *(Hull consistency of a time interval of a discrete change) A time interval T_d of a discrete change with respect to an ODE system $\mathcal{O}$ and an ask condition $f_d = 0$ (F_d is an interval extension of f_d) is hull-consistent with respect to $T_d = [t_l, t_u]$, if the condition*

$$(F_d(D_l) > \bar{0} \wedge F_d(D_u) < \bar{0}) \vee (F_d(D_l) < \bar{0} \wedge F_d(D_u) > \bar{0})$$

holds, where $D_l = S(t_0, D_0, t_l)$ and $D_u = S(t_0, D_0, t_u)$.

Figure 4.5 (a) and (b) illustrates the result of the bisection method. The result is merged by obtaining the box enclosure (line 1). The algorithm enlarges B by w at line 2, as TRACE shifts trajectories for w(T_0) ($= w$). Moreover, we can prune B' by solving the condition C over the (x, y)-component (line 3, Figure 4.5 (c)).

4.6. Experimental results

We evaluated the effect of the proposed method by simulating the example in Section 4.1. In the simulation, we used the existing interval-based solvers; VNODE-LP [NED 06] for ODEs and Elisa [GRA 05] for NLEs. Table 4.2 shows the initial value, the solution at $t = 2$ after the first bounce ($t = \sqrt{2}$), and the solution at $t = 3$ after the

t	(a) Direct computation	(b) Proposed method	(c) Theoretical solution
0	[10.00000, 10.00000]	[10.00000, 10.00000]	10
2	[2.424746, 2.427454]	[2.426262, 2.426676]	2.426406...
3	[0.4398578, 0.4739294]	[0.4559554, 0.4613689]	0.4594154...

Table 4.2. *Experimental results*

second bounce ($t = 2\sqrt{2}$). We solved ODEs with the step size of 0.0001. We can see an improvement in accuracy by comparing the results obtained by (a) direct interval computation without refinements and (b) our method. We can also make sure that the results enclose the theoretical solution (c).

4.7. Related work

Techniques for modeling and verification of hybrid systems based on constraint logic programming have been proposed. Hickey and Wittenberg [HIC 04] presented an approach using the CLP(F) language, which can describe analytic relations between real variables and functions. CLP(F) supports interval arithmetic. However suppression of interval divergence in computing discrete changes is not considered. Urbina [URB 96] proposed a method using the CLP($\mathcal{R}$) language. Computations in the method are not based on intervals, and solutions are not accurate. Moreover, only linear numerical constraints are supported.

Various safety verification techniques for hybrid systems, which compute reachable state space represented by intervals, have been developed. Some of the techniques use box representation of continuous parts of state space to check how a trajectory moves over their boundaries [STU 97, HEN 00]. Ratschan and She [RAT 05] proposed methods to refine the box representation using constraint propagation techniques. Refinement is done by adding constraints describing continuous and discrete changes. In these techniques hybrid systems are modeled based on hybrid automata.

4.8. Conclusion

We proposed a method for obtaining interval enclosures of discrete changes in hybrid systems. After relaxing the assumptions in Section 4.5.1, we will next implement the proposed method in the Hybrid cc interpreter. The improved interpreter provides a reliable framework to model and control variety of hybrid systems that occur in real-life applications.

Acknowledgement.

The authors thank Mitsuhiro Nishimura, Ryota Nishizawa, Yoshiyuki Ohno, Yui Sasajima, Shinichi Tobita and Satoshi Usami for helping the evaluation of the proposed method.

4.9. Bibliography

[CAR 98] CARLSON B., GUPTA V., "Hybrid cc with Interval Constraints", *HSCC 1998*, vol. 1386 of *LNCS*, Springer, 1998.

[CRU 03] CRUZ J., BARAHONA P., "Constraint Satisfaction Differential Problems", *CP 2003*, vol. 2833 of *LNCS*, Springer, p. 259–273, 2003.

[DEV 98] DEVILLE Y., JANSSEN M., HENTENRYCK P. V., "Consistency Techniques for Ordinary Differential Equations", *CP 1998*, vol. 1520 of *LNCS*, Springer, p. 162–176, 1998.

[GRA 05] GRANVILLIERS L., SORIN V., "Elisa", http://sourceforge.net/projects/elisa/, 2005.

[GUP 95] GUPTA V., JAGADEESAN R., SARASWAT V. A., BOBROW D., "Programming in Hybrid Constraint Languages", *Hybrid Systems 1994*, vol. 999 of *LNCS*, Springer, 1995.

[HEN 00] HENZINGER T. A., HOROWITZ B., MAJUMDAR R., WONG-TOI H., "Beyond HyTech: Hybrid Systems Analysis Using Interval Numerical Methods", *HSCC 2000*, vol. 1790 of *LNCS*, Springer, 2000.

[HIC 04] HICKEY T. J., WITTENBERG D. K., "Rigorous Modeling of Hybrid Systems using Interval Arithmetic Constraints", *HSCC 2004*, vol. 2993 of *LNCS*, p. 402–416, 2004.

[LOH 92] LOHNER R. J., "Computation of Guaranteed Enclosures for the Solutions of Ordinary Initial and Boundary Value Problems", *Computational Ordinary Differential Equations*, Oxford University Press, 1992.

[MOO 66] MOORE R. E., *Interval Analysis*, Prentice-Hall, 1966.

[NED 99] NEDIALKOV N. S., JACKSON K. R., CORLISS G. F., "Validated Solutions of Initial Value Problems for Ordinary Differential Equations", *Applied Mathematics and Computation*, vol. 105, num. 1, p. 21–68, 1999.

[NED 06] NEDIALKOV N. S., "VNODE-LP: A Validated Solver for Initial Value Problems in Ordinary Differential Equations,", *TR CAS-06-06-NN*, McMaster University, 2006.

[RAT 05] RATSCHAN S., SHE Z., "Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement", *HSCC 2005*, vol. 3414 of *LNCS*, Springer, 2005.

[STU 97] STURSBERG O., KOWALEWSKI S., HOFFMANN I., PREUSIG J., "Comparing Timed and Hybrid Automata as Approximations of Continuous Systems", *Hybrid Systems 1996*, vol. 1273 of *LNCS*, Springer, p. 361–377, 1997.

[URB 96] URBINA L., "Analysis of Hybrid Systems in CLP(R)", *CP 1996*, vol. 1118 of *LNCS*, Springer, 1996.

[Van 97] VAN HENTENRYCK P., MCALLESTER D., KAPUR D., "Solving Polynomial Systems Using a Branch and Prune Approach", *SIAM Journal on Numerical Analysis*, vol. 34, num. 2, p. 797–827, 1997.