Guaranteed Deterministic Global Optimization using Constraint Programming through Algebraic, Functional and Piecewise Differential Constraints

Hugo Joudrier

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Génie Industriel : Conception et Production**

Arrêté ministériel : 25 mai 2016

Présentée par

Hugo JOUDRIER

Thèse dirigée par **Khaled HADJ-HAMOU**, Professeur des Universités, INSA Lyon

préparée au sein du **Laboratoire des Sciences pour la Conception, l'Optimisation et la Production de Grenoble**
dans **l'École Doctorale I-MEP2 – Ingénierie, Matériaux, Mécanique, Environnement, Énergétique, Procédés, Production**

Optimisation Globale Déterministe Garantie sous Contraintes Algébriques et Différentielles par Morceaux

Guaranteed Deterministic Global Optimization using Constraint Programming through Algebraic, Functional and Piecewise Differential Constraints

Thèse soutenue publiquement le **19 janvier 2018**,
devant le jury composé de :

Remerciements

Une thèse c'est une histoire, une fraction de vie, une épreuve d'endurance avec des temps forts, des hauts et des bas. J'ai compris beaucoup de choses pendant ces quelques années, au-delà de l'aspect scientifique, j'ai appris sur le monde et sur moi-même. Je tiens à remercier Khaled Hadj-Hamou pour m'avoir proposé d'encadrer ma thèse et pour m'avoir permis de vivre cette aventure. Merci pour ta confiance et ta patience, pour ta capacité à trouver les mots justes et réconfortants lors des moments un peu plus difficiles, pour m'avoir convaincu que ce travail de recherche est bien une thèse, pour l'autonomie que tu m'as accordée et pour avoir laissé mon esprit vagabonder librement.

Je remercie Frédéric Messine et Gilles Trombettoni d'avoir accepté de rapporter ce travail. Je remercie également Alexandre Chapoutot pour sa participation à mon jury et Bernard Penz de m'avoir fait l'honneur de présider celui-ci. Je vous remercie de l'intérêt que vous avez porté à mon travail, d'avoir fait le déplacement pour participer à ma soutenance, de votre lecture approfondie de ce manuscrit, de la qualité de vos remarques et de nos échanges.

Je remercie le laboratoire G-SCOP pour son accueil ainsi que Marie Jo, Fadila, Myriam et les différents services pour l'organisation du laboratoire. En particulier, je remercie Marie Jo pour sa bienveillance, sa bonne humeur et ses nombreux encouragements.

Je remercie le laboratoire Nicta de m'avoir accueilli et Pascal Van Hentenryck de m'y avoir invité. Je remercie la région Rhône-Alpes qui a financé une partie de ce séjour par l'intermédiaire d'une bourse Explo'ra-doc.

Je remercie l'Université Joseph Fourier de m'avoir permis de découvrir l'univers de l'enseignement pendant deux années. En particulier, je remercie Xavier Giraud, Emeric Malevergne, Philippe Morat, Anne Rasse, Fabienne Carrier et Anne Guérin avec qui j'ai eu le plaisir de travailler, sans oublier les étudiants qui ont traversé mes salles de classe.

Je remercie les membres passés, présents et futurs de l'A-DOC pour les différentes activités organisées et à venir. Je salue chaleureusement Florence, Nicolas, Quentin avec qui nous avons animé cette association, ainsi qu'Amandine et Christine pour l'aide qu'elles nous ont apporté dans l'organisation des journées G-SCOP.

Je remercie mes compagnons d'infortune, du 5B où il fait chaud et du 2F où l'on s'entasse, qui ont accompagné ma traversée du laboratoire G-SCOP : Mauricio, Maud, Julien, Anne-Laure, Widad, Grigori, Wafa, Abduh Sayid et Juliana. Je suis très heureux d'avoir partagé quelques-uns de ces moments avec vous et je vous en remercie.

Je remercie de manière plus générale les membres passés, présents, temporaires et permanents pour la richesse scientifique du laboratoire G-SCOP au sein duquel il est très agréable d'évoluer. Je salue les joueurs de coinche et de backgammon, les adeptes de la pause café, et je remercie particulièrement : Florence, Lucas, Bertrand, Yohann, Pauline, Benjamin, Ingwild, Julie, Chloé, Lucie D., Lucile, Boris, Egor, Olga, Audrey, Laura, Issam, Marcel, Francielly, Michaël, Maxime, Bérangère, Khadija, Lamiae, Lucie P., Matthieu, Yacine, Tom, Sylvain, Quentin, Rémi, Clément, Nicolas B., Emmanuelle, Lisa, Justine, Élodie, Franck, Nadia, Nicolas C., Jean-Philippe, Olivier, Hadrien, Bernard, Vincent, Guillaume, Gilles et j'en oublie… Merci à Lucie D. pour les nombreuses fois

où nous avons nagé côte à côte, dans les méandres de nos réflexions profondes ou, plus simplement, à la piscine. Merci à Lucile pour avoir pailleté ma vie de doctorant. Merci à Gilles pour les sorties vélo et à Vincent pour les discussions passionnantes.

Je remercie la “dream-team” avec qui nous partagions quelques morceaux de craies sur les tableaux noirs de l'UFR IM^{2}AG en élaborant des plans de carrière alternatifs[1] :

- merci Florence, nous avons partagé beaucoup d'aventures et de projets au cours de ces dernières années : un master, une thèse, une année à l'A-DOC, une conférence, un challenge ROADEF, un article, etc. Nous avons été traversé par les mêmes doutes au cours des mêmes épreuves, puis un jour au détour d'une discussion, alors que tu venais de rendre ton manuscrit, mes peurs ce sont envolées et ma thèse s'est terminée. Je te remercie d'avoir partagé avec moi ces quelques aventures et ce lien particulier.

- merci Lucas, nous avons commencé par étudier un article[2] ensemble, court mais dense, puis nous avons coloré Léa. depuis ce jour tu parcours le monde des graphes à la manière de notre “randonnée bouquetin” dans le Trièves.

- merci Elizabeth pour les rires que nous avons partagés entre les cours, les révisions et pendant le projet d'allotissement. Maintenant que nous sommes tous docteurs il ne reste plus que toi.

Je remercie également mes amis, Marion, Eleonora, Amine, Emanuela, Coline, Márta, Gabrielle, Mélanie, Marc et en particulier Yasin, pour t'être déplacé pour ma soutenance, pour tous les TPs[3] et autres projets que nous avons partagé. Il y'a un peu de toi dans cette thèse.

I would like to thanks all the people that crossed my road when I was living in Canberra. In particular, thanks to my ex-flatmates: David, Kate and Rebecca, it was a great pleasure to share those moments with you. A special and emotional thanks to you Marina, for all these talks and adventures that we shared. Also thanks to you to Franco, for sharing your office at Nicta, the games of chess and all these parties.

L'aboutissement de ce manuscrit trouve des origines antérieures au début même de ma thèse. A ce titre, je remercie donc Khaled Hadj-Hamou, Jean Bigeon, Philippe Bidinger, Anne Rasse, Jean Marc Vincent, Laurent Mounier et Michaël Périn qui m'ont fait découvrir toujours un peu plus, lors de différents stages, le monde de la recherche académique. Je remercie également Nadia Brauner qui m'a fait découvrir la Recherche Opérationnelle au cours de ma première année de master et Christophe Luciani pour m'avoir fait aimer les Mathématiques lors de mon lycée.

Un mot pour mes parents, Vincent et Francoise, que je ne remercierai jamais assez d'avoir cru en moi, de m'avoir poussé, soutenu et encouragé toutes ces années. Grace à

[1]Maintenant que nous avons un peu de temps libre, c'est le moment de commencer à chercher un endroit où nous pourrions nous installer.

[2]Alexander Schrijver, A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time, Journal of Combinatorial Theory, Series B, Volume 80, Issue 2, 2000, Pages 346-355, ISSN 0095-8956, https://doi.org/10.1006/jctb.2000.1989.

[3]Je n'oublierai pas que les pingouins volent

vous je traverse mon existence outillé des savoirs que vous m'avez légués. Je remercie également Sylvie, Jean-Pierre, Gilles, Dominique et Pierre pour m'avoir accompagné ces dernières années.

Je remercie mon frère, Thomas, pour son soutien, sa bienveillance et son absence de jugement, pour nos soirées jeux[4], nos débats endiablés et son humour qui m'ont permis de voir bien au-delà des difficultés apparentes. Je le remercie également d'avoir pris du temps pour participer très activement à la finalisation de ce manuscrit via ses nombreuses relectures et propositions de réécritures.

Enfin, comment ne pas remercier Marie, qui m'a accompagné jour et nuit sur ce parcours. Je te remercie pour la patience et l'écoute que tu as montré chaque jour. Un nouveau départ s'annonce à présent, une aventure de plus à partager, pleine de bonheur et de surprises.

[4]Sacré Ryan!

Preface

The work that is introduced in this thesis takes its roots in a previous work that was done at the end of my *Master de Recherche* [Joudrier, 2013]. Following a brief introduction, the state of the art is detailed - it mentions and defines all of the prerequisite concepts that one should be familiar with in the following chapters. At the end of the thesis, the results are presented as well as the limits of the method. It also states how future works might originate from this thesis.

Also, within the scope of the ROADEF/EURO international challenge 2014 titled "Rolling Stock Unit Management on Railways sites" [Ramond and Nicolas, 2014], another research project has been studied in collaboration with Florence Thiard at the time of this research. However the methods that were applied to deal with this problem has little to do with the methods that are described in this thesis. The use of a specific arithmetic that would deal with sets of values and the use of constraint programming are two of these methods.

Unfortunately, because of the large amount of time this challenge required from us, the global and deterministic optimization method used in this thesis could not be applied on this project on time. However, some positive results allowed us to reach the first rank in the junior category of the contest and the all categories fourth rank. The results of this cooperation are presented in the appendix of this document and are published in the Annals of Operations Research [Joudrier and Thiard, 2017].

Contents

List of Figures

List of Tables

Part I

Introduction

In this thesis, a set of methods are presented in order to solve multi-physics dynamic problems with guarantees. These systems are largely used in fundamental research (physic, chemistry, biology, environment, etc) and they can be applied in various domains such that engineering design process, model of chemical reactions and electronic components, simulation of biological and economic systems or even to predict an athletic performance.

The resolution of these optimization problems is made of two stages that are named the model stage and the resolution stage. These two steps are interdependent because the resolution methods depend on the model that is used and the model is affected by the resolution processes that are considered. The model stage consists in defining the mathematical model by setting up the equations for the problem. It is made of a set of variables, a set of constraints and one or several cost functions that have to be minimized.

The engineering sciences require some tools to help the decision making through the development stages. Indeed, a lot of decisions (represented by a set of real variables $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^p$ in the model) made during design processes have a strong impact on final products. These decisions are restricted by the specifications that the final product has to respect. The specifications of the problems that are considered can be modeled with two sets of constraints.

These constraints link the decision variables $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^p$ with output secondary variables $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^s$ and output unary-functions $\mathbf{u} \in \mathcal{U} \subseteq (\mathbb{R} \to \mathbb{R})^n$ of the model.

- The decision variables $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^p$ are real variables that describe the system and allow us to modify its behavior. These variables evolve in a set of values $\mathcal{X} \subseteq \mathbb{R}^p$ that is the search space.

- The output secondary variables $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^s$ are variables that describe features on which it is not possible or desirable to act directly. However, these variable are used to control the properties that depend on the decision variables, the functional variables and the constraints.

- The output unary-functions $\mathbf{u} \in \mathcal{U} \subseteq (\mathbb{R} \to \mathbb{R})^n$ are functional variables which are used to control the dynamics of the system that result from the decision variables, the output secondary variables and the constraints.

The first set of constraints is the set of *Algebraic Constraints* Eq. (1). It usually stems from the engineering and is used to describe the components from a static point of view. These constraints are usually non-convex, sometimes non-continuous and can describe some properties (size, weight, density, volume, stiffness, etc) of one or a set of sub-components.

$$\mathbf{g}(\mathbf{x}, \mathbf{y}, \mathbf{u}) \leq 0 \tag{1}$$

where $\mathbf{g}$ is an application from $\mathcal{X} \times \mathcal{Y} \times \mathcal{U}$ to $\mathbb{R}^m$.

The second set of constraints is the set of *Differential Constraints* Eq. (2). They are used to describe some physico-chemical properties (electric, mechanical, electromagnetic, thermal, kinetics, etc), wear or any modelable features evolving with the component use. These constraints are defined by *Ordinary Differential Equations* (ODEs).

$$\mathbf{u}'(t) = \mathbf{d}(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) \tag{2}$$

where $\mathbf{d}$ is an application from $\mathcal{X} \times \mathcal{Y} \times \mathrm{Im}(\mathcal{U})$ (with $\mathrm{Im}(\mathcal{U}) = \mathbb{R}^n$) to $\mathbb{R}^n$ that describes the behavior of $\mathbf{u}$ and t in $\mathbb{R}$ is a descriptive variable that represents the state of progress of the physical components or the chemical reactions.

The combination of the algebraic constraints and the differential constraints makes possible the description of dynamical components and bound a feasible solution set $\mathcal{S}_{\mathcal{X}}$ Eq. (3). A feasible solution $\tilde{\mathbf{s}}_{\mathbf{x}}$ in $\mathcal{S}_{\mathcal{X}}$ is a value in $\mathcal{X} \times \mathcal{Y} \times \mathcal{U}$ such as all the constraints Eq. (1) and (2) from the model are satisfied.

$$\mathcal{S}_{\mathcal{X}} = \left\{ (\mathbf{x}, \mathbf{y_x}, \mathbf{u_x}) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{U} \;\middle|\; \begin{array}{c} \mathbf{g}(\mathbf{x}, \mathbf{y_x}, \mathbf{u_x}) \leq 0 \\ \mathbf{d}(\mathbf{x}, \mathbf{y_x}, \mathbf{u_x}(t)) = \mathbf{u}'(t) \end{array} \right\} \tag{3}$$

While the sets of constraints bound the model and draw the set of solutions $\mathcal{S}_{\mathcal{X}}$ that respect the specifications, the optimization is driven by a cost function which has to be minimized Eq. (4). The cost function $\mathbf{f_{cost}}$ is used to measure the quality of a feasible solution for the final components depending on one or several criteria (volume, weight, environmental impact factor, reaction rate, energy produced, etc). It could mean that these features are sensitive but do not require to be strongly controlled by some algebraic constraints.

$$\textit{Minimize } \mathbf{f_{cost}}(\mathbf{x}, \mathbf{y}, \mathbf{u}) \tag{4}$$

Where $\mathbf{f_{cost}}$ is a function from $\mathcal{X} \times \mathcal{Y} \times \mathcal{U}$ to $\mathbb{R}^k$. Depending on the value of k, single and multi-objectives optimization problems can be considered indiscriminately. The multi-objective optimization is motivated by handling some conflicting objectives like

- maximize a population and minimize its impact on the environment,
- maximize an athlete performance and minimize the energy it requires,
- maximize the quality of a product and minimize its production cost,
- maximize the temperature of a room and minimize the energy expenditure,

because design problems often involve multiple criteria such as investment, profit, quality, efficiency, operation time, etc.

The cost function provides a partial order on the feasible solution set $\mathcal{S}_{\mathcal{X}}$ and allows us to compare some of them. An optimal solution from an optimization problem is a set of values $\mathbf{s}^*$ that is feasible ($\mathbf{s}^*$ is an element of $\mathcal{S}_{\mathcal{X}}$) and minimal according to the cost function $\mathbf{f_{cost}}$. Then a *Dominant Set* Eq. (5) can be formed gathering all the optimal solutions:

$$\mathcal{D}_{\mathcal{X}} = \{(\mathbf{x}^*, \mathbf{y}^*, \mathbf{u}^*) = \mathbf{s}^* \in \mathcal{S}_{\mathcal{X}} \mid \nexists \tilde{\mathbf{s}}_{\mathbf{x}} \in \mathcal{S}_{\mathcal{X}},\ \mathbf{f_{cost}}(\tilde{\mathbf{s}}_{\mathbf{x}}) < \mathbf{f_{cost}}(\mathbf{s}^*)\} \tag{5}$$

The heterogeneity of models that are considered drives them hardly solvable. It comes from the combination and the complexity of the different types of constraints (algebraic, differential), their attributes (non-continuous, non-convex) and the variety of the rough sizes involved in the problem description. Moreover, when the cost function is non-regular, a large number of local minima exists which makes finding the optimal solution more difficult.

In addition to this complexity, some special differential constraints are defined in the dynamic design problems that are considered. The specificity of this new type of constraints comes from the piecewise differential description Eq. (6) which makes them more hardly solvable than classic differential constraints.

$$\mathbf{u}'(t) = \begin{cases} \mathbf{d_1}(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) & \text{if} \quad c_1(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) \\ & \vdots \\ \mathbf{d_q}(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) & \text{if} \quad c_q(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) \end{cases} \tag{6}$$

where for all i from 1 to q, $\mathbf{d_i}$ from $\mathcal{X} \times \mathcal{Y} \times \mathrm{Im}(\mathcal{U})$ to $\mathbb{R}^n$ is the description of the i-th dynamical behavior guarded by the Boolean function c_i from $\mathcal{X} \times \mathcal{Y} \times \mathrm{Im}(\mathcal{U})$ to $\mathbb{B}$ ($\mathbb{B}$ is the Boolean set $\{False, True\}$).

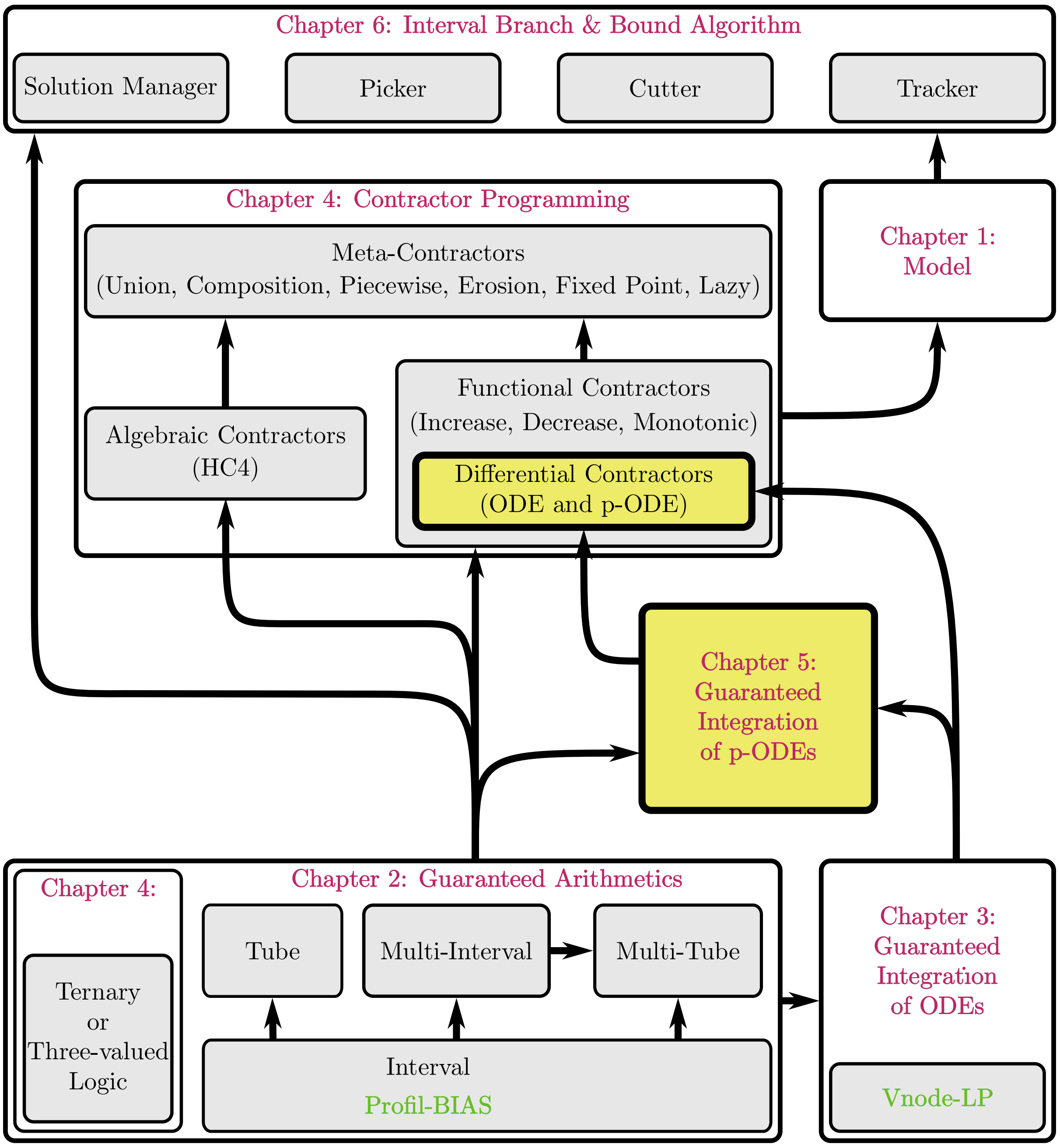
These constraints are useful to model some dynamic behaviors with some changing phases depending on the state of the components. As an example, this formalism makes it possible to model a mechanical component with a first dynamic describing its motion, and a second one that depends on a set of states reached through the first dynamic. Note that this kind of dynamic systems can be viewed as a peculiar case of hybrid systems [Ábrahám and Schupp, 2012].

The second stage that is the optimization consists in using a set of methods in order to extract one or all the solutions from the model. However, the complexity of these problems as well as the limits of the existing tools prevent the use of guaranteed global optimization methods. Consequently, for each of these two phases, different levels of approximation are presented in the scientific literature.

In some cases, engineering design problems are modeled and solved. They are many approximation levels of the model, from the sidelining of the dynamic part to the convexification. But the dynamic modelization is an important part of these problems and the solutions extracted from these approximated models are not relevant in use. Also, some approximated methods can be used in the solving process. About the optimization we could tackle evolutionary algorithms [Coello et al., 2002] that do not guarantee the optimality of the solution. Numerical approaches like numerical integration used to integrate differential equations [Cartwright and Piro, 1992] and propagate these constraints inside the optimization algorithm do not certify the feasibility of the solution. However, it gives some good results in practice but it does not guarantee the feasibility or the optimality of the solutions.

That are the reasons why a global optimization tool, that could solve these problems while trying to guarantee that the solutions are feasible and optimal, has been developed during this research work. The software named GDODynS *(Global and Deterministic*

Optimization for Dynamic Systems) has been implemented in C++, and its architecture (see Fig. 0) is based on several modules that are detailed in the next chapters.



Chapter 6: Interval Branch & Bound Algorithm
Solution Manager
Picker
Cutter
Tracker
Chapter 4: Contractor Programming
Meta-Contractors (Union, Composition, Piecewise, Erosion, Fixed Point, Lazy)
Chapter 1: Model
Functional Contractors (Increase, Decrease, Monotonic)
Algebraic Contractors (HC4)
Differential Contractors (ODE and p-ODE)
Chapter 5: Guaranteed Integration of p-ODEs
Chapter 4:
Ternary or Three-valued Logic
Chapter 2: Guaranteed Arithmetics
Tube
Multi-Interval
Multi-Tube
Interval
Profil-BIAS
Chapter 3: Guaranteed Integration of ODEs
Vnode-LP

Figure 0: Architecture of the optimisation tool developed: GDODynS

Following this introduction, the model of the problems that are considered in this thesis is detailed in Chapter 1 with two design problems, a constant magnetic engine [Kone et al., 1993] and an electromagnetic contactor [Mazhoud, 2014], that come from industrial applications.

In Chapter 2, the guaranteed arithmetics are introduced with the interval arithmetic [Moore, 1966] and the tube arithmetic [Aubry et al., 2013].

- The interval arithmetic (IA) is a specific arithmetic that is used to guarantee the results of numerical operations as well as to represent continuous sets of values. At first, the guarantee is assured by representing the real values with the smallest numerically representable intervals containing them. In a second step, the guarantee is preserved by redefining the arithmetic operators and the usual functions with a strict control on the rounding of the computation unit that provides rigorous bounds for the results of operations. In this chapter, the benefits as well as the drawbacks of the interval arithmetic are discussed.

- The tube arithmetic (TA) is an extension of the interval arithmetic that is used to describe sets of unary functions and to extend the guarantee of numerical operations on these functions. In this chapter several representations of tubes are presented and detailed.

Chapter 3 is dedicated to the guaranteed integration of ordinary differential equations (ODEs) in order to deal with the differential constraints defined in the model. In this chapter an iterative algorithm is studied that is made of two steps that compute first a global enclosure of the solution and then its contraction inside a local enclosure. Several methods are exposed in order to reduce the enclosure and decrease the computation time it requires. In particular, two methods based on the Picard's operator (FOE and HOE [Nedialkov et al., 2001]) are developed to compute the global enclosure. Also, Taylor series with mean value forms [Nedialkov et al., 1999], modifications of the coordinate systems using QR-Factorization [Householder, 1958, Lohner, 1987] and pruning methods based on consistency properties [Deville et al., 2000] are detailed to compute the local enclosures.

Bases of Constraint Programming (CP) are introduced in Chapter 4 to solve Constraint Satisfaction Problems (CSP) with filtering and contraction methods. These problems consist in finding one or several feasible solutions. The algorithms developed in this chapter are used to provide a faster convergence toward feasible domains that restrain the search space by using the constraints. Atomic contractors based on the hull-consistency [Benhamou et al., 1999] like $HC4$ are detailed to deal with algebraic constraints and the methods from Chapters 3 and 5 are used to develop specific contractors for differential constraints. Peculiar contractor schemes dedicated to specific set of constraints are introduced with meta-contractors through contractor programming [Chabert and Jaulin, 2009a].

In Chapter 5, the issues related to the guaranteed integration of piecewise differential equations (p-ODEs) are discussed and the specificities of the developed guaranteed integration method is detailed. The algorithm and the methods applied on the guaranteed integration of ODEs from Chapter 3 are limited by peculiar areas when applied on p-ODEs. Therefore, a theorem is proposed with its proof and is used to create a guaranteed integration algorithm for those specific areas.

All the methods previously described are integrated in Chapter 6 within a deterministic global optimization algorithm. It is based on an interval extension of the Branch and Bound algorithm (IBBA) [Messine et al., 2001] using interval constraint propagation techniques [Messine, 2004] with contactor programming from Chapter 4. Also, in this chapter, a paradoxical limit of the guaranteed optimization that is the loss of the guarantee for systems containing related constraints is detailed.

In Chapter 7, the methods described in this thesis are applied on different systems. First, the guaranteed optimization algorithm is applied on the industrial problem that is detailed in the second chapter. Because of this first problem does not contain any differential constraint, the guaranteed integration algorithm detailed in Chapters 3 and 5 is applied on a specific piecewise ordinary differential equations. The global optimization algorithm from Chapter 6 is applied on an optimization problem that contains piecewise differential constraints.

The final part summarizes the work presented in this thesis and broaches some perspectives on this topic.

Model 1

What distinguishes a mathematical model from, say, a poem, a song, a portrait or any other kind of "model", is that the mathematical model is an image or picture of reality painted with logical symbols instead of with words, sounds or watercolors.

– John L. Casti

Contents

Abstract

In this chapter, the modelling formalism that is considered in this thesis is introduced in the first section (Sect. 1.1) and then two models are introduced as examples. The first one is detailed in Section 1.2. It models the design problem of a constant magnetic engine. The second model describes the design problem of an electromagnetic contactor and is introduced in Section 1.3. This model is more complex because it contains a specific piecewise dynamic constraint that is difficult to integrate with rigorous bounds.

The guaranteed optimization process applied on this specific model motivated at first the research work presented in this thesis.

1.1 Generic Model

The goal of the work presented in this thesis is to optimize with guarantee the results of complex dynamic problems that can be modeled such as the following generic model:

$$\text{Minimize}_{\mathbf{x}\in\mathcal{X}} \quad \mathbf{f_{cost}}(\mathbf{x},\mathbf{y},\mathbf{u}) \tag{1.1}$$

$$s.c \quad \mathbf{g}(\mathbf{x},\mathbf{y},\mathbf{u}) \leq 0 \tag{1.2}$$

$$\mathbf{u}'(t) = \begin{cases} \mathbf{d_1}(\mathbf{x},\mathbf{y},\mathbf{u}(t)) & \textbf{if} \quad c_1(\mathbf{x},\mathbf{y},\mathbf{u}(t)) \\ & \vdots \\ \mathbf{d_q}(\mathbf{x},\mathbf{y},\mathbf{u}(t) & \textbf{if} \quad c_q(\mathbf{x},\mathbf{y},\mathbf{u}(t)) \end{cases} \tag{1.3}$$

where

- $\mathbf{x}$ is the vector of decision variables (input) in $\mathcal{X} \subseteq \mathbb{R}^p$.
- $\mathbf{y}$ is the vector of control variables (output) in $\mathcal{Y} \subseteq \mathbb{R}^s$.
- $\mathbf{u}$ is the vector of dynamic variables (output unary functions) in $\mathcal{U} \subseteq (\mathbb{R} \to \mathbb{R})^n$.
- $\mathbf{f_{cost}}$ is a function from $\mathcal{X} \times \mathcal{Y} \times \mathcal{U}$ to $\mathbb{R}^k$.
- $\mathbf{g}$ is a function from $\mathcal{X} \times \mathcal{Y} \times \mathcal{U}$ to $\mathbb{R}^m$.
- $\mathbf{d_i}$ are functions from $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}^n$ to $\mathbb{R}^n$.
- c_i are Boolean functions $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}^n$ to $\mathbb{B}$.

The existence of solutions and the guarantee of the results provided by the tool GDO-DynS that has been implemented in this work are discussed in Sections 3.2.5 and 6.3.3.

Such a model makes possible to deal with a lot of various problems. In the next sections, two design problems are introduced. They are extracted from a previous work [Mazhoud, 2014] where the author studied interval optimization methods and the impact of the reformulation of these problems in order to get an optimal model to optimize such problems. The research work presented in this thesis follows this previous work and focuses on the specific use of guaranteed methods to solve the dynamic parts of the problems.

1.2 Design of a constant magnetic engine

The first problem is the design of a magnetic engine. This problem does not contain dynamic parts. However, it is composed of a set of constraints to model several physical aspects: electric, magnetic, mechanical and thermal from a static point of view. The problem is multi-physic and the constraints model the relationship between the electromagnetism and the flow conservation [Kone et al., 1993] [Messine et al., 1998].

The model is composed of 17 parameters

- 10 decision variables (Table 1.1)
- 7 fixed parameters (Table 1.2)

and 6 algebraic constraints that are non-linear and non-convex.

Label	Definition	Range value
D	The cylinder bore	$[0.001; 0.5]$
B_e	The magnetic field inside the air-gap	$[0.1; 1]$
K_f	The leak coefficient	$[0.01; 0.5]$
J_{cu}	The power density inside the copper	$[10^5; 10^7]$
e	The mechanical air-gap	$[0.0001; 0.005]$
l_a	The thickness of the magnets	$[0.001; 0.05]$
E	The thickness of the winding	$[0.001; 0.05]$
C	The thickness of the cylinder heads	$[0.001; 0.05]$
β	The polar arc coefficient	$[0.8; 1]$
λ	The shape factor of the device	$[1; 2.5]$

Table 1.1: Parameters and range values

Label	Definition	Value
k_r	The filling coefficient of the winding	0.7
B_{fer}	The induction of the iron	1.5
E_{ch}	The heating of the device	10^{11}
Γ_{em}	The electromagnetic torque	10
M	The magnetisation of the magnets	0.9
p	The number of pole couple	4
Δ_p	The double polar step	0.1

Table 1.2: Fixed values

Because of the bill of specifications, the values of e and K_f must respect these additional constraints:

$$e \geq 0.001$$
$$K_f \leq 0.3$$

The magnetic torque Γ_{em}, the heating of the engine E_{ch}, the magnetic leak coefficient K_f, the magnetic field B_e inside the air-gap, the thickness of the cylinder heads C and the number of pole couples p are computed and controlled by the following equations

$$\Gamma_{em} = \frac{\pi}{2\lambda}(1-K_f)\sqrt{k_r.\beta.E_{ch}.E}D^2(D+E)B_e \tag{1.4}$$
$$E_{ch} = k_r.E.J_{cu}^2 \tag{1.5}$$
$$K_f = 1.5p\beta\frac{e+E}{D} \tag{1.6}$$
$$B_e = \frac{2l_aM}{Dlog\left[\frac{D+2E}{D-2(l_a+e)}\right]} \tag{1.7}$$
$$C = \frac{\pi\beta B_e}{4pB_{fer}}D \tag{1.8}$$
$$p = \frac{\pi D}{\Delta_p} \tag{1.9}$$

More details of this model can be found in [Kone et al., 1993]. The optimization problem is to minimize the volume of the active parts of the device.

$$V_u = \pi\frac{D}{\lambda}(D+E-e-l_a)(2C+E+e+l_a) \tag{1.10}$$
$$V_a = \pi\beta l_a\frac{D}{\lambda}(D-2e-l_a) \tag{1.11}$$
$$P_j = \pi\rho_{cu}\frac{D}{\lambda}(D+E)E_{ch} \tag{1.12}$$

where $\rho_{cu} = 17.10^{-9}$ is the electrical resistivity of the copper. The search space is continuous and is detailed in the previous table (Table 1.1 and 1.2).

The reformulation given by [Mazhoud, 2014] reduces the complexity of the model by decreasing the number of decision variables. From 10 decision variables in the initial model, the new formulation only contains 4 decision variables (the other variables depend on these 4 variables) that are detailed in Table 1.3.

Label	Definition	Range value
e	The mechanical air-gap	$[0.0001; 0.005]$
l_a	The thickness of the magnets	$[0.001; 0.05]$
E	The thickness of the winding	$[0.001; 0.05]$
β	The polar arc coefficient	$[0.8; 1]$

Table 1.3: Parameters and range values

Consequently, the model is rewritten as follows:

$$\text{Minimize } V_u = \frac{\pi D}{\lambda}(D + E - e - l_a)(2C + E + e + l_a);$$

$$s.c \begin{cases} D = \dfrac{p\Delta_p}{\pi} \\ J_{cu} = \sqrt{\dfrac{E_{ch}}{k_r E}} \\ B_e = \dfrac{2l_a M}{Dlog\left(\frac{D+2E}{D-2(l_a+e)}\right)} \\ K_f = 1.5p\beta\dfrac{(e+E)}{D} \\ \lambda = \dfrac{\pi}{2\times\Gamma_{em}}(1-K_f)D^2(D+E)B_e\sqrt{k_r\beta E_{ch}E} \\ C = \pi\beta\dfrac{B_e}{4pB_{fer}}D \end{cases}$$

1.3 Design of an electromagnetic contactor

The electromagnetic contactor is an industrial device made of a coil powered by an electric circuit and a magnetic circuit which is composed of two air gaps made of iron. One of the air-gap is fixed when the other part is free to move.

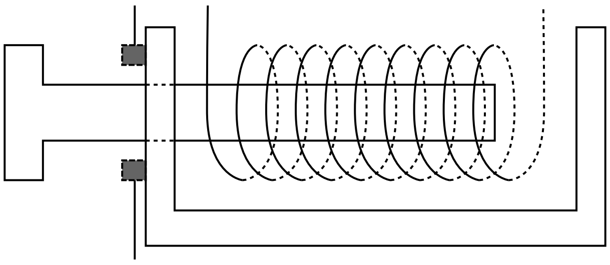


Figure 1.1: Design of the electromagnetic contactor - open position

On figure 1.1 the electromagnetic contactor is opened. The kernel of the device closes the contactor with the electromagnetic field that is produced, when the coil is powered, which attracts the hammer. The consequence is that it closes the electric circuit on the left (Fig. 1.2).

Because of the online command on the thyristor, the signal that gives power to the coil is not known beforehand. Moreover, the coil resistor is significant and has to be added on the equivalent electrical circuit.

The air-gap consists in a cut in a magnetic circuit. It increases the reluctance of the circuit and increases the power that overloads the circuit as well. Also, it assures the movement of a part of the circuit, for example on stator/rotor devices.

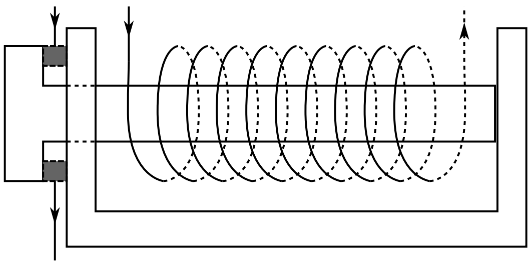


Figure 1.2: Design of the electromagnetic contactor - closed position

This device can be modeled with the following set of equations in order to model the electrical behavior Eq. (1.13) and the mechanical behavior Eq. (1.14) of the component. Note that some coefficients of the equations are variables.

$$U(t) = R \times i(t) + \frac{\partial}{\partial t}(L(i(t), z(t)) \times i(t)) \tag{1.13}$$

$$F(i(t), z(t)) = m\frac{d^2 z(t)}{dt^2} \tag{1.14}$$

where i is the power, z is the length of the air gap, R is the resistor of the coil, L is the induction of the coil, m is the weight of the kernel and F is the force applied on the kernel which is a composition of the magnetic force, the friction and the electric charge. Note that the magnetic behavior is not modeled in the following equations because it requires to compute the force and the induction which are functions of the power in the coil and the width of the air-gap. It turns out to do the assumption that the electromagnetic phenomena are fast compared to the electrical and mechanical time constants.

Because of the need of robustness and safety in the use of the contactor in the industry, some additional constraints are required. First the contactor must be lightning-struck resistant, then it has to accept any voltage from 50 to 400 volts and finally the closing of the magnetic component must happen in less than 30ms without any damage.

The goal of the optimization problem associated to the device is to minimize its global volume while respecting all the previous constraints.

The system is modeled with a reluctance network that consists in 37 non linear algebraic constraints, a large multi-dimensional piecewise ODE system and 61 real and functional variables. Because of practical reasons and a non-disclosure agreement, this model is not detailed in this thesis.

This design problem can be modeled from a static point of view, but the length of the air-gap varies in relation with the use of the component and it impacts the electric behavior as well as the mechanical behavior. Indeed, when the length $z(t)$ changes with time t, the value of $L(i(t), z(t))$ changes as well as $U(t)$ and the mechanical force $F(i(t), z(t))$. That is why it is more relevant to solve this problem from a dynamic point of view.

However, the dynamic model is difficult to solve, because the motion of the mechanical part is stopped when the air-gap is closed. This is the break in the dynamic behavior of the component and this is what motivated the work presented in this thesis. Two approaches can be considered in order to describe the dynamic model.

- The first model is based on an hybrid system in which the state machine is made of two states. One state is used to model the closure action and the other one is used to model the device when the air-gap is closed.
- The second model is based on a piecewise differential equation in which the first piece is used to model the system when the air-gap is positive and the second piece is used to model the system when the air-gap is negative.

The first model is closer to the reality but the behavior of the component after the closure is not important. Moreover the second model has been studied in a previous work [Mazhoud, 2014] in which the dynamic parts were not solved using guaranteed integration methods. It motivated this work through the development of guaranteed methods in order to tackle such constraints.

Part II
Literature Review

2 Guaranteed Arithmetics

- And you guarantee that they don't deal the drugs in those neighborhoods?
- I don't guarantee that. I guarantee that I'll kill anyone who does.

– The Godfather: Part III

Contents

2.1 Introduction

This chapter introduces basic notions of guaranteed arithmetics. They have been developed in order to quantify the errors of numerical computations. Because the computed results are often rounded to the nearest representable values and because computers have become faster gradually, thus increasing the rate of operations number computable in one time unit, the global errors rate increases at the same time without any control over it.

Through linear and non-linear systems of equations [Neumaier, 1990], path planning, rolling stock unit management (Appendix A), and the evolution of dynamic systems, guaranteed arithmetics are a robust counterpart to the probabilistic approaches usually used to solve these complex problems.

This chapter is organized as followed: Section 2.2 introduces the basics of interval arithmetic and Section 2.3 presents its evolution towards the multi-interval arithmetic. A different approach is briefly described in Section 2.4 via the affine arithmetic. These first sections are used to model sets of real values. Ultimately, in order to represent and deal with sets of functions, Section 2.5 defines the tube arithmetic which is an extension of these first concepts.

2.2 Interval Arithmetic

The main aspect of the interval arithmetic [Moore, 1966] is the use of interval representation instead of using real numbers. Values and results of operations (noted $\bullet$) are bounded by controlling the rounding (up $\bullet^{\uparrow}$, down $\bullet_{\downarrow}$) of the unit core computing and by redefining basic operations and functions.

The interval arithmetic makes possible the manipulation of uncertainty on parameters and the representation of random variables with imprecise probability density functions. It is possible to induce some reasoning on a continuous set of values without having to enumerate all of them. It is relevant when dealing with global optimization over real inputs.

2.2.1 Definitions

The set of intervals $\mathbb{IR}$ on the real line $\mathbb{R}$ is defined as below:

$$\mathbb{IR} := \left\{ [\underline{a}; \overline{a}] \,\middle|\, \begin{array}{c} (\underline{a}, \overline{a}) \in \mathbb{R}^2 \\ \underline{a} \leq \overline{a} \end{array} \right\}$$

The lighter notation $[a] = [\underline{a}, \overline{a}]$ is used. Let $[x] = [\underline{x}, \overline{x}]$ a non-empty interval with floating point bounds ($[x] \in \mathbb{I}\mathcal{F} \subset \mathbb{IR}$). It defines the following set:

$$[\underline{x}, \overline{x}] := \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\} \tag{2.1}$$

The middle Eq. (2.3), width Eq. (2.2) and the interior Eq. (2.5) of the non-empty interval $[x]$ are defined by:

$$width([x]) := \overline{x} - \underline{x} \tag{2.2}$$

$$midpoint([x]) := \frac{\underline{x} + \overline{x}}{2} \tag{2.3}$$

$$magnitude([x]) := \max(|\underline{x}|, |\overline{x}|) \tag{2.4}$$

$$interior([x]) := \{x \in \mathbb{R} \mid \underline{x} < x < \overline{x}\} \tag{2.5}$$

The intersection and union of two non-empty intervals $[x]$ and $[y]$ are defined as below:

$$[x] \cap [y] = \begin{cases} [\max(\underline{x}, \underline{y}); \min(\overline{x}, \overline{y})] & \textbf{if } \max(\underline{x}, \underline{y}) \leq \min(\overline{x}, \overline{y}) \\ \emptyset & \textbf{otherwise} \end{cases} \tag{2.6}$$

$$[x] \cup [y] = [\min(\underline{x}, \underline{y}); \max(\overline{x}, \overline{y})] \tag{2.7}$$

The interval union defined here is the smallest interval hull of the simple *union*, meaning the simple union is bounded by the interval union although they are not necessarily equal.

2.2.1.1 Example: Some unions and intersections on intervals

$$[2; 4] \cap [6; 8] = \emptyset \qquad [3; 6] \cap [4; 7] = [4; 6]$$

$$[2; 4] \cup [6; 8] = [2; 8] \qquad [3; 6] \cup [4; 7] = [3; 7]$$

2.2.2 Arithmetic Operators and Usual Functions

In order to compute operations on $\mathbb{IR}$, arithmetic operators $\diamond$ and functions $f \in \{abs, cos, sin, tan, exp, log, power, root\}$ can be extended to the interval arithmetic Eq. (2.8) and (2.9). Let $[x]$ and $[y]$ two non-empty intervals, then:

$$[x] \diamond [y] = \{x \diamond y \mid x \in [x], y \in [y]\} \tag{2.8}$$

$$f([x]) = \{f(x) \mid x \in [x]\} \tag{2.9}$$

The operations redefined on these models Eq. (2.8) and (2.9) are inclusion monotonic. Let $[x]$, $[X]$, $[y]$ and $[Y]$ four intervals such that $[x] \subseteq [X]$ and $[y] \subseteq [Y]$, then

$$\begin{gathered} [x] \diamond [y] \subseteq [X] \diamond [Y] \\ f([x]) \subseteq f([X]) \end{gathered} \tag{2.10}$$

The basic binary arithmetic operators $\diamond \in \{+, -, \times, \div\}$ are redefined in interval arithmetic as below:

$$[x] + [y] = [\underline{x} +_{\downarrow} \underline{y}; \overline{x} +^{\uparrow} \overline{y}] \tag{2.11}$$

$$[x] - [y] = [\underline{x} -_{\downarrow} \overline{y}; \overline{x} -^{\uparrow} \underline{y}] \tag{2.12}$$

$$[x] \times [y] = [xy_{\downarrow}; xy^{\uparrow}] \textbf{ where } \begin{cases} xy_{\downarrow} = \min\{\underline{x} \times_{\downarrow} \underline{y}, \underline{x} \times_{\downarrow} \overline{y}, \overline{x} \times_{\downarrow} \underline{y}, \overline{x} \times_{\downarrow} \overline{y}\} \\ xy^{\uparrow} = \max\{\underline{x} \times^{\uparrow} \underline{y}, \underline{x} \times^{\uparrow} \overline{y}, \overline{x} \times^{\uparrow} \underline{y}, \overline{x} \times^{\uparrow} \overline{y}\} \end{cases} \tag{2.13}$$

$$\frac{[x]}{[y]} = [x] \times \frac{1}{[y]} = [x] \times [1 \div_{\downarrow} \overline{y}; 1 \div^{\uparrow} \underline{y}] \textbf{ with } 0 \notin [y] \tag{2.14}$$

Note that for all intervals $[x]$ and $[y]$ with $0 \in [y]$, the operation $\frac{[x]}{[y]}$ is strictly forbidden. The interval arithmetic is just allowed to represent closed intervals [Hansen, 1968], thus the segment $]0; 3]$ cannot be represented by interval, the extreme value 0 has to be included in the interval $[0; 3]$. The inversion is legal on the first segment $]0; 3]$ but illegal on its interval representation $[0; 3]$. In order to avoid this, an extension to the infinite bounds intervals of the interval arithmetic has been developed [Kahan, 1968] [Hansen and Walster, 2003]. A second version of the inversion can be used, which is not included in the IEEE 1788-2015 Standard for Interval Arithmetic, to compute with any interval containing 0.

$$\frac{1}{[y]} = \begin{cases} \emptyset & \textbf{if } [y] = [0; 0] \\ [1 \div_{\downarrow} \overline{y}; 1 \div^{\uparrow} \underline{y}] & \textbf{if } 0 \notin [y] \\ [1 \div_{\downarrow} \overline{y}; +\infty] & \textbf{if } \underline{y} = 0 \textbf{ and } \overline{y} > 0 \\ [-\infty; 1 \div^{\uparrow} \underline{y}] & \textbf{if } \underline{y} < 0 \textbf{ and } \overline{y} = 0 \\ [-\infty; +\infty] & \textbf{if } \underline{y} < 0 \textbf{ and } \overline{y} > 0 \end{cases} \tag{2.15}$$

2.2.2.1 Example: Some arithmetic operations

$$\begin{aligned}
[-1;3] + [5;6] &= [4;9] \\
[-1;3] - [5;6] &= [-7;-2] \\
[-1;3] \times [5;6] &= [-6;18] \\
[-1;3] \div [5;6] &= [-1/6;3/5]
\end{aligned}$$

Most of the usual unary functions are extended to the interval arithmetic exploiting their monotonic properties. Let $[x]$ a non-empty interval, then:

$$abs([x]) = \begin{cases} [x] & \textbf{if } 0 < \underline{x} \\ [-_{\downarrow}\overline{x}; -^{\uparrow}\underline{x}] & \textbf{if } \overline{x} < 0 \\ [0; \max(-^{\uparrow}\underline{x}, \overline{x})] & \textbf{otherwise} \end{cases} \tag{2.16}$$

$$exp([x]) = [exp_{\downarrow}([\underline{x}]); exp^{\uparrow}([\overline{x}])] \tag{2.17}$$

$$log([x]) = [log_{\downarrow}([\underline{x}]); log^{\uparrow}([\overline{x}])] \textbf{ if } 0 < \underline{x} \tag{2.18}$$

When functions are not monotonic they should be considered as piecewise monotonic. Therefore each monotonic piece of the functions is detailed.

2.2.2.2 Example:

Because the trigonometric functions *cosine* and *sine* are non-monotonic on the interval $[-\pi, \pi]$, then

$$\left.\begin{aligned} cos([-\pi,\pi]) &= [-1;1] \\ [cos_{\downarrow}(-\pi); cos^{\uparrow}(\pi)] &= [-1;-1] \end{aligned}\right\} \Rightarrow cos([-\pi,\pi]) \neq [cos_{\downarrow}(-\pi); cos^{\uparrow}(\pi)]$$

However, the function *cosine* is piecewise monotonic, increasing on $[-\pi;0]$ and decreasing on $[0;\pi]$.

$$\begin{aligned}
cos([-\pi,\pi]) &= cos([-\pi;0]) \cup cos([0;\pi]) \\
&= [cos_{\downarrow}(-\pi); cos^{\uparrow}(0)] \cup [cos_{\downarrow}(\pi); cos^{\uparrow}(0)] \\
&= [-1;1] \cup [-1;1] = [-1;1]
\end{aligned}$$

2.2.3 Boxes or Interval Vectors

An interval vector (box) $[\mathbf{x}]$ of $\mathbb{R}^n$ is the direct product of n intervals. The set of all interval vectors of $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$.

$$[\mathbf{x}] = ([x_1], \ldots, [x_n]) = ([\underline{x_1}; \overline{x_1}], \ldots, [\underline{x_n}; \overline{x_n}]) \tag{2.19}$$

A simple interval is denoted by $[x]$ instead of an interval vector $[\mathbf{x}]$. All operators and functions on simple intervals can be extended to boxes, by applying them on each component of the boxes.

Let $[\mathbf{x}]$ a box of size n. The bounds $\underline{\mathbf{x}}$, $\overline{\mathbf{x}}$ and the *width* are defined :

$$\underline{\mathbf{x}} = (\underline{x_1}, \ldots, \underline{x_n}) \tag{2.20}$$
$$\overline{\mathbf{x}} = (\overline{x_1}, \ldots, \overline{x_n}) \tag{2.21}$$
$$width([\mathbf{x}]) = \max_{1 \le i \le n} width([x_i]) \tag{2.22}$$

Some additional quantities can be defined on boxes such as the volume

$$volume([\mathbf{x}]) = \prod_{i=1}^{i \le n} width([x_i]) \tag{2.23}$$

The term box is used as well as interval vector or interval (which can be considered as a 1-dimensional interval vector).

2.2.4 Inclusion function

Considering a function $\mathbf{f}$ from $\mathbb{R}^n$ to $\mathbb{R}^m$ and intervals $[\mathbf{x}]$ in $\mathbb{IR}^n$, $range(\mathbf{f}, [\mathbf{x}])$ (Fig. 2.1) in $\mathbb{IR}^m$ is the minimal enclosure of the evaluation of $\mathbf{f}$ on $[\mathbf{x}]$.

$$range(\mathbf{f}, [\mathbf{x}]) = \{\mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in [\mathbf{x}]\} \tag{2.24}$$

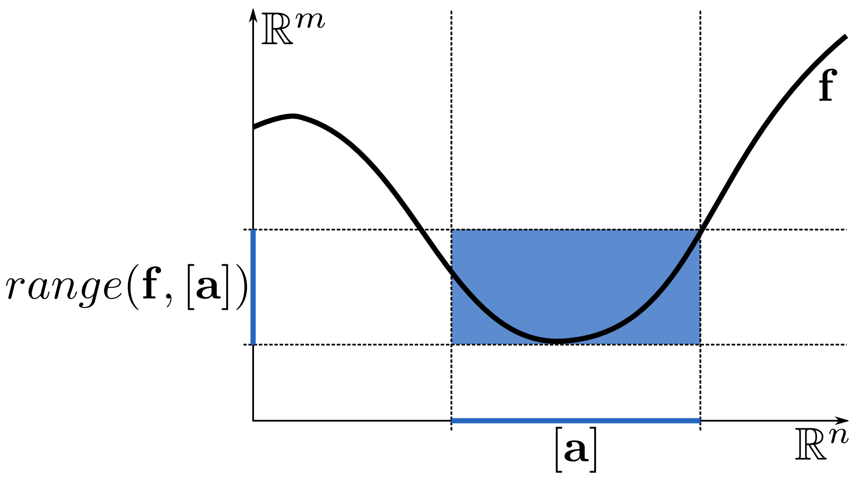


Figure 2.1: range of $\mathbf{f}$ over $[\mathbf{a}]$

Let $\mathbf{f}$ a function on the set of reals $\mathbb{R}^n$. To evaluate $\mathbf{f}$ over a box $[\mathbf{x}]$ from $\mathbb{IR}^n$ an inclusion function $[\mathbf{f}]$ of $\mathbf{f}$ has to be defined. An inclusion function has to bound the minimal enclosure to be valid on $[\mathbf{x}]$. In other words, $[\mathbf{f}]$ is valid iff:

$$range(\mathbf{f}, [\mathbf{x}]) \subseteq [\mathbf{f}]([\mathbf{x}]) \tag{2.25}$$

2.2.4.1 Definition: Inclusion monotonicity

The inclusion functions should be monotonic with the inclusion operator:

$$[\mathbf{x}] \subseteq [\mathbf{y}] \Rightarrow [\mathbf{f}]([\mathbf{x}]) \subseteq [\mathbf{f}]([\mathbf{y}]) \tag{2.26}$$

2.2.4.2 Remark:

For each function $\mathbf{f}$ an infinite number of inclusion functions $[\mathbf{f}]$ exist which are more or less accurate. It is a complex problem to find the best expression of the inclusion function $[\mathbf{f}]$ for a function $\mathbf{f}$ [Ratschek, 1985].

2.2.4.3 Definition: Hausdorff metric on intervals

To measure how far two intervals $[\mathbf{x}]$ and $[\mathbf{y}]$ are from each other, the set $\mathbb{IR}$ can be equipped with the Hausdorff metric $q([\mathbf{x}], [\mathbf{y}])$ [Henrikson, 1999], such that

$$\begin{aligned} q([\mathbf{x}], [\mathbf{y}]) &= \min \left\{ q \in \mathbb{R}^+ \;\middle|\; \begin{matrix} [\mathbf{x}] \subseteq [\mathbf{y}] + [-q; q] \\ [\mathbf{y}] \subseteq [\mathbf{x}] + [-q; q] \end{matrix} \right\} \\ &= max(|\underline{\mathbf{x}} - \underline{\mathbf{y}}|, |\overline{\mathbf{x}} - \overline{\mathbf{y}}|) \\ &= |midpoint([\mathbf{x}]) - midpoint([\mathbf{y}])| + \frac{|width([\mathbf{x}]) - width([\mathbf{y}])|}{2} \end{aligned} \tag{2.27}$$

When $[\mathbf{x}] \subseteq [\mathbf{y}]$ the Hausdorff distance computes the overestimation of $[\mathbf{y}]$ in relation to $[\mathbf{x}]$. Therefore, the Hausdorff distance can be used to compare and contrast the accuracy of several inclusion functions on the same expression.

2.2.4.4 Definition: Natural Inclusion Function

The natural inclusion function of a complex function $\mathbf{f}(x_1, \ldots, x_n)$ over intervals $[a_1] \ldots [a_n]$, denoted by $[\mathbf{f_N}]([a_1], \ldots, [a_n])$ is obtained by replacing all variables $(x_1 \ldots x_n)$ with their respective interval values $([a_1] \ldots [a_n])$ and by replacing all usual functions with their interval extensions. Natural inclusion functions provide the exact results not only when all the intervals are degenerate

$$\underline{\mathbf{x}} = \overline{\mathbf{x}} \quad \Rightarrow \quad [\mathbf{f_N}]([\mathbf{x}]) = range(\mathbf{f}, [\mathbf{x}]) = f([\mathbf{x}])$$

but also when all the non-degenerated intervals are paired with single occurrence variables from the expression $[\mathbf{f_N}]$.

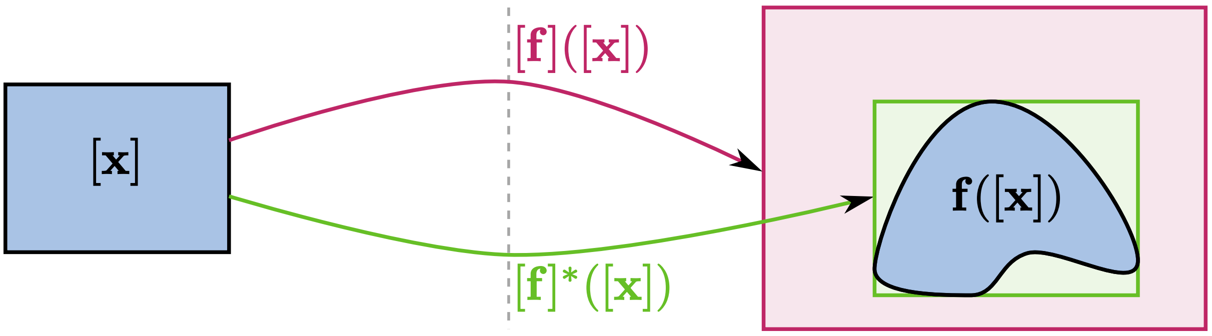


Figure 2.2: Image of the box $[\mathbf{x}]$ by the function $\mathbf{f}$, by the inclusion function $[\mathbf{f}]$ and by the minimal inclusion function $[\mathbf{f}]^*$

2.2.4.5 Definition: Convergence

Let f a function from $\mathcal{D}_f \subseteq \mathbb{R}^n$ to $\mathbb{R}$ and $[f]$ an inclusion function of f. This inclusion function $[f]$ has an order of convergence α through the Hausdorff distance q when for all $[\mathbf{x}] \subseteq \mathcal{D}_f$

$$q(range(f, [\mathbf{x}]), [f]([\mathbf{x}])) = \mathcal{O}(width([\mathbf{x}])^{\alpha}) \tag{2.28}$$

2.2.4.6 Property: Lipschitz condition and order of convergence

When the inclusion function satisfies a Lipschitz condition on $\mathcal{D} \subseteq \mathbb{R}^n$, it has a linear convergence for any $[\mathbf{x}] \subseteq \mathcal{D}$ ([Moore, 1966] and then [Neumaier, 1990]):

$$q(range(f, [\mathbf{x}]), [f]([\mathbf{x}])) = \mathcal{O}(width([\mathbf{x}])) \tag{2.29}$$

where q is the Hausdorff metric.

2.2.4.7 Definition: Mean value form [Moore, 1966]

Let f a function from $\mathbb{R}^n$ to $\mathbb{R}$, $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ and $[\mathbf{x}]$ an interval vector such that $\mathbf{x} \in [\mathbf{x}]$. If f is continuously differentiable on $\mathcal{X}$, then

$$f(\mathbf{x}) \in [f_M]([\mathbf{x}]) = f(\widehat{\mathbf{x}}) + [f']([\mathbf{x}] - \widehat{\mathbf{x}}) \tag{2.30}$$

where $\widehat{\mathbf{x}} \in [\mathbf{x}] \subseteq \mathcal{X}$.

2.2.4.8 Property: Quadratic convergence of the mean value form

When the inclusion function $[f']$ of the differential f' is interval lipschitzien, the first order inclusion function of the function f given by the mean value form checks that

$$q(range(f, [\mathbf{x}]), [f_M]([\mathbf{x}])) \leq \mathcal{O}(width([\mathbf{x}])^2) \tag{2.31}$$

Therefore the mean value form is a quadratically convergent approximation [Neumaier, 1990] and provides more relevant results than the natural inclusion functions. The distance between $range(f, [\mathbf{x}])$ and $[f_M]([\mathbf{x}])$ tends towards 0 as $width([\mathbf{x}])^2$, when $width([\mathbf{x}])$ tends towards 0.

2.2.4.9 Remark: The inclusion functions using mean value forms have a better convergence than natural inclusion functions. However, the natural form usually provides tighter enclosures when the size of the arguments is large. The mean value form is mostly used when the width of the intervals is small.

2.2.4.10 Definition: Integration of inclusion functions

Let $\mathbf{f}$ be a continuous function from $\mathbb{R}$ to $\mathbb{R}^m$ and $[a] \in \mathbb{IR}$ an interval. The function $\mathbf{f}$ can be integrated on the interval $[a]$, using the operators previously defined.

$$\int_{\underline{a}}^{\overline{a}} \mathbf{f}(x)dx = (\overline{a} - \underline{a}) \times range(\mathbf{f}, [a]) \in width([a]) \times [\mathbf{f}]([a]) \tag{2.32}$$

2.2.5 Evaluation

Let f the real function from $\mathbb{R}$ to $\mathbb{R}$ such as

$$f(x) = x^2 - 2x \tag{2.33}$$

and $[a]$ the interval $[2;5]$. The evaluation of f over $[a]$ denoted by $[f]([a])$ must bound the minimal enclosure $range(f,[a]) = range(f,[2;5]) = [0;15]$.

The natural evaluation of f over the interval $[a] = [2;5]$ is computed step by step as below:

$$[f]([2;5]) = [2;5]^2 - 2 \times [2;5] = [4;25] - [4;10] = [-6;21] \tag{2.34}$$

The evaluation of the inclusion function acquired by the natural extension to interval arithmetic over-approximates the exact result (which is its own limit). This limit comes from the presence of multi-occurrence variables in the explicit form of the inclusion function. As a consequence, when the evaluation is performed, the value of x can be considered equal to 2 in the first part of the expression x^2 and equal to 5 in the second part $-2x$ at the same time. In other words, all the occurrences of the same variable are independent during the evaluation step.

However, there are several ways to tackle this overestimation effect. The first one is to make a piecewise evaluation [Moore, 1966] (Fig. 2.3) computing many evaluations of f over sub-intervals (for example $\{[2;3],[3;4],[4;5]\}$ sub-intervals of $[2;5]$) thus strengthening the relationship between all the occurrences of the same variable.

$$\begin{aligned}[f]([2;3]) &= [-2;5]\\ [f]([3;4]) &= [1;10]\\ [f]([4;5]) &= [6;17]\end{aligned}$$

Then finally

$$[f]([2;5]) = [f]([2;3]) \cup [f]([3;4]) \cup [f]([4;5]) = [-2;17]$$

A second method is to rewrite the expression of the inclusion function $[f]$ into a syntactically different but mathematically equivalent function, in order to either set a limit to the number of occurrences of each variable or to reinforce their dependencies. Automatic methods exist such as using *Horner forms* or *Bernstein basis* [Martin et al., 2002] but the expressions they provide are usually sub-optimals. For example, on the following equations the expression of f_1 is the Horner form of the initial expression f while f_2 is a recast of the expression f such that the variable x becomes a single occurrence variable.

$$f(x) = x^2 - 2x \quad \Leftrightarrow \quad f_1(x) = x(x-2) \quad \Leftrightarrow \quad f_2(x) = (x-1)^2 - 1$$

$$\begin{aligned}[f_1]([2;5]) &= [2;5] \times ([2;5] - 2) = [2;5] \times [0;3] = [0;15]\\ [f_2]([2;5]) &= ([2;5] - 1)^2 - 1 = [1;4]^2 - 1 = [1;16] - 1 = [0;15]\end{aligned}$$

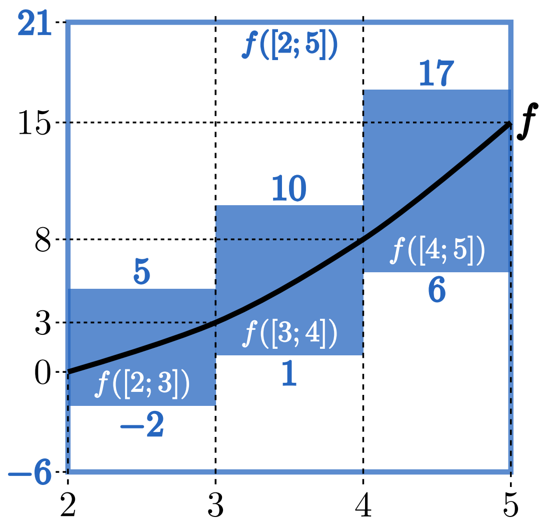


Figure 2.3: Piecewise evaluation of $f(x) = x^2 - 2x$ over $[0;5]$

Note that over $[a] = [2;5]$, the inclusion functions $[f_1]$ and $[f_2]$ are optimals. On the contrary, when the evaluation is computed over $[a] = [-1;1]$, the *Honer form* $[f_1]([-1;1]) = [-3;3]$ provides an enclosure that is larger than the *natural extension* $[f]([-1;1]) = [-2;3]$ while the inclusion function $[f_2]([-1;1]) = [-1;3]$ computes the exact enclosure.

2.2.6 Automatic Differentiation

Automatic differentiation [Wengert, 1964], [Linnainmaa, 1970] is a technique to evaluate the derivative of a function. It is based on the separation of the differential along the following chain rule:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w}\frac{\partial w}{\partial x}$$

Let f the function of two variables x_1, x_2 and its syntax tree composed by the set of nodes $\{w_1, \dots, w_6\}$ such as:

$$\begin{aligned} f(x_1,x_2) &= exp(x_1) + (x_1x_2)^2 \\ &= exp(w_1) + (w_1w_2)^2 \\ &= w_3 + w_4^2 \\ &= w_3 + w_5 \\ &= w_6 \end{aligned} \tag{2.35}$$

Two modes have been developed to compute the final derivative, the forward accumulation Eq. (2.36) and the reverse accumulation Eq. (2.37).

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial w_1}\frac{\partial w_1}{\partial x_1} = \frac{\partial f}{\partial w_1}\left(\frac{\partial w_1}{\partial w_2}\frac{\partial w_2}{\partial x_1}\right) = \frac{\partial f}{\partial w_1}\left(\frac{\partial w_1}{\partial w_2}\left(\frac{\partial w_2}{\partial w_3}\frac{\partial w_3}{\partial x_1}\right)\right) = \dots \tag{2.36}$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial w_1}\frac{\partial w_1}{\partial x_1} = \left(\frac{\partial f}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x_1} = \left(\left(\frac{\partial f}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x_1} = \dots \tag{2.37}$$

2.2.6.1 Forward Mode

The forward mode [Wengert, 1964] is easy to implement since the differential computing is performed during the evaluation of the syntax tree of the expression. The atomic differential values depend on the variable (x_1 or x_2) chosen to compute the derivative and are coupled with the leaves of the syntactic tree:

$$w'_1 = \frac{\partial w_1}{\partial x_1} = 1\ ,\ w'_2 = \frac{\partial w_2}{\partial x_1} = 0 \qquad \textbf{or} \qquad w'_1 = \frac{\partial w_1}{\partial x_2} = 0\ ,\ w'_2 = \frac{\partial w_2}{\partial x_2} = 1$$

Now the chain rule can be used to compute the derivative of the function f. Here the computation of the derivative with x_1 as the independent variable (in other words $\frac{\partial f}{\partial x_1}$), is introduced:

Syntactic tree	**Operations to compute derivative**	**Value of the derivative**
$w_1 = x_1$	$w'_1 = 1$	$w'_1 = 1$
$w_2 = x_2$	$w'_2 = 0$	$w'_2 = 0$
$w_3 = exp(w1)$	$w'_3 = w'_1 exp(w_1)$	$w'_3 = exp(x_1)$
$w_4 = w_1 w_2$	$w'_4 = w'_1 w_2 + w_1 w'_2$	$w'_4 = x_2$
$w_5 = w_4^2$	$w'_5 = 2w'_4 w_4$	$w'_5 = 2x_1 x_2^2$
$w_6 = w_3 + w_5$	$w'_6 = w'_3 + w'_5$	$w'_6 = exp(x_1) + 2x_1 x_2^2$

2.2.6.2 Reverse Mode

The reverse mode [Linnainmaa, 1970] is the exact opposite of the forward accumulation. In this method, the derivative is computed from the root of the syntactic tree to the leaves, but cannot be expressed explicitly from the root. Then a new quantity, the adjoint $\overline{w_i}$, is used for each node of the tree to represent the differential of the expression f over the subexpression w_i.

$$\overline{w_i} = \frac{\partial f}{\partial w_i} \tag{2.38}$$

The set of all the adjoints can be computed with the following operations:

$$\begin{aligned}
\overline{f} &= \overline{w_6} = 1 \\
\overline{w_5} &= \overline{w_6}\frac{\partial w_6}{\partial w_5} = \overline{w_6} \\
\overline{w_4} &= \overline{w_5}\frac{\partial w_5}{\partial w_4} = 2w_4\overline{w_5} \\
\overline{w_3} &= \overline{w_6}\frac{\partial w_6}{\partial w_3} = \overline{w_6} \\
\overline{w_2} &= \overline{w_4}\frac{\partial w_4}{\partial w_2} = w_1\overline{w_4} \\
\overline{w_1} &= \overline{w_3}\frac{\partial w_3}{\partial w_1} + \overline{w_4}\frac{\partial w_4}{\partial w_1} = exp(w_1)\overline{w_3} + w_2\overline{w_4}
\end{aligned}$$

By substitution, we finally get

$$\begin{aligned}\frac{\partial f}{\partial x_1} &= \overline{w_1} = exp(w_1) + 2w_1w_2^2 = exp(x_1) + 2x_1x_2^2\\ \frac{\partial f}{\partial x_2} &= \overline{w_2} = 2w_2w_1^2 = 2x_1^2x_2\end{aligned}$$

Note that in one round, the partial derivatives of the function f are known on each dimension.

The interval arithmetic (IA) allows fast computations and does not require much memory. It makes safe and rigorous computing as well as global reasoning possible, even on infinite-sized sets. However, by overestimating the range of an expression, it may be difficult to get accurate and relevant results. Two methods can be used to gain control over these negative aspects: reformulation and piecewise evaluation. Yet, to reformulate any expression into a better one while keeping it mathematically equivalent (to the former one) is a truly complex issue [Ratschek, 1985] (hard problem in terms of complexity). Another method which is easy to implement is the piecewise evaluation. It increases the number of operations during the evaluation (p^n computations with n the number of variables and p the number of pieces for each variables), but those can be controlled by reducing the number of pieces p for each variable. As an example, single occurrence variables should not be concerned with piecewise treatment.

2.3 Multi-intervals

Another limit to the interval arithmetic is the incapacity to represent a value in a disjointed set of range which is inherent to the interval representation. It can be surpassed by using a natural evolution of the interval arithmetic: the *Multi-Interval Arithmetic* (MIA). The latter is very similar with the IA, in terms of objectives and processes. The difference is in the representation of values: an interval is a pair of reals whereas a multi-interval is a set of intervals.

2.3.1 Definitions

The set of multi-intervals noted $\mathbb{MIR}$ is defined as the power set $P(\mathbb{IR})$:

$$\mathbb{MIR} = P(\mathbb{IR}) = \left\{ \{[a_1], \ldots, [a_k]\} \;\middle|\; \begin{array}{c} k \in \mathbb{N},\ k \geq 1 \\ \forall i \in \{1 \ldots k\},\ [a_i] \in \mathbb{IR} \end{array} \right\} \tag{2.39}$$

A lighter notation $[a] = \{[a_1], \ldots, [a_k]\}$ is used. Let $[x] = \{[x_1], \ldots, [x_k]\} \in \mathbb{MIR}$ a non-empty multi-interval, it represents the following set (Fig. 2.4):

$$[x] = \{[x_1], \ldots, [x_k]\} = \{x \in \mathbb{R} \mid \exists [x_i] \in [x],\ x \in x_i\} \tag{2.40}$$

In the next pages, the notation is simplified as below:

- $[x_i] \in [x]$ means that $[x] = \{[x_1], \ldots, [x_i], \ldots, [x_k]\}$. In other words, $[x_i]$ is an interval in the description of $[x]$, which is different that $[x_i] \subseteq [x]$.
- $x \in [x]$ means that x is a real value contained in $[x]$ (the latter can be an interval or a multi-interval)

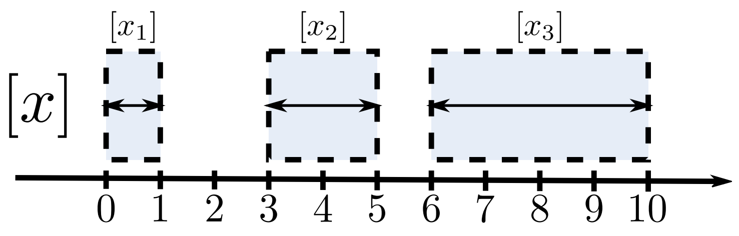


Figure 2.4: Representation of the multi-interval $[x] = \{[0;1], [3;5], [6;10]\}$ Eq. (2.40)

2.3.1.1 Definition: Equivalence class

Two multi-intervals $[x]$ and $[y]$ are equivalent Eq. (2.41) if they represent Eq. (2.40) a similar set of reals.

$$[x] \sim [y] \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \forall x \in [x], \exists y \in [y],\ x = y \\ \forall y \in [y], \exists x \in [x],\ x = y \end{array} \right. \tag{2.41}$$

2.3.1.2 Definition: Canonical form

When there is not a smaller set of intervals $[y] \in \mathbb{MIR}$ leading to the same subset of reals, a multi-interval $[x] = \{[x_1], \ldots, [x_k]\}$ is on its canonical form Eq. (2.42). The latter is unique and is necessarily defined by a set of disjointed intervals.

$$[x] \text{ is canonic} \quad \Leftrightarrow \quad \forall (i,j) \in \{1, \ldots, k\}^2,\ i \neq j \Rightarrow [x_i] \cap [x_j] = \emptyset \tag{2.42}$$

2.3.1.3 Example: A few multi-intervals

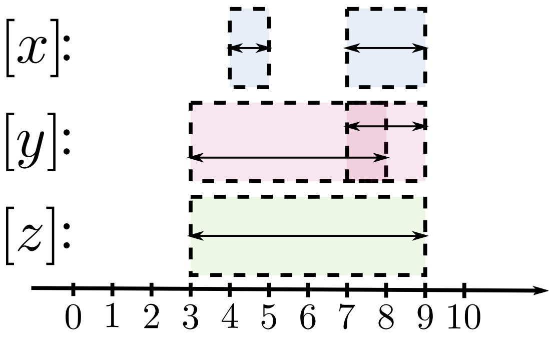
Let $[x]$, $[y]$ and $[z]$ three sets of intervals or multi-intervals (Fig. 2.5a):

$$\begin{aligned} [x] &= \{[4;5], [7;9]\} \\ [y] &= \{[3;8], [7;9]\} \\ [z] &= \{[3;9]\} \end{aligned} \tag{2.43}$$

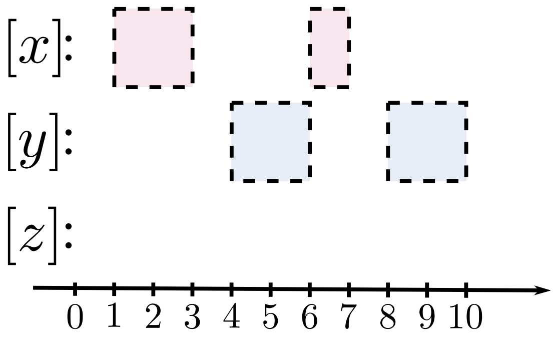
In this example, the multi-interval $[x]$ is a subset of $[y]$ and $[z]$. Note that the multi-intervals $[y]$ and $[z]$ are equivalent because they lead to the similar set of reals.

$$\begin{aligned} [y] &= \{x \in \mathbb{R} \mid x \in [3;8] \ \vee \ x \in [7;9]\} \\ &= \{x \in \mathbb{R} \mid 3 \le x \le 8 \ \vee \ 7 \le x \le 9\} \ \sim \{x \in \mathbb{R} \mid 3 \le x \le 9\} \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad = [z] \end{aligned}$$

Because the multi-interval $[z]$ is the shortest way to describe the set of reals it produces, it is considered as canonic. On the contrary, $[y]$ is not canonic.



(a) Multi-intervals $[x]$, $[y]$ and $[z]$ Eq. (2.43)



(b) Multi-intervals $[x]$, $[y]$ and $[z] = \emptyset$

Figure 2.5: Few examples of multi-intervals

Note that on Figure 2.5b, the multi-interval $[z]$ is empty whereas the multi-interval $[x]$ (resp. $[y]$) is equal to $\{[1;3], [6;7]\}$ (resp. $\{[4;6], [8;10]\}$).

2.3.2 Functions and operations

In order to manipulate multi-intervals, some quantities are defined which are related to the set of multi-intervals. Let $[x] = \{[x_1], \ldots, [x_k]\}$ a multi-interval, we define

- the number of intervals in the set: $size([x]) = k$

- the i-th interval of the set:

$$interval([x], i) = \begin{cases} [x_i] & \textbf{if } 1 \leq i \leq k \\ \emptyset & \textbf{otherwise} \end{cases}$$

- the lower and upper bounds of a multi-interval:

$$lower([x]) = \min_{[x_i]\in[x]} \underline{x_i}$$
$$upper([x]) = \max_{[x_i]\in[x]} \overline{x_i}$$

- the minimal interval bounding a multi-interval:

$$hull([x]) = [lower([x]); upper([x])] \tag{2.44}$$

In order to decrease the difficulty of dealing with intervals and multi-intervals, intervals will be considered as single multi-intervals. Indeed any interval $[x]$ can be converted into a multi-interval $\{[x]\}$. By opposition, multi-interval $[x]$ can be converted into a single interval via the hull function $hull([x])$ when necessary. Then we note $\underline{x} = lower([x])$ and $\overline{x} = upper([x])$ the lower and upper bounds of a value bounded by a multi-interval $[x]$.

Also, all the functions available on intervals can be applied on multi-intervals once the latter is converted into the former using the *hull* as mentioned before Eq. (2.44). Just a few functions can be specified, such as the *width* or the *interior*:

$$width([x]) := \sum_{[x_i]\in[x]} width([x_i]) \tag{2.45}$$

$$interior([x]) := \cup_{[x_i]\in[x]} interior([x_i]) \tag{2.46}$$

About the binary operators such that $\{+, -, \times, \div\}$, an automatic computation scheme can be applied, although it has for a consequence to increase the number of operations required to perform the computation. Let $\diamond^{\mathbb{IR}}$ a binary operator valid on intervals, $[x]$ and $[y]$ two multi-intervals. The operator $\diamond^{\mathbb{IR}}$ can be extended to the multi-intervals $\diamond^{\mathbb{MIR}}$ as follows:

$$\diamond^{\mathbb{MIR}}([x], [y]) = \bigcup_{\substack{[x_i]\in[x] \\ [y_j]\in[y]}} \diamond_{\mathbb{IR}}([x_i], [y_j]) \tag{2.47}$$

where the union $\cup$ is the classic union operation of sets.

Note that the extension of these operators on multi-intervals requires much more operations and memory depending on the number of intervals within multi-intervals. At the end of several operations, the result might be made of a large number of intervals.

2.3.2.1 Reduction process towards the canonical form

A first answer to these issues is to always keep and deal with canonic form intervals and do as such for any computations. Let $[x] = \{[x_1], \ldots, [x_k]\} \in \mathbb{MIR}$ a non-empty multi-interval. To reduce $[x]$ towards its canonical form, the following method can be used:

1. sorting the set $\{[x_1], \dots, [x_k]\}$ using the order defined below:

$$\underline{x_i} < \underline{x_j} \;\vee\; (\underline{x_i} \leq \underline{x_j} \wedge \overline{x_i} < \overline{x_j}) \quad \Rightarrow \quad [x_i] \prec [x_j] \tag{2.48}$$

2. grouping the adjacent intervals $[x_i]$, $[x_{i+1}]$ iff

$$[x_i] \cap [x_{i+1}] \neq \emptyset \quad \Rightarrow \quad [x] \leftarrow ([x] \setminus \{[x_i], [x_{i+1}]\}) \cup ([x_i] \cup [x_{i+1}]) \tag{2.49}$$

   where the first $\cup$ is the set operator while the second one $([x_i] \cup [x_{i+1}])$ is the interval operator.

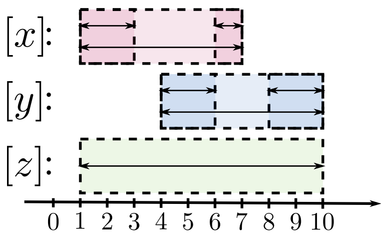
2.3.2.2 Example: Computation of the union with disjointed intervals

Let $[x] \in \mathbb{MIR}$ and $[y] \in \mathbb{MIR}$ two multi-intervals such that $[x] \cap [y] = \emptyset$. Let $[x] = \{[1;3],[6;7]\}$ and $[y] = \{[4;6],[8;10]\}$ two multi-intervals (Fig. 2.5b). The result $[z]$ of the union of $[x]$ and $[y]$ can be computed with the interval arithmetic (Fig. 2.6a):

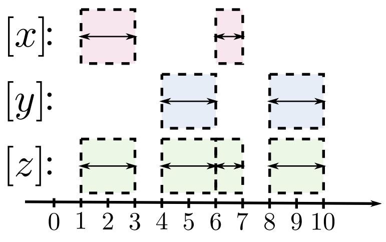
$$hull([x]) \cup hull([y]) = [1;7] \cup [4;10] = [1;10] \tag{2.50}$$

with the multi-interval arithmetic (Fig. 2.6b) and the canonic reduction (Fig. 2.6c):

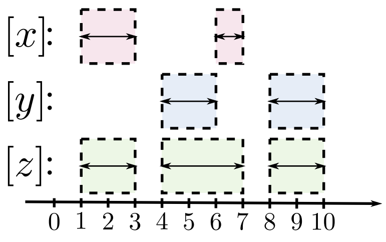
$$[x] \cup [y] = \{[1;3],[4;6],[6;7],[8;10]\} \sim \{[1;3],[4;7],[8;10]\} \tag{2.51}$$



(a) Computation of hulls (interval)



(b) Union of sets (multi-interval)



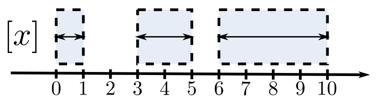
(c) Canonic union of sets (multi-interval)

Figure 2.6: Three versions of the computation $[z] \leftarrow [x] \cup [y]$

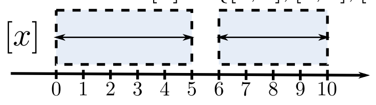
2.3.2.3 Approximate the best enclosure

A second solution could be to limit the size of multi-intervals to a value k, and then to merge the nearest pairs of intervals as long as the size is illegal. Let $[x]$ a multi-interval of size $k+1$, several multi-intervals $[y]$ exist such that $[x] \subset [y]$ and $size([y]) = size([k]) - 1$ (Fig. 2.7).

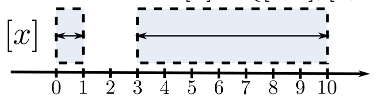
Then a measure of the distance between two intervals $[x_i]$ and $[x_j]$ has to be determined. Naively, the distance from the first midpoint to the second could be considered, but it would not minimize the over-approximation.



(a) Multi-interval $[x] = \{[0;1],[3;5],[6;10]\}$



(b) Multi-interval $[x] = \{[0;5],[6;10]\}$



(c) Multi-interval $[x] = \{[0;1],[3;10]\}$

Figure 2.7: Limitation of the size of a multi-interval (Fig. 2.4)

Let $[x] = \{[x_1],\dots,[x_k]\}$ a canonic multi-interval of size k. What is the nearest multi-interval $[y]$ of size $k-1$? It is the tiniest multi-interval (in terms of *width*). In other words, $[y]^*$ is such that

$$[x] \subset [y]^* \tag{2.52}$$
$$\wedge \quad \forall [y] \in \mathbb{MIR},\ [x] \subset [y] \wedge size([y]) = k-1 \Rightarrow width([y]^*) \leq width([y]) \tag{2.53}$$

Because $[x]$ is on its canonical form then $width([x]) \leq width([y]^*)$. As a result of $size([y]^*) = size([x]) - 1$, two intervals $[x_i]$ and $[x_j]$ in $[x]$ have to merge into one another then must be deleted. Then

$$width([y]^*) = width([x]) - width([x_i]) - width([x_j]) + width([x_i] \cup [x_j])) \tag{2.54}$$

Consequently, the tiniest multi-interval $[y]^*$ of size $k-1$ that contains $[x]$ is the one that merges the two nearest elements of $[x]$ depending on the following metric

$$distance([x],[y]) = width(hull([x] \cup [y])) - width([x]) - width([y]) \tag{2.55}$$

2.3.2.4 Example:

From the initial example (Fig. 2.4) in which $[x] = \{[0;1];[3;5];[6;10]\}$,

$$[x_1] = interval([x],1) = [0;1]$$
$$[x_2] = interval([x],2) = [3;5]$$
$$[x_3] = interval([x],3) = [6;10]$$

the computation of the distances provides the following results

$$\begin{aligned}
distance([x_1],[x_2]) &= width([0;5]) - width([0;1]) - width([3;5]) = 5-1-2 = 2\\
distance([x_1],[x_3]) &= width([0;10]) - width([0;1]) - width([6;10]) = 10-1-4 = 5\\
distance([x_2],[x_3]) &= width([3;10]) - width([3;5]) - width([6;10]) = 7-2-4 = 1
\end{aligned}$$

Because the distance between $[x_2]$ and $[x_3]$ is the tiniest, they are the best candidates in order to be merged together and decrease the size of the multi-interval. Note that multi-intervals are defined with successive intervals, thus the computation of the distance from $[x_1]$ to $[x_3]$ does not make any difference and only $k-1$ computations are required.

In this section the multi-interval arithmetic (MIA) has been introduced in order to improve the ability to represent non-continuous sets of values. Just like the interval arithmetic, it allows fast computations and does not require much memory while the size of the multi-intervals is controlled and remains acceptable (in terms of number of intervals contained in their definition). The multi-interval arithmetic has the same advantages and limits as the interval arithmetic such as rigorous computing, global reasoning and overestimation of the range of expressions. As a consequence similar methods can be used to get more accurate and relevant results: reformulation of the inclusion functions and piecewise evaluation.

2.4 Affine Arithmetic

The affine arithmetic [De Figueiredo and Stolfi, 2004] is a computational model which provides guaranteed enclosures. The possibilities given by the affine arithmetic are similar to the ones provided by the interval arithmetic. The particularity of the former resides in the use of the linear combinations to represent values including error terms. Let x a value, its affine representation is described as below

$$x = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n$$

where x_i are reals and ε_i are symbolic variables representing errors through intervals equal to $[-1; 1]$. The strength of this arithmetic is the accuracy of the results it provides during computations, as opposed to the interval arithmetic and its dependency problems. The affine representation has some interesting properties, given that quantities are represented by zonotopes (which are invariant by rotations). But the representation with first order approximation increases the operations cost (important memory and computing time requirements).

The affine arithmetic is efficient in global optimization algorithms [Messine, 2002], but it is more difficult to implement and the cost in terms of memory and time computing are the reason the use of interval arithmetic is preferred in this work. Therefore this arithmetic is not defined here but it provides several advantages, thus it should be consider to extend the methods developed in the tool GDODynS that has been implemented through this work.

2.5 Tube Arithmetic

The tube arithmetic is a natural evolution of the interval arithmetic when it comes to represent sets of functions [Kurzhanskiĭ and Vályi, 1997], [Le Bars et al., 2012]. The advantages of this arithmetic are quite similar to the ones of the interval arithmetic: supporting safe computations and reasoning on sets of *a priori* unknown functions.

The interval arithmetic is useful to compute safely on real values, but the model that is considered in this thesis also contains functional variables, meaning variables which represent functions instead of reals.

Dealing with functional values improves the expressive power of the model because a single functional value can replace an infinite number of real values.

2.5.1 Definition

The set of tubes $\mathbb{IF}$ on the real functions from $\mathbb{R}$ to $\mathbb{R}$ is defined as below Eq. (2.56).

$$\mathbb{IF} = \left\{ [\underline{u};\overline{u}] \;\middle|\; \begin{array}{l} \underline{u} \in (\mathbb{D}_{\underline{u}} \subseteq \mathbb{R}) \to \mathbb{R} \\ \overline{u} \in (\mathbb{D}_{\overline{u}} \subseteq \mathbb{R}) \to \mathbb{R} \end{array} \right\} \tag{2.56}$$

where $\mathbb{D}_{\underline{u}}$ and $\mathbb{D}_{\overline{u}}$ are the definition domains of the functions $\underline{u}$ and $\overline{u}$. A tube $[u] = [\underline{u};\overline{u}]$ represents a set of unary functions u from $\mathbb{D}_u \subseteq \mathbb{D}_{\underline{u}} \cap \mathbb{D}_{\overline{u}}$ to $\mathbb{R}$, such that the latter is bounded by $\underline{u}$ and $\overline{u}$ Eq. (2.57). The intersection of the domains $\mathbb{D}_{\underline{u}} \cap \mathbb{D}_{\overline{u}}$ is noted $\mathbb{D}_{[u]}$.

$$[u] = [\underline{u};\overline{u}] = \left\{ u \in \mathbb{D}_u \to \mathbb{R} \;\middle|\; \forall x \in \mathbb{D}_u \subseteq \mathbb{D}_{[u]},\ \underline{u}(x) \leq u(x) \leq \overline{u}(x) \right\} \tag{2.57}$$

Figure 2.8 represents a tube $[u]$ defined from the set of values x in $\mathbb{D}_{[u]} = \{[a;f],[g;j]\}$ to a subset of $\mathbb{R}$ which is bounded by the two functions $\underline{u}$ and $\overline{u}$.

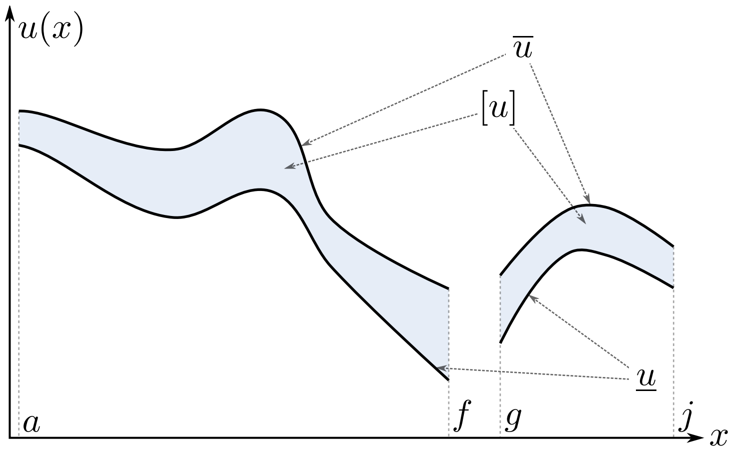


Figure 2.8: Tube $[\mathbf{u}]$ enclosing $\mathbf{u}$

2.5.1.1 Definition: Function enclosed in a tube

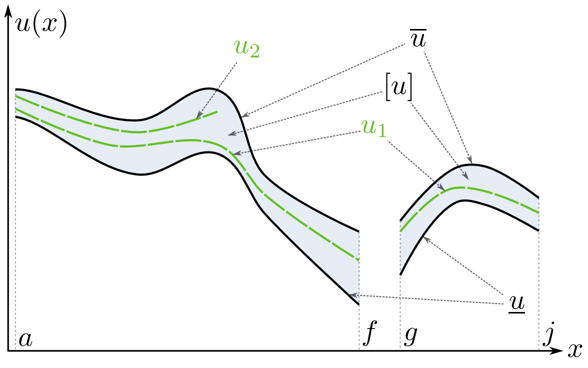
Let $[u] \in \mathbb{IF}$ a tube, and u a real function from $\mathbb{D}_u \subseteq \mathbb{R}$ to $\mathbb{R}$. The real function u is in the tube $[u]$ iff

$$\mathbb{D}_u \subseteq \mathbb{D}_{[u]} \tag{2.58}$$
$$\wedge \quad \forall x \in \mathbb{D}_u,\ \underline{u}(x) \leq u(x) \leq \overline{u}(x) \tag{2.59}$$

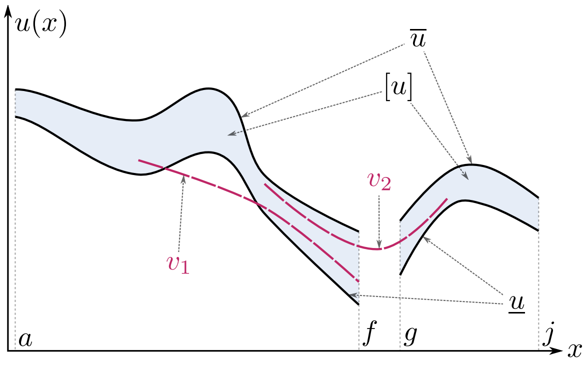
2.5.1.2 Example: Inclusion of functions in a tube (Fig. 2.9)

On Figure 2.9, a tube $[u] = [\underline{u}; \overline{u}]$ and four functions u_1, u_2, v_1 and v_2 are defined. The first pair of functions u_1 and u_2 is enclosed in the tube $[u]$ (Fig. 2.9a) whereas the second pair (v_1 and v_2) is not (Fig. 2.9b).

- $v_1 \notin [u]$: v_1 is not enclosed in the tube $[u]$ because there is a set of real values x such that $v_1(x)$ is lower than the lower bound $\underline{u}(x)$.
- $v_2 \notin [u]$: v_2 is not enclosed in the tube $[u]$ because there is a set of real values x such that $v_2(x)$ is defined while $\underline{u}$ and $\overline{u}$ are not ($x \notin \mathbb{D}_{[u]} = \mathbb{D}_{\underline{u}} \cap \mathbb{D}_{\overline{u}}$).



(a) u_1 and u_2 enclosed by the tube $[\mathbf{u}]$



(b) v_1 and v_2 **not** enclosed by the tube $[\mathbf{u}]$

Figure 2.9: Functions inside and outside a tube

The set operations can be applied on tubes which represent sets of unary functions Eq. (2.57).

2.5.1.3 Definition: Inclusion of tubes

Let two tubes $[u] \in \mathbb{IF}$ and $[v] \in \mathbb{IF}$. $[v]$ is a subset of $[u]$ iff

$$\forall f \in (\mathbb{R} \to \mathbb{R}), f \in [v] \Rightarrow f \in [u] \tag{2.60}$$

2.5.1.4 Remark: On Figure 2.9, both pairs of functions u_1, u_2 and v_1, v_2 define a tube. There is an interesting side to the tube $[u_1; u_2]$, the boundary function u_1 is apart from the set it bounds. Similarly, v_1 and v_2 are not elements of the tube $[v_1; v_2]$. The two tubes $[u_1; u_2]$ and $[v_1; v_2]$ are included in the tube $[u]$.

2.5.1.5 Remark: Regarding at the definition of a tube, the inclusion property can be described as below

$$[v] \subseteq [u] \quad \Leftrightarrow \quad \begin{cases} & \mathbb{D}_{[v]} \subseteq \mathbb{D}_{[u]} \\ \wedge & \forall x \in \mathbb{D}_{[v]}, \begin{cases} & \underline{v}(x) > \overline{v}(x) \\ \vee & \underline{u}(x) \leq \underline{v}(x) \leq \overline{v}(x) \leq \overline{u}(x) \end{cases} \end{cases} \tag{2.61}$$

2.5.1.6 Definition: Intersection and Union of tubes
The intersection is defined by a restriction over the domains where the functions are defined as well as on the set of accessible values. Let $[u]$ and $[v]$ two tubes, then

$$[u] \cap [v] = \{f \in \mathbb{R} \to \mathbb{R} \mid f \in [u] \;\wedge\; f \in [v]\} \tag{2.62}$$

The union is defined by an extension of the domains and the reachable values which are defined by at least one of the tubes.

$$[u] \cup [v] = \{f \in \mathbb{R} \to \mathbb{R} \mid f \in [u] \;\vee\; f \in [v]\} \tag{2.63}$$

2.5.2 Equivalence classes and canonical form

The definition of the set of tubes $\mathbb{IF}$ induces a large number of equivalent tubes Eq. (2.64). A set of functions can be generated by different tubes, all of them being equivalent and members of the same equivalence class. For each equivalence class, a tube exists which is revealed on a canonic form.

2.5.2.1 Definition: Equivalence class
Two tubes $[u] \in \mathbb{IF}$ and $[v] \in \mathbb{IF}$ are equivalent if they represent a similar set of functions.

$$[u] \sim [v] \quad \Leftrightarrow \quad [u] \subseteq [v] \wedge [v] \subseteq [u] \tag{2.64}$$

Then many tubes $[u]$ can be gathered within an equivalence class Eq. (2.65) noted $\mathbb{IF}_{[u]}$. Therefore all the tubes from the same class are equivalent.

$$\mathbb{IF}_{[u]} = \{[v] \in \mathbb{IF} \mid [v] \sim [u]\} \tag{2.65}$$

In the next paragraphs, the reduction from each tube towards its canonic equivalent is presented through two complementary methods.

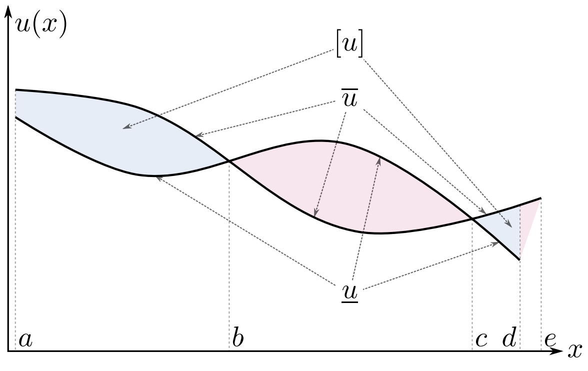
2.5.2.2 First reduction
The first reduction consists in limiting the domains $\mathbb{D}_{\underline{u}}$ and $\mathbb{D}_{\overline{u}}$ of the boundary function $\underline{u}$ and $\overline{u}$.

Let $[u] = [\underline{u}; \overline{u}]$ a tube defined over $\mathbb{D}_{[u]} = \mathbb{D}_{\underline{u}} \cap \mathbb{D}_{\overline{u}}$, where $\underline{u}$ in $\mathbb{D}_{\underline{u}} \to \mathbb{R}$ and $\overline{u}$ in $\mathbb{D}_{\overline{u}} \to \mathbb{R}$. Let $\overline{\mathbb{D}_{[u]}}$ a non-empty set of reals, such that $\overline{\mathbb{D}_{[u]}} = (\mathbb{D}_{\underline{u}} \cup \mathbb{D}_{\overline{u}}) \setminus \mathbb{D}_{[u]}$. The set $\overline{\mathbb{D}_{[u]}}$ is equal to $[d; e]$ on Figure 2.10a. Then the tube $[u_c] = [\underline{u_c}; \overline{u_c}]$ where

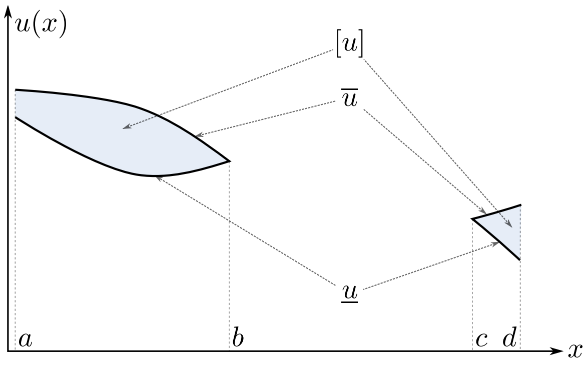
$$\underline{u_c} : \begin{array}{r} (\mathbb{D}_{\underline{u}} \setminus \overline{\mathbb{D}_{[u]}}) \to \mathbb{R} \\ x \mapsto \underline{u}(x) \end{array} \quad \textbf{and} \quad \overline{u_c} : \begin{array}{r} (\mathbb{D}_{\overline{u}} \setminus \overline{\mathbb{D}_{[u]}}) \to \mathbb{R} \\ x \mapsto \overline{u}(x) \end{array} \tag{2.66}$$

is equivalent to $[u]$.

$$[u] = [\underline{u}; \overline{u}] = \left\{ u \in \mathbb{D}_u \to \mathbb{R} \;\middle|\; \begin{array}{l} \mathbb{D}_u \subseteq \mathbb{D}_{\underline{u}} \cap \mathbb{D}_{\overline{u}} \\ \forall x \in \mathbb{D}_u,\ \underline{u}(x) \leq u(x) \leq \overline{u}(x) \end{array} \right\}$$
$$\Leftrightarrow \left\{ u \in \mathbb{D}_u \to \mathbb{R} \;\middle|\; \begin{array}{l} \mathbb{D}_u \subseteq (\mathbb{D}_{\underline{u}} \setminus \overline{\mathbb{D}_{[u]}}) \cap (\mathbb{D}_{\overline{u}} \setminus \overline{\mathbb{D}_{[u]}}) \\ \forall x \in \mathbb{D}_u,\ \underline{u_c}(x) \leq u(x) \leq \overline{u_c}(x) \end{array} \right\} = [\underline{u_c}; \overline{u_c}] = [u_c]$$



(a) Tube $[\mathbf{u}]$: Non canonic



(b) Tube $[\mathbf{u}]$: Canonic

Figure 2.10: Two equivalent tubes $[u]$

2.5.2.3 Second reduction

Let $[u] \in \mathbb{IF}$ a tube and $\mathbb{X}_{[u]} \subseteq \mathbb{D}_{[u]}$ a set or reals such that

$$\mathbb{X}_{[u]} = \{x \in \mathbb{D}_{[u]} \mid \underline{u}(x) > \overline{u}(x)\} \tag{2.67}$$

The set $\mathbb{X}_{[u]}$ is equal to $[b; c]$ on Figure 2.10a. Thus the set $\mathbb{X}_{[u]}$ can be removed from the domain $\mathbb{D}_{[u]}$ given that a function $u \in [u]$ cannot be defined over it. Let $\underline{v}$ and $\overline{v}$ two functions such that:

$$\underline{v} : \begin{array}{l} \mathbb{D}_{[v]} \to \mathbb{R} \\ x \mapsto \underline{u}(x) \end{array} \quad \textbf{and} \quad \overline{v} : \begin{array}{l} \mathbb{D}_{[v]} \to \mathbb{R} \\ x \mapsto \overline{u}(x) \end{array} \tag{2.68}$$

where $\mathbb{D}_{[v]} = (\mathbb{D}_{\underline{u}} \cap \mathbb{D}_{\overline{u}}) \setminus \mathbb{X}$. Then the tube $[v] = [\underline{v}; \overline{v}]$ is equivalent to $[u]$.

$$[v] = [\underline{v}; \overline{v}] = \{v \in \mathbb{D}_v \to \mathbb{R}\}\,. \tag{2.69}$$

2.5.2.4 Definition: Canonical form of a tube

A tube $[u] = [\underline{u}; \overline{u}]$ is on a canonical form iff there are no functions $\underline{v}$ from $\mathbb{D}_{\underline{v}} \subset \mathbb{D}_{\underline{u}}$ to $\mathbb{R}$ or $\overline{v}$ from $\mathbb{D}_{\overline{v}} \subset \mathbb{D}_{\overline{u}}$ to $\mathbb{R}$ such that $[u] \sim [\underline{u}; \overline{v}]$ or $[u] \sim [\underline{v}; \overline{u}]$. In other words:

$$[u] \text{ is canonic} \quad \Leftrightarrow \quad \begin{array}{rl} & \mathbb{D}_{\underline{u}} = \mathbb{D}_{\overline{u}} = \mathbb{D}_{[u]} \\ \wedge & \forall x \in \mathbb{D}_{[u]},\ \underline{u}(x) \leq \overline{u}(x) \end{array} \tag{2.70}$$

2.5.3 Numerical representation

Previously, a tube $[u]$ was defined as an enclosure made of two functions $\underline{u}$ and $\overline{u}$, in order to enclose sets of functions and compute operations on them. But the storage of such functions (using expression trees) requires memory while the computations need time. Here two different ways to represent the boundary functions $\underline{u}$ and $\overline{u}$ are introduced.

- On one hand, piecewise linear functions to model the lower and upper bounds $\underline{u}$ and $\overline{u}$ are used.
- On the other hand, piecewise **constant** functions will be used to replace the piecewise **linear** functions introduced in the first point. This will lead to the use of piecewise **interval** functions in order to model tubes.

2.5.3.1 Definition: Piecewise linear function

A piecewise linear function f is a function from $\mathbb{D}_f \subseteq \mathbb{R}$ to $\mathbb{R}$, defined on k segments $\mathbb{D}_{f_i}$ Eq. (2.71) such that:

$$f : \begin{array}{ccl} \mathbb{D}_f & \to & \mathbb{R} \\ x & \mapsto & \begin{cases} f_1(x) & \textbf{if} \quad x \in \mathbb{D}_{f_1} \\ \vdots & \\ f_k(x) & \textbf{if} \quad x \in \mathbb{D}_{f_k} \end{cases} \end{array} \qquad \textbf{where} \qquad \bigcup_{1\leq i\leq k} \mathbb{D}_{f_i} = \mathbb{D}_f \tag{2.71}$$

in which each piece f_i of f is an affine function :

$$f_i : \begin{array}{ccc} \mathbb{D}_{f_i} & \to & \mathbb{R} \\ x & \mapsto & \alpha_i x + \beta_i \end{array}$$

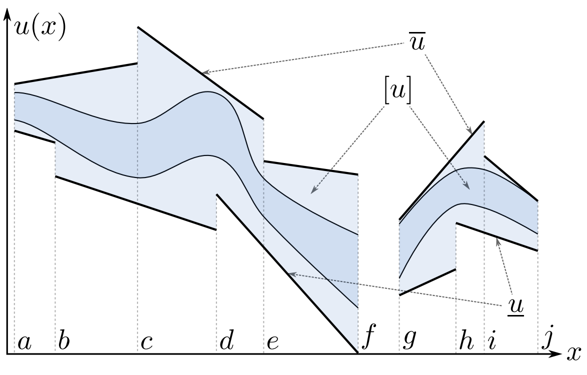
2.5.3.2 Example: Affine representation of a tube (Fig. 2.11a)

On Figure 2.11a, the tube $[u]$ is bounded by two piecewise linear functions $\underline{u}$ and $\overline{u}$. They are defined from $\{[a; f], [g; j]\}$ to $\mathbb{R}$ as below:

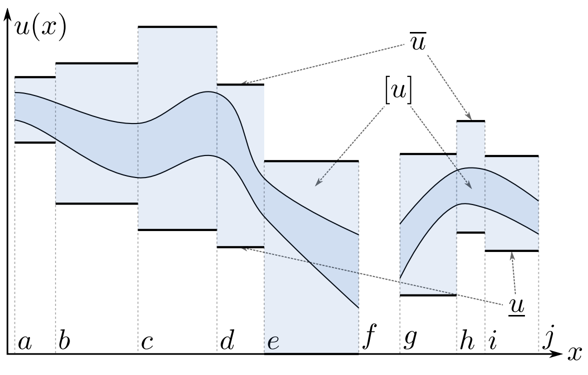
$$\underline{u} : x \mapsto \begin{cases} \underline{u_1}(x) & \textbf{if } x \in \mathbb{D}_{\underline{u_1}} = [a; b] \\ \underline{u_2}(x) & \textbf{if } x \in \mathbb{D}_{\underline{u_2}} = [b; d] \\ \underline{u_3}(x) & \textbf{if } x \in \mathbb{D}_{\underline{u_3}} = [d; f] \\ \underline{u_4}(x) & \textbf{if } x \in \mathbb{D}_{\underline{u_4}} = [g; h] \\ \underline{u_5}(x) & \textbf{if } x \in \mathbb{D}_{\underline{u_5}} = [h; j] \end{cases} \quad \textbf{and} \quad \overline{u} : x \mapsto \begin{cases} \overline{u_1}(x) & \textbf{if } x \in \mathbb{D}_{\overline{u_1}} = [a; c] \\ \overline{u_2}(x) & \textbf{if } x \in \mathbb{D}_{\overline{u_2}} = [c; e] \\ \overline{u_3}(x) & \textbf{if } x \in \mathbb{D}_{\overline{u_3}} = [e; f] \\ \overline{u_4}(x) & \textbf{if } x \in \mathbb{D}_{\overline{u_4}} = [g; i] \\ \overline{u_5}(x) & \textbf{if } x \in \mathbb{D}_{\overline{u_5}} = [i; j] \end{cases}$$

Note that the definition domains of the pieces of $\underline{u}$ and $\overline{u}$ are not synchronized.

Tubes can be represented with piecewise linear functions and the bound functions definition can be simplified to its extreme limit through the use of piecewise constant functions. This is a peculiar case of the affine function, whose the terms α_i are forced to be equal to 0. At last by synchronizing the boundary functions $\underline{u}$ and $\overline{u}$, the tube $[u]$ can be defined by a single piecewise interval function (Fig. 2.11b).



(a) Tube $[\mathbf{u}]$: Affine representation



(b) Tube $[\mathbf{u}]$: Interval representation

Figure 2.11: Numerical representations of a tube

2.5.3.3 Definition: Piecewise interval function

A piecewise interval function is a function defined on segments Eq. (2.71) such that each of them is mapped to an interval. Let $[u]$ a piecewise interval function made of k pieces, then $[u]$ is defined as below:

$$[u]: \begin{array}{ccl} \mathbb{D}_{[u]} & \rightarrow & \mathbb{R} \\ x & \mapsto & \begin{cases} [u_1] & \textbf{if} \quad x \in \mathbb{D}_{[u_1]} \\ \vdots & \\ [u_k] & \textbf{if} \quad x \in \mathbb{D}_{[u_k]} \end{cases} \end{array} \qquad \textbf{where} \quad \bigcup_{1 \leq i \leq k} \mathbb{D}_{[u_i]} = \mathbb{D}_{[u]} \tag{2.72}$$

2.5.3.4 Example: Interval representation of a tube (Fig. 2.11b)

On Figure 2.11*b*, the tube $[u]$ is defined by a piecewise interval function made of eight pieces:

$$[u] : x \mapsto \begin{cases} [u_1] & \textbf{if } x \in \mathbb{D}_{[u_1]} = [a; b] \\ [u_2] & \textbf{if } x \in \mathbb{D}_{[u_2]} = [b; c] \\ [u_3] & \textbf{if } x \in \mathbb{D}_{[u_3]} = [c; d] \\ [u_4] & \textbf{if } x \in \mathbb{D}_{[u_4]} = [d; e] \\ & \dots \end{cases} \quad \begin{cases} & \dots \\ [u_5] & \textbf{if } x \in \mathbb{D}_{[u_5]} = [e; f] \\ [u_6] & \textbf{if } x \in \mathbb{D}_{[u_5]} = [g; h] \\ [u_7] & \textbf{if } x \in \mathbb{D}_{[u_5]} = [h; i] \\ [u_8] & \textbf{if } x \in \mathbb{D}_{[u_5]} = [i; j] \end{cases} \tag{2.73}$$

A simplified notation of the tubes details will be used in the rest of this thesis.

$$[u] = \begin{bmatrix} [x_1] \mapsto [u_1] \\ \vdots \\ [x_k] \mapsto [u_k] \end{bmatrix} \tag{2.74}$$

2.5.3.5 Definition: Canonical form of the piecewise interval function

Using piecewise numerical representation introduces new conditions for tubes to be canonic. Since all the pieces of the function are defined on closed domains, they can be locally covered on the borders of their domain. Let $[u]$ a piecewise interval function Eq. (2.72), whose $[x_i] \mapsto [u_i]$ and $[x_j] \mapsto [u_j]$ are two different pieces. The tube $[u]$ is canonic iff the previous Eq. (2.70) and new Eq. (2.75), (2.76) and (2.77) conditions are respected. The first condition Eq. (2.75) guarantees that the pieces are not more covered than on the border.

$$interior([x_i]) \cap interior([x_j]) = \emptyset \tag{2.75}$$

The aim of the second condition Eq. (2.76) is to protect the tube from having two similar adjacent pieces (they should merged into a single piece).

$$[x_i] \cap [x_j] \neq \emptyset \quad \Rightarrow \quad [u_i] \cap [u_j] \subset [u_i] \cup [u_j] \tag{2.76}$$

The third condition Eq. (2.77) is applied on degenerate domains $[x_i]$ where $\underline{x_i} = \overline{x_i}$ because of the first condition Eq. (2.75). It allows the degenerate intervals to be covered by another one, iff they are decreasing the local width of the tube.

$$[x_i] \subseteq [x_j] \quad \Rightarrow \quad [u_i] \subset [u_j] \tag{2.77}$$

2.5.3.6 Definition: Union and intersection of piecewise interval functions

Using the interval representation $[u] = \{[x_i] \mapsto [u_i]\}_{1 \leq i \leq m}$ and $[v] = \{[y_i] \mapsto [v_i]\}_{1 \leq i \leq n}$, the intersection $[u] \cap [v]$ as well as the union $[u] \cup [v]$ produces a tube $[w] = [u] \cap [v] = \{[z_i] \mapsto [w_i]\}_{1 \leq i \leq o}$.

- The intersection produces a tube $[w]$ such that for all $([x_i] \mapsto [u_i])$ in $[u]$ and $([y_j] \mapsto [v_j])$ in $[v]$:

$$[x_i] \cap [y_j] \neq \emptyset \;\Rightarrow\; \exists [z_k] \mapsto [w_k] \in [w], \; \begin{cases} & [z_k] = [x_i] \cap [y_j] \\ \wedge & [w_k] = [u_i] \cap [v_j] \end{cases} \tag{2.78}$$

- The union produces a tube $[w]$ such that $[u] \subseteq [w]$ and $[v] \subseteq [w]$.

2.5.3.7 Definition: Subset of piecewise interval function

Let $[u] = \{[x_i] \mapsto [u_i]\}_{1 \leq i \leq m}$ and $[v] = \{[y_i] \mapsto [v_i]\}_{1 \leq i \leq n}$ two piecewise interval functions. The tube $[v]$ is a subset of the tube $[u]$ if and only if:

- the domain $\mathbb{D}_{[v]}$ on which the function $[v]$ is defined, is a subset of the domain $\mathbb{D}_{[u]}$ on which the tube $[u]$ is defined:

$$domain([v]) \subseteq domain([u]) \tag{2.79}$$

- the set of reachable values of the function $[v]$ is a subset of the values reached by the function $[u]$:

$$\forall([x_i] \mapsto [u_i]) \in [u], \; \forall([y_j] \mapsto [v_j]) \in [v], \; [x_i] \cap [y_j] \neq \emptyset \Rightarrow [v_j] \subseteq [u_i] \tag{2.80}$$

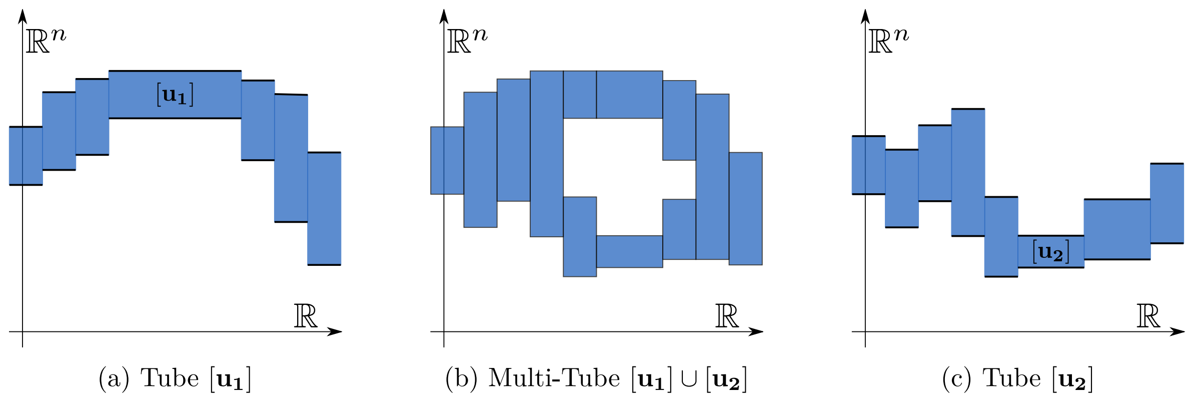


Figure 2.12: Operation on tubes $[\mathbf{u_1}]$ and $[\mathbf{u_2}]$

2.5.3.8 Remark: Limits of interval representation

Unfortunately, as shown on Figure 2.13, the definition of a tube $[u]$ with a piecewise interval function might suffer from an over-approximation problem. Compared to the minimal enclosure of the functions u (u_1, u_2, $\underline{u}$ and $\overline{u}$ are represented) in which the functions v_1 and v_2 are not enclosed (see Fig. 2.9b), the interval representation does enclose v_1.

This last point is a price that has to be paid in order to keep computations fast and efficient. Moreover, this drawback can be reduced by increasing the number of pieces to define the tube $[u]$.

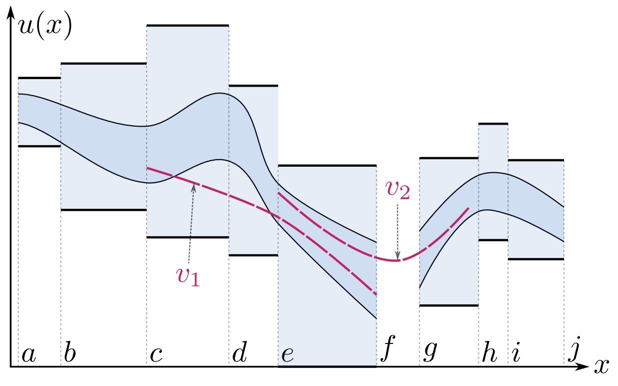


Figure 2.13: Tube $[\mathbf{u}]$ enclosing $\mathbf{u}$: Numerical representation

2.5.3.9 Remark: Variations around piecewise interval functions

The set of piecewise interval functions $[u] = \{([x_i] \mapsto [u_i])\}_{1 \leq i \leq k}$ in $\mathbb{IF}$ can be upgraded by replacing the set of images. Instead of the interval $[u_i]$ in $\mathbb{IR}$, we can use

- boxes ($[\mathbf{u_i}]$ in $\mathbb{IR}^n$) to describe n-dimensional tubes $\mathbb{IF}^n$
- multi-intervals ($[u_i]$ in $multi - \mathbb{IR}$) to describe multi-tubes $multi - \mathbb{IF}$

- multi-interval boxes ($[u_i]$ in $multi - \mathbb{IR}^n$) to describe n-dimensional multi-tubes $multi - \mathbb{IF}^n$.

In this thesis, the interval representation of tubes is used because it is easy to implement (a layer above the interval arithmetic) and is efficient in numerous ways : First, this representation does not require much memory and the operations on these structures are fast. Moreover, it takes the advantages of the interval arithmetic use and, ultimately, it can easily adapt itself to take into account interval vectors, multi-intervals, and multi-interval vectors thus defining n-dimensional multi-tubes.

However, the internal way to represent tubes (arrays, lists, trees, etc) does not impact any of the theoretical aspects presented here; only the experimental results might differ (in terms of memory or computation time) because of the computational complexity to perform basics operations (to get the k-th segment of the tube, to add a new segment, etc).

2.5.4 Metrics, evaluation and operations on tubes

The tubes $[u]$ in $\mathbb{IF}$, are arithmetic objects, therefore a set of quantities are available.

2.5.4.1 Definition: A few metrics on tubes

Let $[u] \in \mathbb{IF}$ a tube defined by k pieces $[x_i] \mapsto [u_i]$. The *length* Eq. (2.83) of a tube is the width of the domain $\mathbb{D}_{[u]} = domain([u])$ Eq. (2.81) while the *width* Eq. (2.84) corresponds to maximal *width* of the domain reached by the application of $[u]$ over its domain $\mathbb{D}_{[u]}$. The *volume* Eq. (2.85) is equal to the surface or the volume (in multi-dimensional cases), as described by the tube. The *interior* Eq. (2.86) is the set of all the functions u strictly enclosed in the tube.

$$domain([u]) = \bigcup_{1 \leq i \leq k} [x_i] \tag{2.81}$$

$$image([u]) = \bigcup_{1 \leq i \leq k} [u_i] \tag{2.82}$$

$$length([u]) = width(domain([u])) \tag{2.83}$$

$$width([u]) = width(image([u])) \tag{2.84}$$

$$volume([u]) = \sum_{i \in \{1,\dots,k\}} width([x_i]) \times volume([u_i]) \tag{2.85}$$

$$interior([u]) = \left\{ u \in [u] \;\middle|\; \forall x \in \mathbb{D}_{[u]},\ \underline{u}(x) < u(x) < \overline{u}(x) \right\} \tag{2.86}$$

2.5.4.2 Definition: Local and global evaluation

Tubes $[u]$ are groups of unknown functions, whose bounds $\underline{u}$ and $\overline{u}$ are known. Those bounds can be used to evaluate these sets of functions over both a local value $x \in \mathbb{R}$ Eq. (2.87) and a global value $[x] \in \mathbb{IR}$ Eq. (2.89) which can be either an interval or a multi-interval.

$$\begin{aligned}[u](x) &= \{u(x) \mid u \in [u]\} && (2.87)\\ &= \bigcap_{([x_i]\mapsto[u_i])\in[u]} \begin{cases} [u_i] & \textbf{if } x \in [x_i] \\ [-\infty;+\infty] & \textbf{otherwise} \end{cases} && (2.88)\end{aligned}$$

$$\begin{aligned}[u]([x]) &= \{u(x) \mid u \in [u],\ x \in [x]\} && (2.89)\\ &= \bigcup_{([x_i]\mapsto[u_i])\in[u]} \begin{cases} [u_i] & \textbf{if } [x]\cap[x_i] \neq \emptyset \\ \emptyset & \textbf{otherwise} \end{cases} && (2.90)\end{aligned}$$

2.5.4.3 Remark:

Note that to perform the evaluation on a single value x the intersection operator is used to accumulate the result, whereas the accumulation is done using the union operator when evaluating the tube over interval values $[x]$. The reason is that the unknown functions considered in this thesis are continuous, then tubes such as the one represented on Figure 2.8 are not required and for each piece $[x_i] \mapsto [u_i]$ of the tube definition, the enclosure is guaranteed which improve the enclosure of the local evaluation.

$$\forall u \in [u],\ \forall x \in [x_i],\ u(x) \in [u_i] \tag{2.91}$$

When $[u]$ is evaluated on real values x covered by several pieces $[x_i] \mapsto [u_i]$ and $[x_j] \mapsto [u_j]$, $[u](x)$ belongs to $[u_i]$ and $[u_j]$. For example, on Figure 2.11b, the value $x = d$ is part of two segments of the function $[u]$ Eq. (2.73), then for all u in $[u]$:

$$\underline{u}(d) \leq u(d) \leq \overline{u}(d) \quad \Rightarrow \quad \begin{matrix} \underline{u_3}(d) \\ \underline{u_4}(d) \end{matrix} \leq u(d) \leq \begin{matrix} \overline{u_3}(d) \\ \overline{u_4}(d) \end{matrix} \quad \Rightarrow \quad \underline{u_3}(d) \leq u(d) \leq \overline{u_4}(d) \tag{2.92}$$

However, when functions contained in tubes are not necessarily continuous, the local evaluation has to be computed with the global evaluation over the degenerate interval.

When the value x is not a single real value but a set of real values, such that an interval $[x]$, if the latter is covered by several pieces $[x_i] \mapsto [u_i]$ and $[x_j] \mapsto [u_j]$, a part of $[u]([x])$ belongs to $[u_i]$ while the other part belongs to $[u_j]$. For example, on the tube $[u]$ that is defined previously Eq. (2.73 and (Fig. 2.11b), let $[x] = [(d+e)/2; (e+f)/2])$ an interval which intersects two segments: $[d; e]$ and $[e; f]$. Then for all function u enclosed by $[u]$:

$$\begin{aligned}\underline{u}([x]) \leq u([x]) \leq \overline{u}([x]) \quad &\Rightarrow \quad \begin{cases} \underline{u_4} \leq u([\underline{x}; e]) \leq \overline{u_4} \\ \underline{u_5} \leq u([e; \overline{x}]) \leq \overline{u_5} \end{cases} \\ &\Rightarrow \quad \min(\underline{u_4}, \underline{u_5}) \leq u([x]) \leq \max(\overline{u_4}, \overline{u_5}) \\ &\Rightarrow \quad \underline{u_5} \leq u([x]) \leq \overline{u_4}\end{aligned} \tag{2.93}$$

2.5.4.4 Warning:

Let $[x] = [d; e]$, then $[c; d] \cap [x] = d \neq \emptyset$ as well as $[e; f] \cap [x] = e \neq \emptyset$, then with the previous formulation of the evaluation Eq. (2.90):

$$[u]([x]) = [u_3] \cup [u_4] \cup [u_5] \tag{2.94}$$

This problem can be overtaken by extracting from $[u]$ a new tube $[v]$ on the domain $[x]$ Eq. (2.96), using a reduction method then makes the result canonic Eq. (2.70), (2.77), (2.75), (2.76), and finally get the hull of the tube :

$$[u]([x]) = hull(canonic(extract([u], [x]))) \tag{2.95}$$

2.5.4.5 Definition: Extraction

Let $[u]$ a tube of k pieces $[x_i] \mapsto [u_i]$ and $[x]$ an interval subset of $\mathbb{D}_{[u]}$. The extraction of a tube $[u]$ on a domain $[x]$ is defined as below.

$$extract([u], [x]) = \bigcup_{[x_i]\mapsto[u_i]\in[u]} extract([x_i], [u_i], [x]) \tag{2.96}$$

where

$$extract([x_i], [u_i], [x]) = \left\{ \begin{array}{ll} \emptyset & \textbf{if } [x_i] \cap [x] = \emptyset \\ ([x_i] \cap [x]) \mapsto [u_i] & \textbf{otherwise} \end{array} \right\} \tag{2.97}$$

2.5.4.6 Properties of tubes

Let the tube $[u]$ in $\mathbb{IF}$, a piecewise interval function.

- Tubes are monotonic; Let $[U]$ in $\mathbb{IF}$ a piecewise interval function such that $[u] \subseteq [U]$. Then
$$\forall [x] \subseteq \mathcal{D}_{[u]},\ [u]([x]) \subseteq [U]([x]) \tag{2.98}$$
- The evaluation of tubes is monotonic in terms of inclusion. Let $[x]$ and $[X]$ two intervals subset of $\mathbb{D}_{[u]}$, then
$$[x] \subseteq [X] \quad \Rightarrow \quad [u]([x]) \subseteq [u]([X]) \tag{2.99}$$

2.5.5 Arithmetic operators and usual functions

Just like interval arithmetic, arithmetic operators have to be overwritten and extended to deal with tubes in order to build tube arithmetic. Also, tubes can be used as interval extended functions.

Let $[u] = \{([x_i], [u_i])\}_{1\leq i\leq l}$ and $[v] = \{([y_j], [v_j])\}_{1\leq j\leq q}$ two non-empty tubes. The result of the binary arithmetic operation $\diamond$ in $\{+, -, \times, \div\}$ is a new tube $[w]$ such that:

$$[w] = [u] \diamond [v] = \{([z_1], [w_1]), \ldots, ([z_p], [w_p])\} \tag{2.100}$$

$$\mathbb{D}_{[w]} = \mathbb{D}_{[u]} \cap \mathbb{D}_{[v]} = \bigcap_{\substack{[x_i]\in\mathbb{D}_{[u]} \\ [y_j]\in\mathbb{D}_{[v]}}} [x_i] \cap [y_j] \tag{2.101}$$

The result $[w]$ is defined such that for all $([x_i],[u_i]) \in [u]$ and for all $([y_j],[v_j]) \in [v]$, a pair $([z_k],[w_k])$ is defined, as follows, for each non-empty intersection $[x_i] \cap [y_j]$:

$$\begin{aligned} [z_k] &= [x_i] \cap [y_j] \\ [w_k] &= [u_i] \diamond [v_j] \end{aligned}$$

where $\diamond$ is the equivalent interval operator.

2.5.5.1 Definition: Operations over a real value

Let $x \in \mathbb{R}$ a real value, then the operations:

$$\begin{aligned} [u] \diamond [v](x) &= [u](x) \diamond [v](x) && (2.102) \\ [u] \circ [v](x) &= [v]([u](x)) && (2.103) \\ shift([u],\alpha)(x) &= [u](x+\alpha) && (2.104) \end{aligned}$$

2.5.5.2 Warning

If the binary operation available on tubes $\diamond$ is accurate over real values, such is not the case over intervals or multi-intervals.

$$[u] \diamond [v]([x]) \subseteq [u]([x]) \diamond [v]([x]) \tag{2.105}$$

2.5.5.3 Example: Let $[u]$ and $[v]$ two tubes such as:

$$[u] = \begin{bmatrix} [0;1] \mapsto [3;4] \\ [1;3] \mapsto [8;9] \end{bmatrix} \quad \textbf{and} \quad [v] = \begin{bmatrix} [0;1] \mapsto [1;5] \\ [1;2] \mapsto [-2;0] \\ [2;3] \mapsto [2;3] \end{bmatrix} \tag{2.106}$$

Then the result of the operation $[u]+[v]$ over the interval $[0;3]$ is such that

$$([u]+[v])([0;3]) \subset [u]([0;3]) + [v]([0;3]) \tag{2.107}$$

where the computations of $[u]([0;3])$ and $[v]([0;3])$ are detailed below Eq. (2.108) as well as the computation of $([u]+[v])([0;3])$ Eq. (2.109).

$$\begin{aligned} [u]([0;3]) &= \{[3;4],[8;9]\} \\ [v]([0;3]) &= \{[-2;0],[1;5]\} \\ [u]([0;3]) + [v]([0;3]) &= \{[3;4],[8;9]\} + \{[-2;0],[1;5]\} \\ &= \{[1;14]\} \end{aligned} \tag{2.108}$$

$$\begin{aligned} ([u]+[v])([0;3]) &= \begin{bmatrix} [0;1] \mapsto [4;9] \\ [1;2] \mapsto [6;9] \\ [2;3] \mapsto [10;12] \end{bmatrix} ([0;3]) \\ &= \{[4;9],[10;12]\} \end{aligned} \tag{2.109}$$

2.5.5.4 Definition: Integration

Tubes can be integrated as any real function and, as mentioned before Eq. (2.57), they represent functions which are not necessarily defined on the domain $\mathbb{D}_{[u]}$. However their integration on an interval $[a;b]$ subset of $\mathbb{D}_{[u]}$ takes exclusively into account the functions u in $[u]$ defined over the same interval $[a;b]$. Trivially

$$\begin{aligned}\int_a^b [u](x)dx &= \left[\int_a^b \underline{u}(x)dx; \int_a^b \overline{u}(x)dx\right] && (2.110)\\ &= \sum_{[x_i]\mapsto[u_i]} (width([a;b]\cap[x_i]) \times [u_i]) && (2.111)\end{aligned}$$

2.5.5.5 Definition: Composition with usual functions

The usual functions $f \in \{abs, cos, sin, tan, exp, log, power, root, \dots\}$ can be composed with tubes $[u]$, in order to create a new tube $[z]$ Eq. (2.112).

$$[z] = [u] \circ f = f([u]) = f\left(\begin{bmatrix}[x_1] \mapsto [u_1]\\ \vdots \\ [x_k] \mapsto [u_k]\end{bmatrix}\right) = \begin{bmatrix}[x_1] \mapsto f([u_1])\\ \vdots \\ [x_k] \mapsto f([u_k])\end{bmatrix} \tag{2.112}$$

In this section, the tube arithmetic has been presented so that, the advantages the guaranteed arithmetics have shown, on the set of reals, could be applied on the set of unary functions. A specific representation of the tubes has been discussed through the use of intervals and their variations (boxes, multi-interval, multi-boxes). A set of methods has been used to get control on internal tubes representations in order to deal with their canonic form. Tubes will be used in the next chapters to represent sets of functions such as the results of guaranteed integration (of ordinary differential equations) and constraint satisfaction problems on sets of functions [Raıssi et al., 2004], [Goldsztejn et al., 2011].

2.6 Conclusion

In this chapter basic notions were introduced and mathematical concepts of guaranteed arithmetics have been defined. Through the use of interval arithmetic (Section 2.2), the advantages (safe computations, robustness, infinite sets representation, etc) and drawbacks (over-approximation) of these approaches have been thought out. Uncertain values were able to be represented more accurately using multi-interval methods (Section 2.3). In order to do so, time computation and memory requirements are increased. The affine arithmetic (Section 2.4) has been briefly described and is useful to perform good quality results of operation but require much memory and computation time.

Finally, tube arithmetic (Section 2.5) has been introduced, in order to extend these methods and these tools to sets of functions and uncertain functions. Several representations have been discussed such that the affine and the interval representations. In this work, the variation of the interval representation defined with multi-interval boxes is used under its canonical form.

In the next chapters, interval and tube arithmetics are used in order to perform guaranteed integration (Chapters 3 and 5) of (piecewise) ordinary differential equations, the solutions of which will be represented as tubes. Then, these arithmetic concepts will be used to solve both constraint satisfaction problems (Chapter 4) using contractors and optimization problems using global optimization methods (Chapter 6).

Guaranteed Integration of Ordinary Differential Equations 3

Science is a differential equation. Religion is a boundary condition.

– *Alan Turing*

Contents

3.1 Introduction

Astronomy, physics and chemistry have found in differential equations the most natural way to express their laws. These systems of equations are now widely used to model dynamic behaviors in engineering, economy, ecology, epidemiology, neuroscience, etc. Thus the resolution of such systems is used in a large number of applications, but most of the differential equations do not have an explicit solution, therefore many approaches can be considered.

Qualitative methods sometimes lead to subsets of solutions which provide some information about the differential equations. However, only a few families of differential equations allow a full theoretical treatment from an analytic point of view — systems of linear differential equations acting as such. In what follows, attention is focused on non-linear differential equations which are not treated on an analytic point of view.

To begin with, this chapter introduces the *Ordinary Differential Equations* in Section 3.2. Then, in Section 3.3, an algorithm is discussed in order to rigorously integrate these differential equations. It is based on two global methods that compute a global enclosure (Section 3.3.2) and a local enclosure (Section 3.3.3). These two methods can be improved using specific processes such that the QR-Factorization that is detailed in Section 3.3.4. Also, a piecewise evaluation and a pruning method that are based on the backward consistency argument are presented in Section 3.4.

3.2 Ordinary Differential Equations (ODEs)

3.2.1 Definition

Ordinary differential equations (ODEs) map out a large number of behaviors which evolve continuously with time. Let $t \in \mathbb{R}$ and $u \in \mathbb{R} \to \mathbb{R}$, so that $u(t)$ is an unknown function in t. Let $d \in \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ a function of t, $u(t)$ and its derivatives. An explicit *Ordinary Differential Equation* (ODE) of order n, is an equation of the form

$$u^{(n)}(t) = d(t, u(t), u'(t), u''(t), \dots, u^{(n-1)}(t)) \tag{3.1}$$

where variable u is called the dependent variable and t the independent variable or time.

Let a family of n unknown functions $u_i \in \mathbb{R} \to \mathbb{R}$ and a set of n applications $d_i \in \mathbb{R}^{n+1} \to \mathbb{R}$. $\boldsymbol{\Theta}$ defines an n-dimensional first-order ODE system.

$$\boldsymbol{\Theta} \begin{cases} u_1'(t) & = & d_1(t, u_1(t), \dots, u_n(t)) \\ & \vdots & \\ u_n'(t) & = & d_n(t, u_1(t), \dots, u_n(t)) \end{cases} \tag{3.2}$$

The system of equations can be represented in a clearer notation: the vectorial notation Eq. (3.3).

$$\mathbf{u}'(t) = \mathbf{d}(t, \mathbf{u}(t)) \tag{3.3}$$

The ordinary differential equation Eq. (3.3) is known as non-autonomous because the function $\mathbf{d}$ depends on the time variable t. In other cases, systems are considered autonomous, meaning their behaviors depend exclusively on their states so that they are conservative along the time window. Most of the natural dynamic systems are autonomous.

$$\mathbf{u}'(t) = \mathbf{d}(\mathbf{u}(t)) \tag{3.4}$$

An ODE system can be controlled by an external input $\mathbf{x} \in \mathbb{R}^p$ such that

$$\mathbf{u}'(t) = \mathbf{d}(t, \mathbf{x}, \mathbf{u}(t)) \tag{3.5}$$

These systems are parametric, they are used to model dynamics depending on outer conditions. When expressing complex problems in which the ODEs are led by other constraints, such systems are adequate, using the parametric variables $\mathbf{x}$ to share information. These systems can be reduced into an equivalent ODE with a new evolution function $\mathbf{d_2}(t, \mathbf{u}(t)) := \mathbf{d}(t, \mathbf{x}, \mathbf{u}(t))$ because the value of $\mathbf{x}$ is constant all along the integration of the dynamic.

In this thesis, ODEs are defined over a time windows $\mathcal{T}$ that can be a restriction of $\mathbb{R}$, then $t \in \mathcal{T} \subseteq \mathbb{R}$.

3.2.2 Reductions

Any n-th order differential equation can be reduced into an n-dimensional first-order ODEs system $\boldsymbol{\Theta}$ Eq. (3.2) through the use of n additional variables. This property is very important since first-order ODEs are easier to solve numerically.

3.2.2.1 Example: Reduction of an ODE into a first order differential system

Let the following ordinary differential equation of order 5:

$$u^{(5)}(t) + cos(u'(t))u^{(4)}(t) - sin(tu^{(3)}(t)) + 8tu^{(2)}(t)^2 + exp(u(t)) = log(t)$$

Then, introducing the variables $u_1 = u$, $u_2 = u'$, $u_3 = u^{(2)}$, $u_4 = u^{(3)}$, $u_5 = u^{(4)}$ the ODE can be turned into the following system:

$$\begin{array}{lclcl}
u_1' & = & u_2 & = & u' \\
u_2' & = & u_3 & = & u^{(2)} \\
u_3' & = & u_4 & = & u^{(3)} \\
u_4' & = & u_5 & = & u^{(4)} \\
u_5' & = & -cos(u')u^{(4)} + sin(tu^{(3)}) - 8tu^{(2)^2} - exp(u) + log(t) & = & u^{(5)} \\
 & = & -cos(u_2)u_5 + sin(tu_4) - 8tu_3{}^2 - exp(u_1) + log(t) & &
\end{array}$$

Using the same technique, any non-autonomous system may become autonomous through the enlargement of the system $\mathbf{\Theta}$, adding a new variable u_0 such that:

$$\begin{aligned}
\mathbf{u_0'}(t) &= 1 \\
\mathbf{u}'(t) &= \mathbf{d}(u_0(t), \mathbf{u}(t))
\end{aligned}$$

3.2.3 Examples of famous ODE Models

Ordinary differential equations are widely used to describe natural phenomena and have immediate repercussions in many fields. In order to illustrate their usefulness, here are some models.

3.2.3.1 Example: Newton

The first use of ODE was introduced by *Newton* in the XVIIth century, so as to define the base model of the cinematic. It consists in a second order differential equation in which the acceleration of a point equals the sum of external forces that are applied on it.

$$\sum F(x) = m \cdot a = m \cdot \frac{d^2x}{dt} \Rightarrow \begin{cases} x'(t) = v(t) \\ v'(t) = \dfrac{1}{m} \sum F(x) \end{cases}$$

Many physical models can be described through the various expressions of F.

$$\begin{aligned}
F(x) &= -kx \\
F(x) &= -m\frac{g}{l}sin(x) \\
F(x) &= \lambda(x)x \\
F(x) &= -gradV(x)
\end{aligned}$$

3.2.3.2 Example: Field-Noyes

Consider a chemical reaction involving three chemical components such that

$$A + B \to C$$

where each product has a concentration a, b and c. Then the consumption and the production of each one of them can be modeled as

$$\frac{dc}{dt} = -\frac{da}{dt} = -\frac{db}{dt}$$

in which the reaction rate of the form $Ka^{\alpha}b^{\beta}c^{\gamma}$ has its values K, α, β and γ chosen according to the experiments. More peculiarly, the Belousov-Zhabotinskii reaction can be modeled using the Field-Noyes model [Hastings and Murray, 1975]:

$$\begin{cases} x'(t) = \dfrac{qy - xy + x(1-x)}{\varepsilon} \\ y'(t) = \dfrac{-qy - xy + 2rz}{\lambda} \\ z'(t) = x - z \end{cases}$$

where ε, λ, q and r are parametric values.

3.2.3.3 Example: Lotka-Volterra

The *Lotka-Volterra* ODE systems are frequently used to describe biological systems composed of two species. The variable x (resp. y) represents the density of preys (resp. predators).

$$\begin{cases} x'(t) = x(\alpha_1 - \alpha_2 y) \\ y'(t) = y(\beta_1 x - \beta_2) \end{cases}$$

where α and β, being positive reals, define the interaction between the two species.

The Lotka–Volterra system of equations is a particular Kolmogorov model which defines a more general framework [Cheng et al., 1982]. The Kolmogorov model manages the dynamics of ecological systems by following several criteria such as predator-prey interactions, competitions, diseases, etc.

3.2.4 Solution of an Initial Value Problem (IVP)

A solution of an ODE is a function $\mathbf{u}(t)$ such that the system

$$\mathbf{u}'(t) = \mathbf{d}(t, \mathbf{x}, \mathbf{u}(t)) \tag{3.6}$$

is respected for all t in $\mathcal{T}$, which is the application domain of the constraint.

3.2.4.1 Example: Explicit solution of an ODE

Let the following ordinary differential equation:

$$\begin{aligned} u_1'(t) &= u_2(t) \\ u_2'(t) &= u_3(t) \\ u_3'(t) &= 0 \end{aligned}$$

Then $u_3(t)$ is a constant function for all t in $\mathcal{T} = [0;1]$. This system describes the movement of an object where u_1 represents the object's position, u_2 represents its velocity and u_3 its acceleration.

A solution to this system is $u_1(t) = u_2(t) = u_3(t) = 0$ for all t in $\mathcal{T}$. However it is not the only one because others can be found. The set of all solutions is defined by the following expression of $\mathbf{u}$:

$$\begin{cases} u_1(t) = u_{10} + u_{20} \cdot t + \frac{t^2}{2} u_{30} \\ u_2(t) = u_{20} + u_{30} \cdot t \\ u_3(t) = u_{30} \end{cases}$$

where u_{10}, u_{20} and u_{30} are reals (initial conditions).

As seen in the previous example, an infinite number of functions are solutions to an ordinary differential equation $\mathbf{\Theta}$. The *Initial Value Problem* (IVP) is defined by adding an initial value $\mathbf{u_0}$ into the system.

$$\begin{aligned} \mathbf{u}'(t) &= \mathbf{d}(t, \mathbf{x}, \mathbf{u}(t)) \\ \mathbf{u}(t_0) &= \mathbf{u_0} \end{aligned} \tag{3.7}$$

Usually, to model most of the natural phenomena, the initial values are acquired through the experiments. The solution $\mathbf{u}^*$ is a continuous function such that $\mathbf{u}^*$ satisfies the following equations:

$$\mathbf{u}^{*\prime}(t) = \mathbf{d}(t, \mathbf{x}, \mathbf{u}^*(t)) \tag{3.8}$$

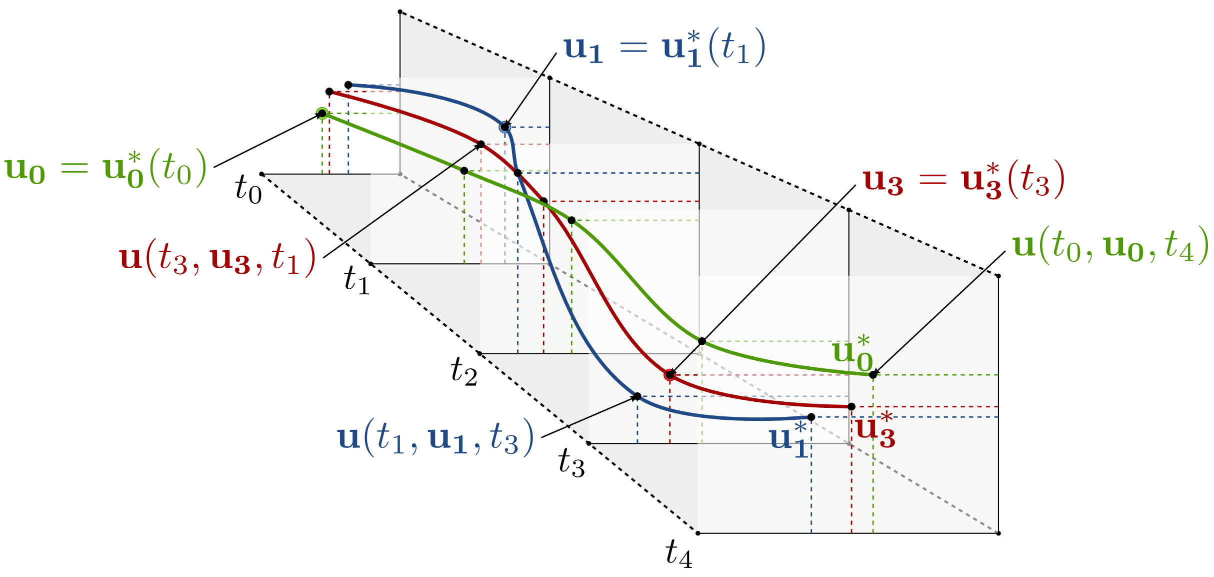
$$\mathbf{u}^*(t_0) = \mathbf{u_0} \tag{3.9}$$

From now on, the notation $\mathbf{u}(t_0, \mathbf{u_0}, t)$ will represent the value $\mathbf{u}^*(t)$ of the solution $\mathbf{u}^*$ at t from the initial value $\mathbf{u}^*(t_0) = \mathbf{u_0}$. Note that $\mathbf{u}^*(t)$ also depends on the value of $\mathbf{x}$ which is not considered in $\mathbf{u}(t_0, \mathbf{u_0}, t)$. The reason is that the value $\mathbf{x}$ is constant all along the integration (from t_0 to t_f), thus it can be replaced by an equivalent constant value in the expression of $\mathbf{d}$.

3.2.4.2 Analysis of Figure 3.1:

This figure depicts three different solutions $\mathbf{u_0^*}$, $\mathbf{u_1^*}$ and $\mathbf{u_3^*}$ of an ODE system defined from $t = t_0$ to $t = t_4$

$$\mathbf{u}^{(n)}(t) = \mathbf{d}(t, \mathbf{u}(t), \mathbf{u}^{(1)}(t), \mathbf{u}^{(2)}(t), \ldots, \mathbf{u}^{(n-1)}(t)) \tag{3.10}$$



$\mathbf{u_1} = \mathbf{u_1^*}(t_1)$
$\mathbf{u_0} = \mathbf{u_0^*}(t_0)$
t_0
$\mathbf{u_3} = \mathbf{u_3^*}(t_3)$
$\mathbf{u}(t_0, \mathbf{u_0}, t_4)$
$\mathbf{u}(t_3, \mathbf{u_3}, t_1)$
t_1
$\mathbf{u_0^*}$
t_2
$\mathbf{u}(t_1, \mathbf{u_1}, t_3)$
$\mathbf{u_1^*}$
$\mathbf{u_3^*}$
t_3
t_4

Figure 3.1: Solutions of various IVPs

when paired with three different initial value $\mathbf{u_0}$, $\mathbf{u_1}$ and $\mathbf{u_3}$.

Note that the initial value is not necessarily defined at the lowest valid time-step t_0. Here $\mathbf{u_1}$ and $\mathbf{u_3}$ are two initial values defined at t_1 and t_3. As a consequence, an initial value fix all the trajectory of the associated solution. Because the solution $\mathbf{u_3^*}$ go through $\mathbf{u_3}$ at t_3 it implies the solution $\mathbf{u_3^*}$ comes from $\mathbf{u}(t_3, \mathbf{u_3}, t_1)$ at t_1.

3.2.5 Existence and uniqueness conditions

Note that on Figure 3.1, the solutions $\mathbf{u_1^*}$, $\mathbf{u_2^*}$ and $\mathbf{u_3^*}$ exist. Moreover, the solution is unique when an initial value is defined. But an ODE may have no solution, a unique solution or an infinite number of solutions. For example, from the initial value $u(0) = 1$, the ODE

- $u'(t) = \sqrt{-u(t) - 2}$ has no solution,
- $u'(t) = 2u(t)$ has a unique solution $u(t) = 1 + t^2$,
- $u'(t) = \frac{u(t)-1}{t}$ has an infinite number of solutions $u(t) = 1 + \alpha t$ with $\alpha \in \mathbb{R}$.

In this thesis, the study is focused on ODEs such that the solutions exist and are unique, in order to guarantee the integration method. These two conditions can be ensured when the evolution function $\mathbf{d}(t, \mathbf{x}, \mathbf{u}(t))$ satisfies a few conditions.

- A sufficient condition to guarantee that the solution exists is that $\mathbf{d}$ is continuous.
- A sufficient condition to guarantee the uniqueness of the solution is that for all t, $\mathbf{x}$, $\mathbf{u_1}(t)$ and $\mathbf{u_2}(t)$ the following inequality is valid:

$$|\mathbf{d}(t, \mathbf{x}, \mathbf{u_1}(t)) - \mathbf{d}(t, \mathbf{x}, \mathbf{u_2}(t))| \leq L|\mathbf{u_1}(t) - \mathbf{u_2}(t)|$$

which is a uniform Lipschitz condition, where L is the Lipschitz constant.

Also, if the ODE is associated with several constraints, the solution could not exist. On the figure, there is no solution associated with the following set of constraints:

$$\mathbf{u}(t_1) = \mathbf{u_1} \quad \wedge \quad \mathbf{u}(t_3) = \mathbf{u_3} \tag{3.11}$$

However, an infinite number of different ODEs (with different evolution functions $\mathbf{d_2}$) which are not represented on this figure

$$\mathbf{u}^{(n)}(t) = \mathbf{d_2}(t, \mathbf{u}(t), \mathbf{u}^{(1)}(t), \mathbf{u}^{(2)}(t), \ldots, \mathbf{u}^{(n-1)}(t)) \tag{3.12}$$

exist such that a solution $\mathbf{u}^*$ could cross the points $\mathbf{u}^*(t_0) = \mathbf{u_0}$, $\mathbf{u}^*(t_1) = \mathbf{u_1}$ and $\mathbf{u}^*(t_3) = \mathbf{u_3}$.

In most initial value problems Eq. (3.7), the explicit expression of their solutions cannot be found. Indeed only some specific cases such as the linear differential equations can be solved thanks to dedicated tools.

Approximations of the solutions can nonetheless be computed through numerical approaches, as in the *Runge-Kutta* methods. They provide good solutions in practice and are therefore widely used in engineering; however, they might derive from the exact result for some problems. The use of the guaranteed methods which are based on a step by step integration scheme can get round this limit. Moreover, in this thesis, IVPs whose the initial values can be uncertain

$$\mathbf{u}'(t) = \mathbf{d}(t, \mathbf{x}, \mathbf{u}(t)) \tag{3.13}$$
$$\mathbf{u}(t_0) \in [\mathbf{u_0}] \tag{3.14}$$

have to be dealt with, which makes the classic numerical approaches obsolete. The notation $\mathbf{u}(t_0, \mathbf{u_0}, t_k)$ is then extended to interval initial value problems, where $[\mathbf{u}](t_0, [\mathbf{u_0}], t)$ represents the set of all the values $\mathbf{u}^*(t)$ of the solution $\mathbf{u}^*$ at t_k for all the initial values $\mathbf{u}^*(t_0) = \mathbf{u_0} \in [\mathbf{u_0}]$.

3.3 Guaranteed Integration Algorithm

In the following paragraphs is detailed an automatic, guaranteed and global approach from literature which is able to solve initial value problems that possess an uncertain initial value. The interval arithmetic is used to describe values and to perform operations. The provided result then describes a tube which guarantees the enclosure of the solutions (Fig. 3.2).

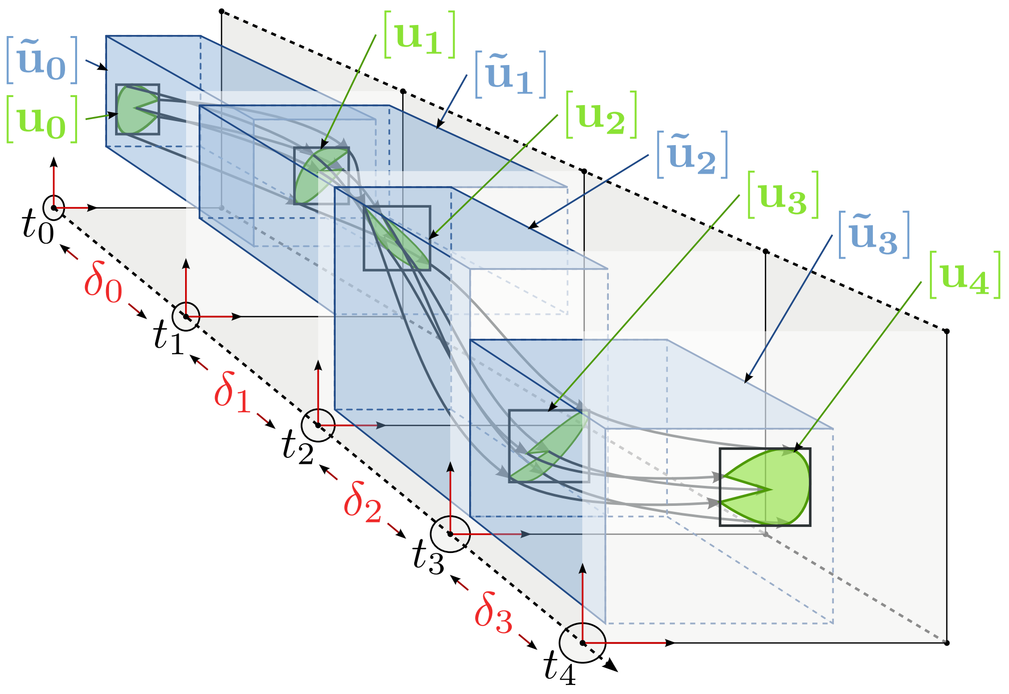


Figure 3.2: Enclosure of an ODE solution $\mathbf{u}^*$ returned by Algorithm 1

3.3.1 Strategy of the algorithm

The guaranteed integration algorithm (Alg. 1) introduced here is a two-step method that is quite similar to the predictor-corrector integration schemes. To sum things up briefly, the guaranteed integration algorithm is a sub-integration loop that defines the tube enclosing the solutions. A set of values t_i are defined on the time window $\mathcal{T} = [t_0; t_f]$ such that

$$t_0 < \cdots < t_{i-1} < t_i < t_{i+1} < \cdots < t_f$$

At each step t_i, two subroutines aim at computing first a global enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ and then a local enclosure $[\mathbf{u}_{\mathbf{i+1}}]$ of the solution to prepare the next iteration at t_{i+1}.

Because these subroutines require a great number of computations, the number of iterations should be kept as low as possible. Unfortunately, the global enclosures are frequently overestimated by the algorithms. However lower stepsizes $\delta_i = t_{i+1} - t_i$ improve the accuracy of enclosures. Therefore it is often necessary for stepsizes to be decreased in

Algorithm 1 : $\mathit{Integrate}(t_0 \in \mathbb{R}, [\mathbf{u_0}] \subseteq \mathbb{IR}^n, t_f \in \mathbb{R})$

```
1: int $i \leftarrow 0$ ; real $t \leftarrow t_0$
2: while $t \neq t_f$ do
3:     $([\mathbf{\tilde{u}_i}], t_{i+1}) \leftarrow \mathit{GlobalEnclosure}(t_i, [\mathbf{u_i}], t_f)$
4:     $[\mathbf{u_{i+1}}] \leftarrow \mathit{LocalEnclosure}(\langle t_0, \dots, t_i \rangle, \langle [\mathbf{u_0}] \dots [\mathbf{u_i}] \rangle, \langle [\mathbf{\tilde{u}_0}] \dots [\mathbf{\tilde{u}_i}] \rangle, t_{i+1})$
5:     $t \leftarrow t_{i+1}$ ; $i \leftarrow i + 1$
6: end while
7: return $\mathit{Tube}(t_0, [\mathbf{u_0}], [\mathbf{\tilde{u}_0}], t_1, [\mathbf{u_1}], [\mathbf{\tilde{u}_1}], \dots, t_{i-1}, [\mathbf{u_{i-1}}], [\mathbf{\tilde{u}_{i-1}}], t_i, [\mathbf{u_i}])$
```

such methods as to certify enclosures. As a consequence, it is essential to have optimal control of the stepsize δ_i [Nedialkov et al., 2001] in order to provide a tight enclosure $[\mathbf{u}]$ of the solution $\mathbf{u}^*$ in reasonable time.

Usually a tolerance threshold set by the user is used to configure how the stepsize should evolve. When the function $\mathbf{d}$ is stable, the stepsize δ_i increases, whereas it decreases when the variations rate of the function $\mathbf{d}$ becomes more intense (Fig. 3.3).

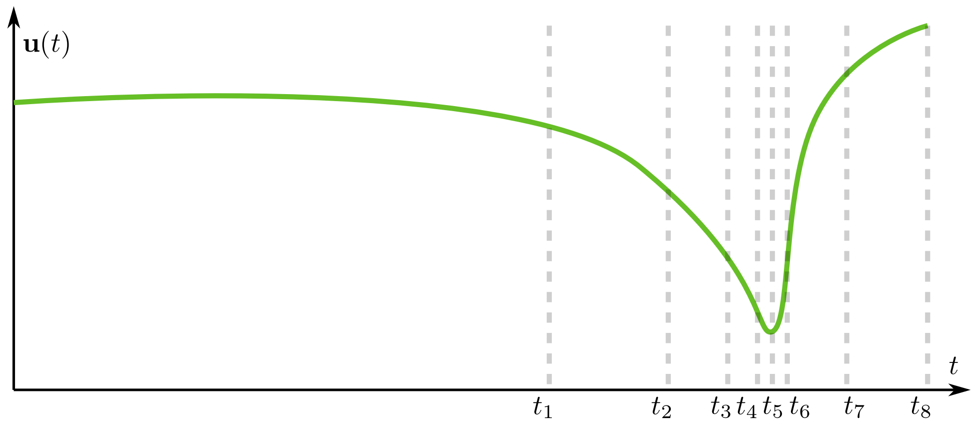


Figure 3.3: Adaptive stepsize

3.3.1.1 Step 1: Global enclosure (Fig. 3.4b)

The global enclosure method is the process of computing a stepsize δ_i and an *a priori* enclosure $[\mathbf{\tilde{u}_i}]$, such that $\mathbf{u}(t_i, \mathbf{u_i}, t)$ exists and is guaranteed to be in $[\mathbf{\tilde{u}_i}]$ for all t in $[t_i; t_i + \delta_i]$ and all $\mathbf{u_i}$ in $[\mathbf{u_i}]$.

$$\mathit{GlobalEnclosure}(t_i, [\mathbf{u_i}], t_f) \tag{3.15}$$

Usually, the global enclosure method uses *Picard's existence and uniqueness theorem* associated with *Banach's fixed point theorem.* Two methods based on these theorems are detailed in Section 3.3.2.

3.3.1.2 Step 2: Local enclosure (Fig. 3.4c)

The local enclosure method is the process of computing an enclosure $[\mathbf{u_{i+1}}]$ of $\mathbf{u_{i+1}} =$

$\mathbf{u}(t_i, \mathbf{u_i}, t_i + \delta_i)$ which is tighter than $[\tilde{\mathbf{u}}_\mathbf{i}]$ through a contraction process. The contraction can be computed from either an explicit or implicit schemes which has natural contractive properties.

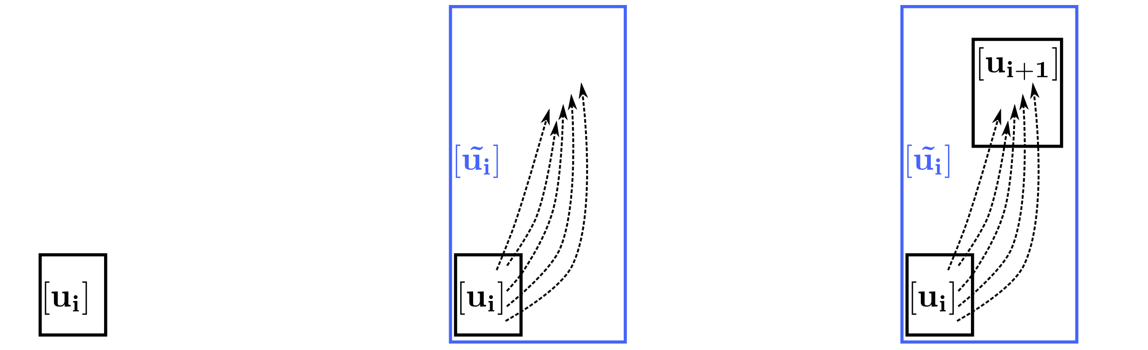
In the multistep approach, the local enclosure at time-step $t_{i+1} = t_i + \delta_i$ is computed from the previous enclosures at time-steps t_{i-k} to t_i Eq. (3.16) where k is a positive integer.

$$\mathit{LocalEnclosure}(\langle t_0 \dots t_i\rangle, \langle[\mathbf{u_0}] \dots [\mathbf{u_i}]\rangle, \langle[\tilde{\mathbf{u}}_\mathbf{0}] \dots [\tilde{\mathbf{u}}_\mathbf{i}]\rangle, t_{i+1}) \tag{3.16}$$

When only the last iteration of the multistep approach is used to compute the following enclosure ($k = 0$), the term single step approach is used Eq. (3.17).

$$\mathit{LocalEnclosure}(t_i, [\mathbf{u_i}], [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) \tag{3.17}$$

This contraction step is necessary to compute the tiniest enclosures that can be reached, minimizing the error and allowing higher stepsizes. In this perspective, several methods which can be combined are introduced in the following sections. In order to enclose and control the error term, the local enclosure is usually computed by Taylor series that are improved when under their Horner form and their Mean-Value form (Section 3.3.3). At the same time, the coordinate system can be adapted to limit what is called the wrapping effect (Section 3.3.4). Ultimately, the contraction may be improved thanks to consistency properties (Section 3.4.2) and the use of a piecewise evaluation (Section 3.4.1).



(a) Initial Value $[\mathbf{u_i}]$ (b) Global Enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ (c) Local Enclosure $[\mathbf{u_{i+1}}]$

Figure 3.4: Global Enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ and Local Enclosure $[\mathbf{u_{i+1}}]$

In the next sections, the method that is detailed does not consider parametric ODEs. However, the guaranteed integration of ODEs with interval parameters has been studied [Lin and Stadtherr, 2006b] and the methods presented in this thesis can be easily adapted to this generic case.

3.3.2 Global Enclosure

The global enclosure method Eq. (3.18) allows the user to compute both a time-step t_{i+1} and an a-priori guaranteed enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ of the solution $[\mathbf{u}](t_i, [\mathbf{u_i}], t)$ for all t in $[t_i; t_{i+1}]$, from any initial enclosure $[\mathbf{u_i}]$ at time t_i.

$$([\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) \leftarrow \mathit{GlobalEnclosure}(t_i, [\mathbf{u_i}], t_f) \tag{3.18}$$

The *GlobalEnclosure* method was introduced by [Moore, 1966],[Al-Abedeen and Arora, 1978] and improved more recently by [Corliss and Rihm, 1996] and [Nedialkov et al., 2001]. Based on the *Picard's Existence and Uniqueness Theorem* and on the *Banach's fixed point theorem for Operators*, this process certifies the return of a safe enclosure of the solution, by iterating on the *Picard Operator* Φ Eq. (3.22). At the same time, it provides a proof that the solution exists and is unique.

Theorem 3.3.1 (Banach's fixed point)
Let $\mathcal{M} = (\mathcal{Y}, d)$ a complete metric space, and $f : \mathcal{Y} \to \mathcal{Y}$ a contraction. Let $0 \leq \alpha < 1$ such that

$$\forall (x_1, x_2) \in \mathcal{Y}^2, d(x_1, x_2) \leq \alpha \times d(f(x_1), f(x_2)) \tag{3.19}$$

then f converges to a fixed point.

Theorem 3.3.2 (Picard-Lindelöf theorem)
Assuming that is $\mathcal{B}$ such that

$$\mathcal{B} = \left\{ (t, y(t)) \in \mathbb{R}^2 \;\middle|\; \begin{array}{ll} & |t - t_0| \leq \alpha \\ \wedge & |y(t) - y_0| \leq \beta \end{array} \right\}$$

and $f(t, y(t))$ a continuous function that satisfies a Lipschitz condition in y; then the problem

$$y'(t) = f(t, y(t)) \tag{3.20}$$
$$y(t_0) = y_0 \tag{3.21}$$

has a unique solution and the box $\mathcal{B}$ is ultimately an a priori bound of the solution y on $[t_0; t]$.

Proof is detailed in [Coddington, 2012], [Teschl, 2012]. The combination of those two theorems is at the origins of the Picard-Lindelöf operator Φ Eq. (3.22). The latter is used in the *First Order Enclosure* (FOE) algorithm [Al-Abedeen and Arora, 1978] [Eijgenraam, 1981] to define the global enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ and to prove both the existence and uniqueness of the solution.

3.3.2.1 Definition: Picard-Lindelöf operator

Let $\mathbf{\Theta}$ the system $\mathbf{u}' = \mathbf{d}(t, \mathbf{u})$, with $\mathbf{d}$ α-lipschitzian having first order partial derivatives (a Jacobian) on $[t_i, t_{i+1}]$. Let $[\mathbf{u_i}]$ and $[\mathbf{\tilde{u}_i}]$ two enclosures such that $[\mathbf{u_i}] \subseteq [\mathbf{\tilde{u}_i}]$. The Picard-Lindelöf operator $\mathbf{\Phi}$ is defined as below:

$$\begin{aligned}\mathbf{u_i} + \int_{t_i}^{t_{i+1}} \mathbf{d}(t, \mathbf{u_i}(t))dt &\in \mathbf{u_i} + \int_{t_i}^{t_{i+1}} [\mathbf{d}](t, [\mathbf{\tilde{u}_i^0}])dt \\ &\subseteq \mathbf{u_i} + [0; \delta_i][\mathbf{d}]([t_i; t_{i+1}], [\mathbf{\tilde{u}_i^0}]) \\ &\subseteq [\mathbf{u_i}] + [0; \delta_i][\mathbf{d}]([t_i; t_{i+1}], [\mathbf{\tilde{u}_i^0}]) = \mathbf{\Phi}([\mathbf{\tilde{u}_i^0}])\end{aligned} \tag{3.22}$$

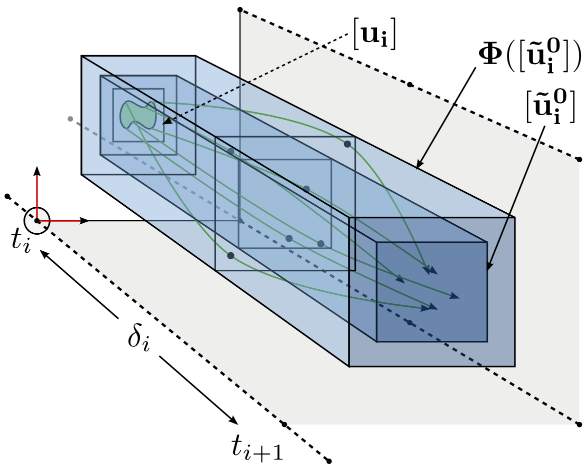
where $\delta_i = t_{i+1} -^{\uparrow} t_i$. If $\mathbf{\Phi}([\mathbf{\tilde{u}_i}]) \subseteq [\mathbf{\tilde{u}_i}]$ then

1. The system $\mathbf{\Theta}$ with the initial value $\mathbf{u}(t_i) \in [\mathbf{u_i}]$ has a unique solution $\mathbf{u}^*$ on $[t_i, t_{i+1}]$.
2. The box $\mathbf{\Phi}([\mathbf{\tilde{u}_i}])$ is an enclosure of $\mathbf{u}^*$ on $[t_i, t_{i+1}]$ with respect to $\mathbf{u}^*(t_i) \in [\mathbf{u_i}]$.

The operator $\mathbf{\Phi}$ previously defined, once used in an algorithm, can compute the stepsize δ_i and the global enclosure $[\mathbf{\tilde{u}_i}]$ of the solution $\mathbf{u}^*$ on $[t_i; t_{i+1}]$ where $t_{i+1} = t_i + \delta_i$. Starting from the initial stepsize $\delta_{i,0}$, this method creates successive boxes $[\mathbf{\tilde{u}_i^j}]$ such that

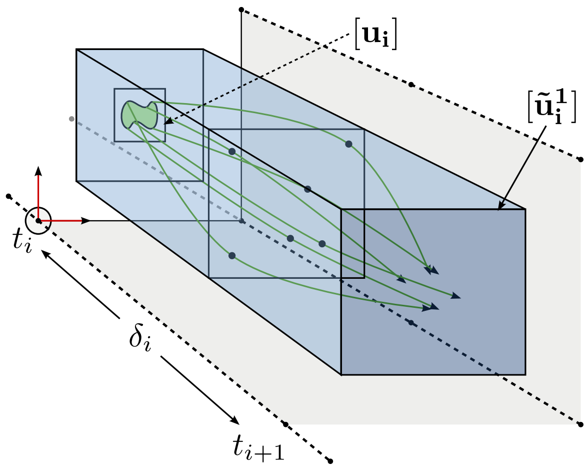
$$[\mathbf{\tilde{u}_i^{j+1}}] = \varepsilon(\mathbf{\Phi}([\mathbf{\tilde{u}_i^j}])) \supseteq \mathbf{\Phi}([\mathbf{\tilde{u}_i^j}]) \supseteq [\mathbf{\tilde{u}_i^j}] \tag{3.23}$$

where ε is a function used to inflate the enclosure as long as the final condition $\mathbf{\Phi}([\mathbf{\tilde{u}_i^j}]) \subseteq [\mathbf{\tilde{u}_i^j}]$ is not reached. When the final enclosure is infinite, the stepsize $\delta_{i,k}$ decreases and the process reboots with the new stepsize $\delta_{i,k+1} < \delta_{i,k}$.



$[\mathbf{u_i}]$
$\mathbf{\Phi}([\mathbf{\tilde{u}_i^0}])$
$[\mathbf{\tilde{u}_i^0}]$
t_i
δ_i
t_{i+1}

(a) $\Phi([\mathbf{\tilde{u}_i^0}]) \nsubseteq [\mathbf{\tilde{u}_i^0}]$

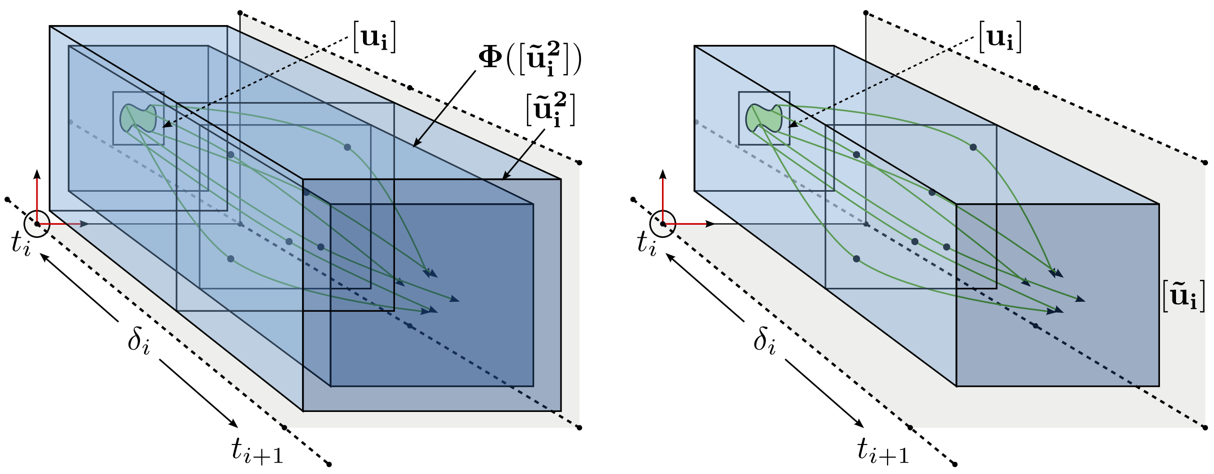


(b) $[\mathbf{\tilde{u}_i^1}] = \Phi([\mathbf{\tilde{u}_i^0}]) + \varepsilon([\mathbf{\tilde{u}_i^0}])$

Figure 3.5: Failure of the Picard iteration, ε-inflation

3.3.2.2 Analysis of Figures 3.5 and 3.6:

On Figure 3.5a, the initial box $[\mathbf{\tilde{u}_i^0}]$ and its image $\mathbf{\Phi}([\mathbf{\tilde{u}_i^0}])$ via the operator $\mathbf{\Phi}$ do not respect the condition $\mathbf{\Phi}([\mathbf{\tilde{u}_i^0}]) \subseteq [\mathbf{\tilde{u}_i^0}]$. Thus a new box $[\mathbf{\tilde{u}_i^1}]$ is computed through the ε-inflation so as to prepare the next iteration, as depicted on Figure 3.5b. On the contrary, on Figure 3.6a, the box $[\mathbf{\tilde{u}_i^2}]$ has enclosed its image $\mathbf{\Phi}([\mathbf{\tilde{u}_i^2}])$ via the operator $\mathbf{\Phi}$. Therefore the iteration can be stopped and the *a priori* enclosure $[\mathbf{\tilde{u}_i}] = \mathbf{\Phi}([\mathbf{\tilde{u}_i^2}])$ is guaranteed to enclose all the solutions $[\mathbf{u}](t_i, [\mathbf{u_i}], t)$ for all t in $[t_i; t_{i+1}]$, as depicted on Figure 3.6b.



(a) $\Phi([\mathbf{\tilde{u}_i^2}]) \subseteq [\mathbf{\tilde{u}_i^2}]$

(b) $[\mathbf{\tilde{u}_i}] = \Phi([\mathbf{\tilde{u}_i^2}])$

Figure 3.6: Success of the Picard iteration

Unfortunately, the Picard-Lindelöf operator Eq. (3.22) frequently decreases the step-size δ_i to provide a global enclosure $[\mathbf{\tilde{u}_i}]$ as it was the case in *AWA* [Lohner, 1992]. As a consequence, the number of iterations increases inside the global enclosure method. Similarly, the number of iterations also increases inside the global algorithm because there are more time-steps. This limit has been tackled in other tools such that *COSY VI* [Berz and Makino, 1998] using high order polynomial.

3.3.2.3 Definition: High Order Enclosure methods (HOE)

The high order enclosure method has been improved by [Nedialkov et al., 2001] and comes from the Corliss and Rihm's theorem [Corliss and Rihm, 1996]. It is an evolution of the FOE method – which is limited and restrains the stepsize used in the integration.

Assuming that $\mathbf{d}$ is uniformly Lipschitz continuous. Let two boxes $[\mathbf{u_i}] \subseteq \mathit{interior}([\mathbf{\tilde{u}_i}])$ and a stepsize δ_i such that

$$[\mathbf{u_i}] + \sum_{j=1}^{k-1} \frac{(t-t_i)^j}{j!}[\mathbf{d^{(j)}}](t_i, [\mathbf{u_i}]) + \frac{(t-t_i)^k}{k!}[\mathbf{d^{(k)}}]([t_i; t_i + \delta_i], [\mathbf{\tilde{u}_i}]) \subseteq [\mathbf{\tilde{u}_i}] \tag{3.24}$$

for all t in $[t_i; t_i + \delta_i]$ where $[\mathbf{d^{(j)}}]$ is the interval extension of the functions $\mathbf{d}^{(j)}$ defined by

$$\mathbf{d}^{(0)}(t, \mathbf{u}(t)) := u_i(t) \tag{3.25}$$

$$\mathbf{d}^{(i)}(t, \mathbf{u}(t)) := \frac{\partial \mathbf{d}^{(j-1)}}{\partial t \partial \mathbf{u}} \mathbf{d}(t, \mathbf{u}(t)) \tag{3.26}$$

Thus, for all t in $[t_i; t_i + \delta_i]$ and all $\mathbf{u_i}$ in $[\mathbf{u_i}]$, a unique solution $\mathbf{u}(t_i, \mathbf{u_i}, t)$ exists and is enclosed by $[\tilde{\mathbf{u}}_\mathbf{i}]$.

When $k = 1$, the HOE method is equivalent to the FOE method, thus the time-step is frequently decreases. The strength of the HOE resides in its ability to use higher orders and, consequently, larger stepsizes. However the computation of high order polynomials first requires the computation of high order differential functions and coefficients. Even through the automatic differentiation, which is a fast and efficient method to compute coefficients, the number of operations still remains far too high and additional methods (heuristics, lazy evaluation, contractions, pruning methods) are used to speed up the guaranteed integration

3.3.2.4 Remark: How to define δ_i and $[\tilde{\mathbf{u}}_\mathbf{i}]$?

The enclosure $[\mathbf{p_i}]$ of all the values of the polynomial part (without the error term) from any initial point $\mathbf{u_i}$ in $[\mathbf{u_i}]$ and at any time-step t in $[t_i; t_i + \delta_{i,0}]$ is defined as below

$$[\mathbf{p_i}] = [\mathbf{u_i}] + \sum_{j=1}^{k-1} \frac{[0; \delta_{i,0}]^j}{j!} [\mathbf{d^{(j)}}](t_i, [\mathbf{u_i}]) \tag{3.27}$$

The error term noted $[\mathbf{e_i}]$ and associated to $[\mathbf{p_i}]$ is defined such that

$$[\tilde{\mathbf{u}}_\mathbf{i}] = [\mathbf{p_i}] + [\mathbf{e_i}] \quad \textbf{and} \quad [\mathbf{u_i}] \in interior([\tilde{\mathbf{u}}_\mathbf{i}]) \tag{3.28}$$

Theorem 3.3.3 [Nedialkov et al., 2001]:
When the value δ_i is equal to the minimum value between $\delta_{i,0}$ and $\delta_{i,1}$, where $\delta_{i,1}$ is defined such that

$$\frac{[0; \delta_{i,1}]^k}{k!} [\mathbf{d^{(k)}}]([t_i; t_i + \delta_{i,0}], [\tilde{\mathbf{u}}_\mathbf{i}]) \subseteq [\mathbf{e_i}] \tag{3.29}$$

the solution $\mathbf{u}(t_i, \mathbf{u_i}, t)$ does exist, is unique and is bounded by $[\tilde{\mathbf{u}}_\mathbf{i}]$ for all $t \in [t_i; t_i + \delta_i]$ and for all $\mathbf{u_i} \in [\mathbf{u_i}]$.

3.3.2.5 Proof:

For any $\mathbf{u_i} \in [\mathbf{u_i}]$ and any $t \in [t_i; t_i + \delta_i]$ it is easy to verify that

$$\mathbf{u_i} + \sum_{j=1}^{k-1} \frac{(t - t_i)^j}{j!} [\mathbf{d^{(j)}}](t_i, \mathbf{u_i}) \in [\mathbf{p_i}] \tag{3.30}$$

Moreover

$$\begin{aligned}
\frac{(t - t_i)^k}{k!} [\mathbf{d^{(k)}}]([t_i; t_i + \delta_i], [\tilde{\mathbf{u}}_\mathbf{i}]) &\subseteq \frac{[0; \delta_i]^k}{k!} [\mathbf{d^{(k)}}]([t_i; t_i + \delta_i], [\tilde{\mathbf{u}}_\mathbf{i}]) && (3.31)\\
&\subseteq \frac{[0; \delta_{i,1}]^k}{k!} [\mathbf{d^{(k)}}]([t_i; t_i + \delta_{i,0}], [\tilde{\mathbf{u}}_\mathbf{i}]) && (3.32)\\
&\subseteq [\mathbf{e_i}] && (3.33)
\end{aligned}$$

Finally

$$\mathbf{u}(t_i, \mathbf{u_i}, t) \in \mathbf{u_i} + \sum_{j=1}^{k-1} \frac{(t-t_i)^j}{j!}[\mathbf{d^{(j)}}](t_i, \mathbf{u_i}) + \frac{(t-t_i)^k}{k!}[\mathbf{d^{(k)}}]([t_i; t_i+\delta_i], [\tilde{\mathbf{u}}_\mathbf{i}]) \tag{3.34}$$

$$\subseteq [\mathbf{p_i}] + [\mathbf{e_i}] = [\tilde{\mathbf{u}}_\mathbf{i}] \tag{3.35}$$

The enclosure of the error $[\mathbf{e_i}]$ is symmetric and not equal to zero on any of the components. Then, in [Nedialkov et al., 2001] the authors assume that the k-th coefficient, $\mathbf{d}^{(k)}$ does not vary much from one step to another. In other words:

$$\frac{[\mathbf{d^{(k)}}]([t_i, t_{i+1}], [\tilde{\mathbf{u}}_\mathbf{i}])}{k!} \approx \frac{[\mathbf{d^{(k)}}]([t_{i-1}, t_i], [\tilde{\mathbf{u}}_\mathbf{i-1}])}{k!} \tag{3.36}$$

Consequently, the error $[\mathbf{e_i}]$ can be approximated using the previous global enclosure method.

$$[\mathbf{e_i}] = \alpha \frac{\delta_{i,0}^k}{k!}[-\beta_\mathbf{i-1}; +\beta_\mathbf{i-1}] \tag{3.37}$$

where $\beta_\mathbf{i}$ is the magnitude Eq. (2.4) of the error on each component of the previous global enclosure and α is a coefficient used to increase the probability of having a correct enclosure of the error term.

$$\beta_\mathbf{i} = magnitude([\mathbf{d^{(k)}}]([t_i; t_i+\delta_i], [\tilde{\mathbf{u}}_\mathbf{i}])) \tag{3.38}$$

3.3.2.6 Stepsize control

The HOE method allows larger stepsizes than the FOE method. Thus the stepsize δ_i can be increased when the function $\mathbf{d}$ is stable. That is why it is necessary to have an efficient adaptive stepsize to use the HOE algorithm at its full potential. A relevant strategy consists in computing an initial stepsize δ_0 and then, at each step, to update the initial value of the next stepsize δ_{i+1} according to the situation at t_i with the stepsize δ_i. Some adaptive stepsize methods are introduced in [Nedialkov et al., 2001]. One of these is easy to put into practice: it consists in considering a linear combination of the stepsizes $\delta_{i,0}$ and $\delta_{i,1}$ as the basis for the next stepsize:

$$\delta_{i+1,0} = (1-\lambda)\delta_{i,0} + \lambda.\delta_{i,1} \tag{3.39}$$

with $0 \leq \lambda \leq 1$. The value $\lambda = 0$ induces a constant stepsize, $\delta_{i,0} = \delta_{0,0}$ for all stepsizes i, which is not efficient. On the other side, the strategy with $\lambda = 1$ ($\delta_{i+1,0} = \delta_{i,1}$) is interesting because during the i-th step, the method proved with ($\delta_i = \delta_{i,1}$) or could have proven with ($\delta_i = \delta_{i,0}$) that the enclosure exists, is unique and guaranteed (using the adaptive stepsize $\delta_{i,1}$).

3.3.3 Local Enclosure

The local enclosure method is used once an a-priori guaranteed enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ is defined on a time window $[t_i; t_{i+1}]$. It basically consists in the contraction of the previously mentioned a-priori guaranteed enclosure towards a tighter guaranteed enclosure $[\mathbf{u}_{\mathbf{i+1}}]$ of the solution $[\mathbf{u}](t_i, [\mathbf{u}_\mathbf{i}], t_{i+1}])$ at t_{i+1}.

$$[\mathbf{u}_{\mathbf{i+1}}] \leftarrow \mathit{LocalEnclosure}(t_i, [\mathbf{u}_\mathbf{i}], [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) \tag{3.40}$$

The local enclosure method introduced here is based on the Taylor models. They are widely used in guaranteed integration schemes [Nedialkov et al., 1999] because they have the ability to compute guaranteed bounds for the solutions of ODE. The guarantee of the enclosures provided by the Taylor series rests on

- the use of interval arithmetic to assure the operations' results
- the knowledge of an *a priori* guaranteed enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ that bounds the Taylor error term

However, and because of the dependency problem of interval arithmetic, the contraction provided by the local enclosure method is limited by the overestimation of operations.

3.3.3.1 Definition: Taylor Model

Let $[\mathbf{u}_\mathbf{i}]$ the initial value at time t_i and $[\tilde{\mathbf{u}}_\mathbf{i}]$ a guaranteed enclosure of the solution $\mathbf{u}^*$ over the integration time-step δ_i. A local enclosure $[\mathbf{u}_{\mathbf{i+1}}]$ of the solution at $t_{i+1} = t_i + \delta_i$ and tighter than $[\tilde{\mathbf{u}}_\mathbf{i}]$ can be computed using the *Taylor* model. This model is made of two parts: the polynomial (noted T_p) and the error term (noted T_e) (Fig. 3.7).

$$\mathit{LocalEnclosure}(t_i, [\mathbf{u}_\mathbf{i}], [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) := T_p(t_i, [\mathbf{u}_\mathbf{i}], t_{i+1}) + T_e(t_i, [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) \tag{3.41}$$

with

$$T_p(t_i, [\mathbf{u}_\mathbf{i}], t_{i+1}) := [\mathbf{u}_\mathbf{i}] + \frac{\delta_i}{1!}[\mathbf{d}^{(\mathbf{1})}](t_i, [\mathbf{u}_\mathbf{i}]) + \cdots + \frac{\delta_i^{k-1}}{(k-1)!}[\mathbf{d}^{(\mathbf{k-1})}](t_i, [\mathbf{u}_\mathbf{i}]) \tag{3.42}$$

$$T_e(t_i, [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1}) := \frac{\delta_i^k}{k!}[\mathbf{d}^{(\mathbf{k})}]([t_i, t_{i+1}], [\tilde{\mathbf{u}}_\mathbf{i}]) \tag{3.43}$$

where $\delta_i = t_{i+1} - t_i$ and $[\mathbf{d}^{(\mathbf{j})}]$ is an interval-arithmetic extension of $\mathbf{d}^{(j)}$ inductively defined by

$$d_i^{(1)}(t, u_1(t), \ldots, u_n(t)) := d_i(t, u_1(t), \ldots, u_n(t)) \tag{3.44}$$

$$d_i^{(j)}(t, u_1(t), \ldots, u_n(t)) := \frac{\partial d_i^{(j-1)}}{\partial t} + \sum_{m=1}^{n} \frac{\partial d_i^{(j-1)}}{\partial u_m} d_m(t, u_1(t), \ldots, u_n(t)) \tag{3.45}$$

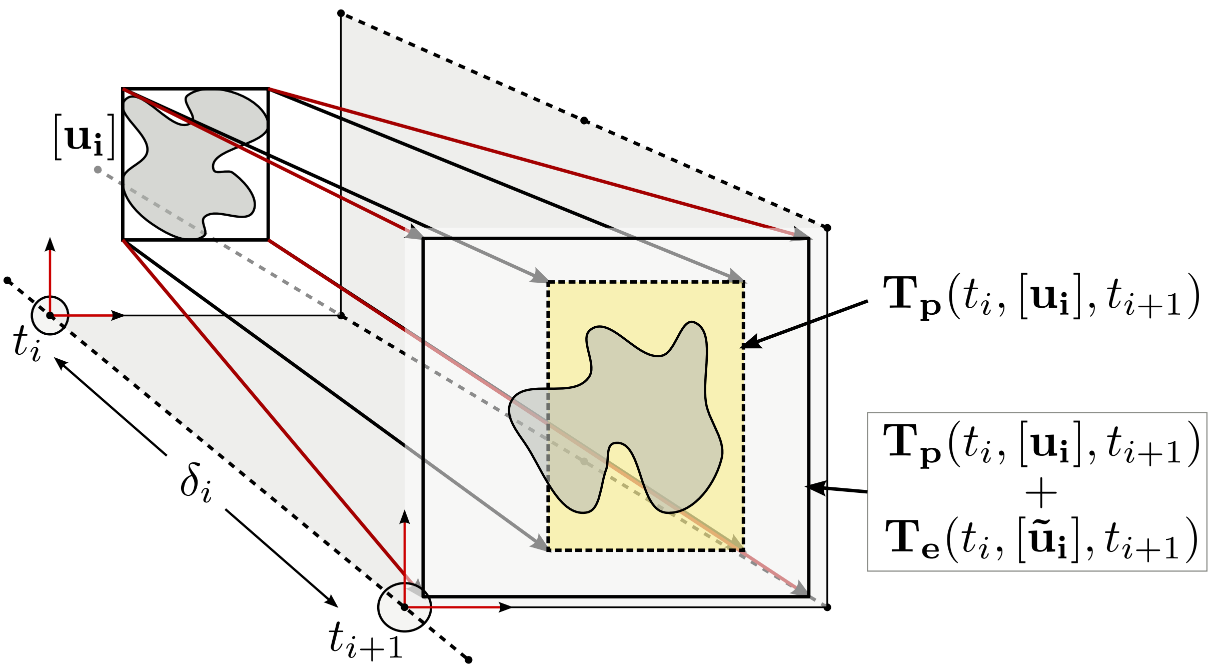

$[\mathbf{u_i}]$
t_i
δ_i
t_{i+1}
$\mathbf{T_p}(t_i, [\mathbf{u_i}], t_{i+1})$
$\mathbf{T_p}(t_i, [\mathbf{u_i}], t_{i+1})$ $+$ $\mathbf{T_e}(t_i, [\tilde{\mathbf{u}}_\mathbf{i}], t_{i+1})$

Figure 3.7: Enclosure $[\mathbf{u_{i+1}}]$ computed with Taylor model

3.3.3.2 Definition: Horner Form

The *Horner form* can be used to strengthen the relationship between variables, thus restraining the dependency issue (that is caused by the use of interval arithmetic) on each one of them.

$$
\begin{aligned}
T_p(t_i, [\mathbf{u_i}], t_{i+1}) &:= [\mathbf{u_i}] + \delta_i \times \frac{[\mathbf{d^{(1)}}](t_i, [\mathbf{u_i}])}{1!} + \cdots + \delta_i^k \times \frac{[\mathbf{d^{(k-1)}}](t_i, [\mathbf{u_i}])}{(k-1)!} \\
&:= [\mathbf{u_i}] + \delta_i \left([\mathbf{d^{(1)}}](t_i, [\mathbf{u_i}]) + \frac{\delta_i}{2} \left(\cdots + \frac{\delta_i}{k-1} \left([\mathbf{d^{(k-1)}}](t_i, [\mathbf{u_i}]) \right) \ldots \right) \right)
\end{aligned}
$$

The Horner form can also be used to compute the enclosure $[\mathbf{p_i}]$ in the HOE algorithm [Nedialkov et al., 2001] used in the global enclosure method.

3.3.3.3 Definition: Mean Value Form

Furthermore, the mean value form's contractive property can be used to limit the dependency problem, thus computing enclosures which are expected to be tinier.

$$
[\mathbf{u_{i+1}}] \subseteq \hat{\mathbf{u}}_\mathbf{i} + \sum_{j=1}^{k-1} \delta_i^j [\mathbf{d^{(j)}}](\hat{\mathbf{u}}_\mathbf{i}) + \delta_i^k [\mathbf{d^{(k)}}]([\tilde{\mathbf{u}}_\mathbf{i}]) + \left(I + \sum_{j=1}^{k-1} \delta_i^j \frac{\partial [\mathbf{d^{(j)}}]}{\partial u} [\mathbf{u_i}] \right) ([\mathbf{u_i}] - \hat{\mathbf{u}}_\mathbf{i}) \tag{3.46}
$$

The limit set to the dependency problem by the mean value form essentially comes from the computation's first terms which now involve real values instead of interval values.

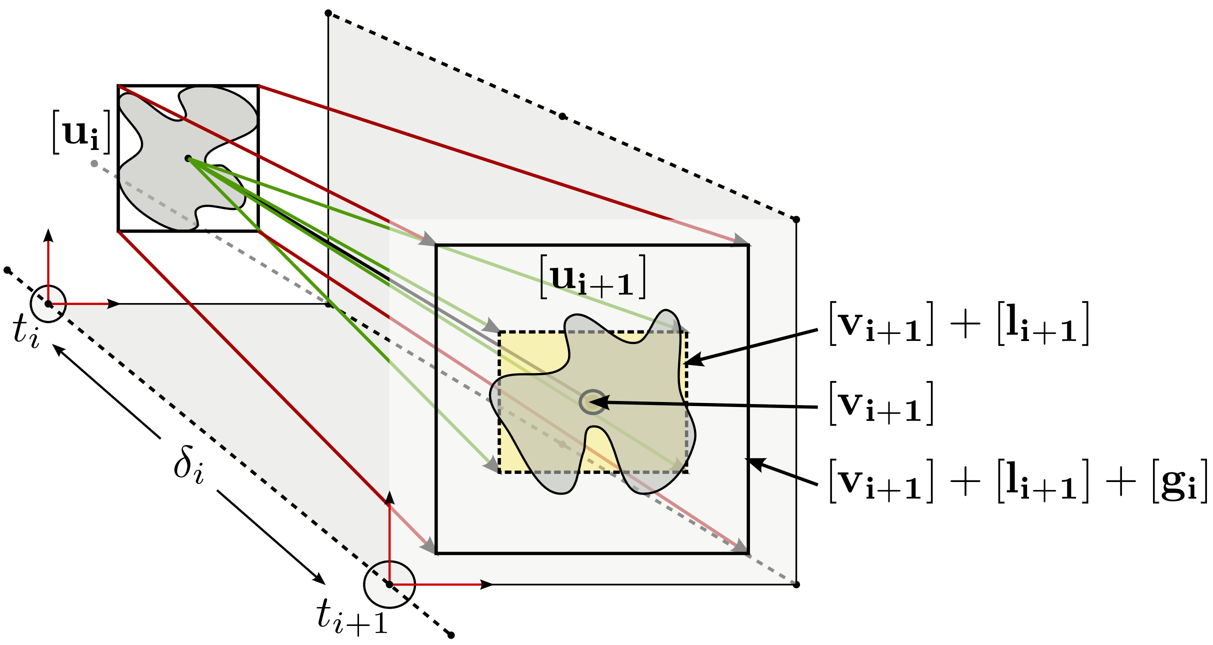

$[\mathbf{u_i}]$
$[\mathbf{u_{i+1}}]$
$[\mathbf{v_{i+1}}] + [\mathbf{l_{i+1}}]$
$[\mathbf{v_{i+1}}]$
$[\mathbf{v_{i+1}}] + [\mathbf{l_{i+1}}] + [\mathbf{g_i}]$
t_i
δ_i
t_{i+1}

Figure 3.8: The three parts of the mean value form

The expression of the mean value form can be decomposed into three parts (Fig. 3.8):

$$[\mathbf{v_{i+1}}] = \hat{\mathbf{u}}_\mathbf{i} + \sum_{j=1}^{k-1} \delta_i^j [\mathbf{d^{(j)}}](\hat{\mathbf{u}}_\mathbf{i}) \tag{3.47}$$

$$[\mathbf{l_{i+1}}] = \delta_i^k [\mathbf{d^{(k)}}]([\tilde{\mathbf{u}}_\mathbf{i}]) \tag{3.48}$$

$$[\mathbf{g_i}] = \left(I + \sum_{j=1}^{k-1} \delta_i^j \frac{\partial [\mathbf{d^{(j)}}]}{\partial u} [\mathbf{u_i}] \right) ([\mathbf{u_i}] - \hat{\mathbf{u}}_\mathbf{i}) \tag{3.49}$$

where

- $[\mathbf{v_{i+1}}]$ is an approximation point of $\mathbf{u}(t_i, \hat{\mathbf{u}}_\mathbf{i}, t_{i+1})$;
- $[\mathbf{l_{i+1}}]$ is an enclosure of the truncation error, which is the local excess added within the enclosure during the i-th integration step;
- $[\mathbf{g_i}]$ is the global excess propagated to t_{i+1}.

Thus the computation of $[\mathbf{u_{i+1}}]$ can be written

$$[\mathbf{u_{i+1}}] = [\mathbf{v_{i+1}}] + [\mathbf{l_{i+1}}] + [\mathbf{g_i}] \tag{3.50}$$

The information given by the values $[\mathbf{l_{i+1}}]$ and $[\mathbf{g_i}]$ can be used in the global algorithm to compute and to adapt the stepsize $\delta_{i+1,0}$.

In this section the Taylor models were introduced along with various forms used to limit the dependency problem that is caused by the interval arithmetic. First, the Horner form was defined as the rewriting of the expression of the Taylor polynomial. Then, the mean value form was defined as one of its arithmetic equivalent with the particularity of having contractive properties and containing real values. However, because of the global excess term $[\mathbf{g_i}]$ Eq. (3.49), an overestimation still limits the contraction. This overestimation derives from another limit named the wrapping effect. This effect is the direct consequence of using interval vectors (boxes) when enclosing multi-dimensional domains.

In the next section, the wrapping effect as well as the method used to contain it are discussed and detailed.

3.3.4 Wrapping Effect

The enclosure of multi-dimensional domains always requires a minimal overestimation when the shape to be enclosed does not match the one of the enclosure. This necessary overestimation is what makes the wrapping effect. More specifically, and as far as intervals are concerned, when multi-dimensional domains cannot be represented by boxes they are overestimated by their enclosures. It also has to be noted that the coordinate system selected to define such enclosures might be sub-optimal.

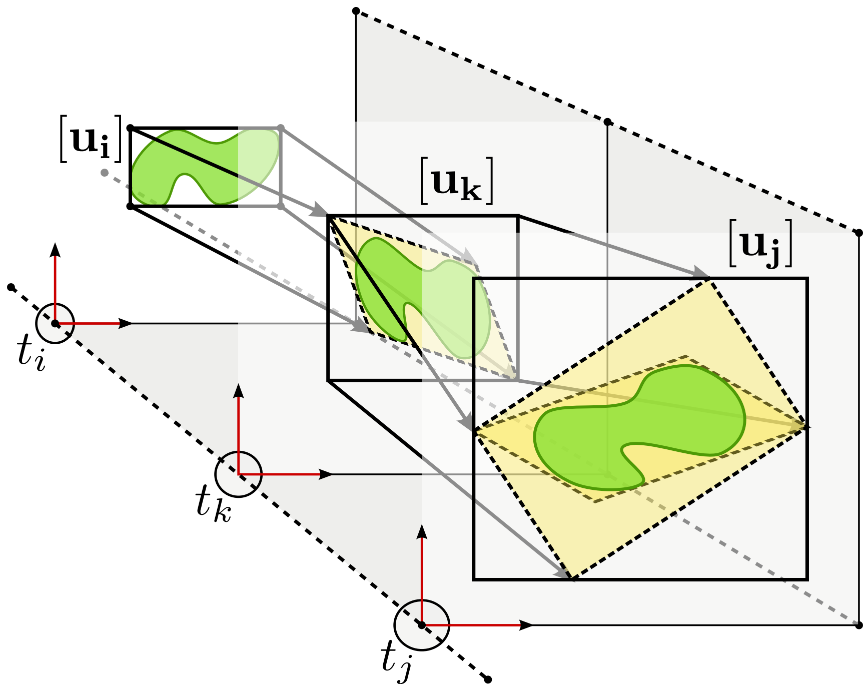


Figure 3.9: Evolution of the enclosure during the guaranteed integration

3.3.4.1 Analysis of Figure 3.9:

On this figure the feasible domain at t_i is wrapped and enclosed by the box $[\mathbf{u_i}]$ along with some irrelevant values. As a consequence, both the relevant values which are situated inside the feasible set and the irrelevant (yet enclosed) values which are situated outside of the feasible set are taken into account during the following computations. On the depicted transformation, the overestimation of the set of solutions is increased from t_i to t_k and then from t_k to t_j because of both the wrapping effect and the succession of enclosures $[\mathbf{u_k}]$ and $[\mathbf{u_j}]$. The result $[\mathbf{u_j}]$ is then enclosed by a larger box in the same coordinate system. Note that in practice the enclosures are larger than the ones on this figure because there are other sources that may lead to an overestimation such as: the error terms that enclose the error of the approximation, the dependency problem of interval arithmetic, and the roundings that assure the guarantee of computations.

In most initial value problems, the guaranteed integration algorithm is not able to enclose solutions because of the wrapping effect. Indeed, a large number of dynamic systems have solutions whose trajectories are curved. That is why, when bounding the domains along the resolution, the orientation of the boxes should be different. In [Moore, 1966], the author expresses the need to re-pack the enclosure of the solutions of the ODE in order to limit the wrapping effect.

3.3.4.2 Definition: Oriented Interval Vectors

Given that a conservative coordinate system is a source of overestimation of the enclosures in multi-dimensional spaces, interval vectors (boxes) should be defined in a moving coordinate system. Thus the enclosures can adapt their coordinate systems to minimize the irrelevant areas enclosed. Boxes are defined as follows:

$$\mathbb{IR}^n = \left\{ \hat{\mathbf{x}} + \mathcal{A}_{\mathbf{x}}[\mathbf{r}_{\mathbf{x}}] \;\middle|\; \begin{array}{l} \hat{\mathbf{x}} \in \mathbb{R}^n \\ \mathcal{A}_{\mathbf{x}} \in \mathbb{R}^{n\times n} \\ [\mathbf{r}_{\mathbf{x}}] \in \mathbb{IR}^n \end{array} \right\} \tag{3.51}$$

where $\mathcal{A}_{\mathbf{x}}$ is nonsingular.

3.3.4.3 Application to the mean value form Eq. (3.50):

The initial enclosure $[\mathbf{u_0}]$ at time-step t_0 can be written $[\mathbf{u_0}] = \hat{\mathbf{u}}_\mathbf{0} + \mathcal{A}_\mathbf{0}[\mathbf{r_0}]$, with:

$$\hat{\mathbf{u}}_\mathbf{0} = midpoint([\mathbf{u_0}]), \quad \mathcal{A}_\mathbf{0} = \mathcal{I} \quad \text{and} \quad [\mathbf{r_0}] = [\mathbf{u_0}] - \hat{\mathbf{u}}_\mathbf{0} \tag{3.52}$$

Let $[\mathbf{u_i}]$ an enclosure of the solution at t_i such that $[\mathbf{u_i}] = \hat{\mathbf{u}}_\mathbf{i} + \mathcal{A}_\mathbf{i}[\mathbf{r_i}]$, the contraction at time-step t_{i+1} is computed as follows

$$[\mathbf{u_{i+1}}] = [\mathbf{v_{i+1}}] + [\mathbf{l_{i+1}}] + \left(I + \sum_{j=1}^{k-1} \delta_i^j \frac{\partial[\mathbf{d^{(j)}}]}{\partial u}[\mathbf{u_i}] \right) \mathcal{A}_i[\mathbf{r_i}] \tag{3.53}$$

Ultimately, the expression $[\mathbf{u_{i+1}}]$ is converted into the new form $[\mathbf{u_{i+1}}] = \hat{\mathbf{u}}_\mathbf{i+1} + \mathcal{A}_\mathbf{i+1}[\mathbf{r_{i+1}}]$ in order to prepare the next iteration, where

$$\hat{\mathbf{u}}_\mathbf{i+1} = midpoint([\mathbf{v_{i+1}}] + [\mathbf{l_{i+1}}]) \tag{3.54}$$

$$[\mathbf{r_{i+1}}] = \mathcal{A}_\mathbf{i+1}^{-1} \left(I + \sum_{j=1}^{k-1} \delta_i^j \frac{\partial[\mathbf{d^{(j)}}]}{\partial u}[\mathbf{u_i}] \right) \mathcal{A}_i[\mathbf{r_i}] + \mathcal{A}_\mathbf{i+1}^{-1} \left([\mathbf{l_{i+1}}] - midpoint([\mathbf{l_{i+1}}])\right) \tag{3.55}$$

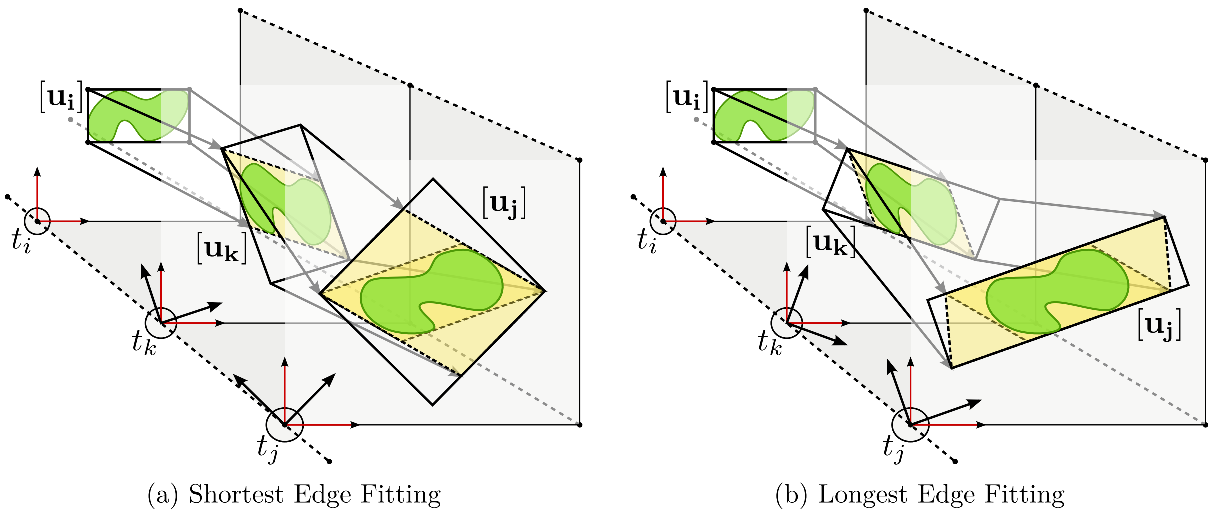
The computation of $\mathcal{A}_\mathbf{i+1}$ is very important and will determine the orientation of the enclosure. In particular, the parallelepiped method [Eijgenraam, 1981] can be defined with

$$\mathcal{A}_\mathbf{i+1} = midpoint \left(\left(I + \sum_{j=1}^{k-1} \delta_i^j \frac{\partial[\mathbf{d^{[j]}}]}{\partial u}[\mathbf{u_i}] \right) \mathcal{A}_\mathbf{i} \right) \tag{3.56}$$

but the enclosure computed using this method is unstable and may often provide bounds that are too large after a few time-steps. A more relevant choice for the computation of $\mathcal{A}_\mathbf{i+1}$ is the Q-factor from the QR-Factorization [Lohner, 1988]:

$$\mathcal{Q}_\mathbf{i+1}\mathcal{R}_\mathbf{i+1} = midpoint \left(\left(I + \sum_{j=1}^{k-1} \delta_i^j \frac{\partial[\mathbf{d^{[j]}}]}{\partial u}[\mathbf{u_i}] \right) \mathcal{A}_\mathbf{i} \right) \tag{3.57}$$

This computation of $\mathcal{A}_{\mathbf{i+1}}$ is more stable [Lohner, 1987] than the parallelepiped method Eq. (3.56) because it always fits the longest edge of the domain to bound (Fig. 3.10b).



(a) Shortest Edge Fitting (b) Longest Edge Fitting

Figure 3.10: Comparison of the shortest and longest edges fitting

In this section the wrapping effect has been discussed as well as the overestimation it leads to, which needs to be limited. Afterwards, a method named QR-Factorization was introduced.

In the next sections additional processes are presented that can be used to reach a better contraction. However these methods require a greater number of computations and are expensive.

3.4 Consistency and Piecewise Evaluation

In this section, two methods are introduced to compute tighter enclosures and to reduce the overestimation. The first method is the piecewise evaluation (Section 3.4.1). It improves the accuracy of the *LocalEnclosure* methods to compute a tighter enclosure $[\mathbf{u_{i+1}}]$ at t_{i+1} from the enclosure $[\mathbf{u_i}]$ at t_i and the *a priori* enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ on $[t_i; t_{i+1}]$.

The second method is the backward consistency pruning (Section 3.4.2). It has the ability to erase some parts of the enclosure $[\mathbf{u_{i+1}}]$ (previously computed) thanks to a consistency property on the enclosure at t_i and t_{i+1}.

These techniques are based on the computation made on tighter enclosures that limits the dependency problem and uses the convergence properties of the inclusion functions (Section 2.2.4). However, these methods could require a greater number of computations.

3.4.1 Piecewise Evaluation

The piecewise evaluation is a technique that is used to reduce the overestimation of the operation's results [Deville et al., 2000]. It uses the convergence properties of inclusion functions. When enclosures become too large, the overestimation increases as well. That is why the piecewise evaluation can be used to increase the accuracy of the results. Indeed, it reduces the dependency problem by reducing the width of the boxes on which computations are done, thus increases as well the convergence of the computation's results. Nevertheless such an operation could require a greater number of computations.

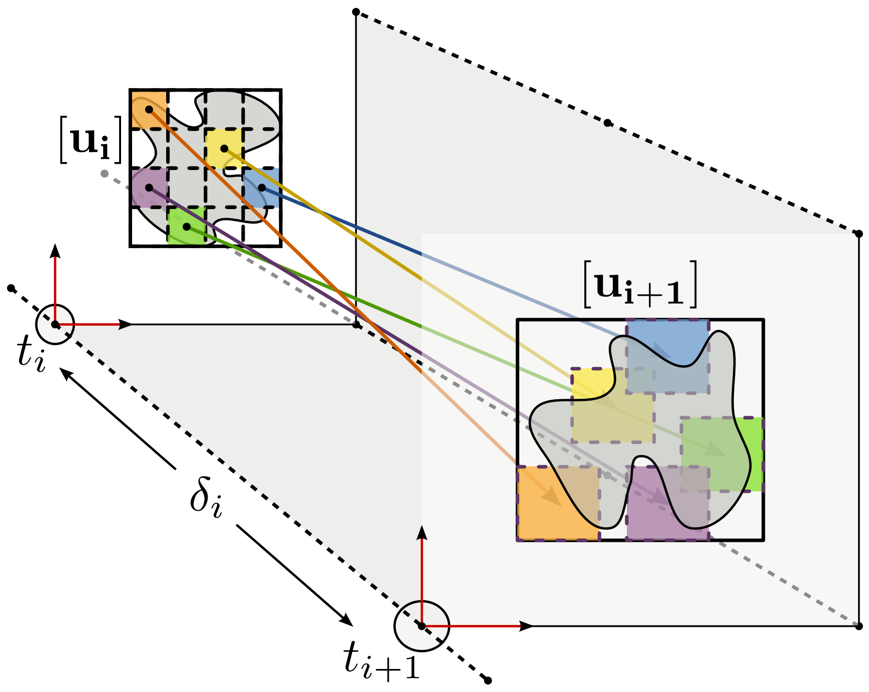


Figure 3.11: Complete piecewise evaluation on $[\mathbf{u_i}]$

3.4.1.1 Definition: Piecewise evaluation

The piecewise evaluation is defined as follows: the enclosure $[\mathbf{u_i}]$ at time-step t_i is divided into p pieces from $[\mathbf{u_i^1}]$ to $[\mathbf{u_i^p}]$, which are tighter enclosures, such that each piece $[\mathbf{u_i^k}]$ is a subset of $[\mathbf{u_i}]$ and

$$[\mathbf{u_i}] \subseteq \bigcup_{1 \leq j \leq p} [\mathbf{u_i^j}] \tag{3.58}$$

Then from each enclosure $[\mathbf{u_i^k}]$ at t_i, a guaranteed enclosure $[\mathbf{u_{i+1}^k}]$ of $[\mathbf{u}](t_i, [\mathbf{u_i^k}], t_{i+1})$ is computed thanks to a guaranteed integration method.

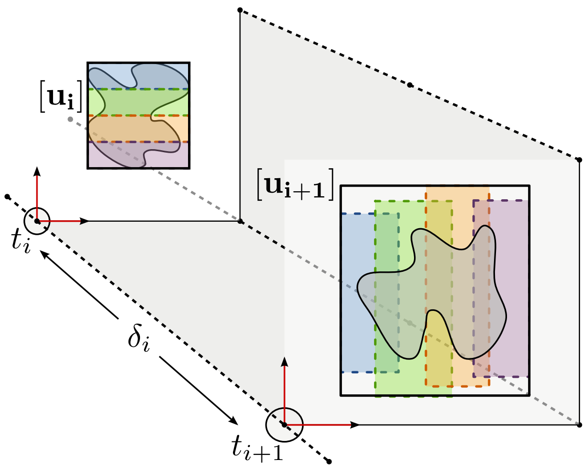
$$[\mathbf{u_{i+1}^k}] = \mathit{LocalEnclosure}(t_i, [\mathbf{u_i^k}], [\mathbf{\tilde{u}_i^k}], t_{i+1}) \tag{3.59}$$

where $[\mathbf{\tilde{u}_i^k}]$ is an *a priori* enclosure of $[\mathbf{u}](t_i, [\mathbf{u_i^k}], t)$ for all t in $[t_i; t_{i+1}]$. Note that $[\mathbf{\tilde{u}_i^k}]$ is a subset of $[\mathbf{\tilde{u}_i}]$, which can be used to avoid the series of computations done by the *GlobalEnclosure* method. Ultimately, the final result is the union of all the enclosures $[\mathbf{u_{i+1}^k}]$

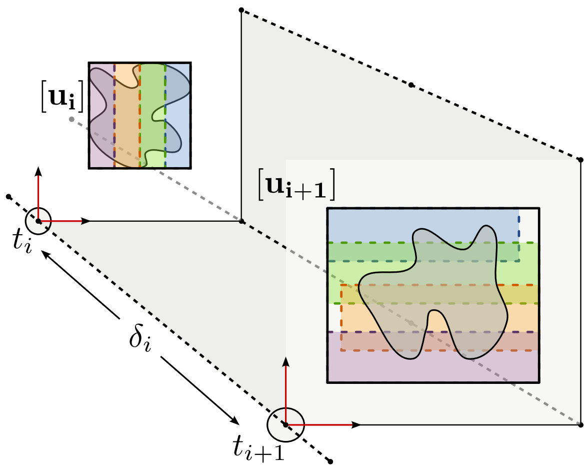
$$[\mathbf{u_{i+1}}] = \bigcup_{1 \leq j \leq p} [\mathbf{u_{i+1}^j}] \tag{3.60}$$

3.4.1.2 Analysis of Figure 3.11:

On this figure, the guaranteed enclosure $[\mathbf{u_i}]$ at t_i is divided into 16 boxes from $[\mathbf{u_i^1}]$ to $[\mathbf{u_i^{16}}]$. The guaranteed integration is consequently performed from t_i to t_{i+1} upon the 16 previously mentioned enclosures. Here, only five integrations are represented. Ultimately, all these enclosures are gathered into a single enclosure $[\mathbf{u_{i+1}}]$.



(a) Piecewise evaluation of $[\mathbf{u_{i+1}^1}]$



(b) Piecewise evaluation of $[\mathbf{u_{i+1}^2}]$

Figure 3.12: Piecewise evaluation

3.4.1.3 Analysis of Figure 3.12:

On these figures, the guaranteed enclosure $[\mathbf{u_i}]$ at t_i is independently divided into four pieces on each of the two dimensions $[\mathbf{u_i^{1,1}}] \ldots [\mathbf{u_i^{1,4}}]$ and $[\mathbf{u_i^{2,1}}] \ldots [\mathbf{u_i^{2,4}}]$. Both of the enclosure $[\mathbf{u_{i+1}^1}]$ (computed from $[\mathbf{u_i^{1,1}}] \ldots [\mathbf{u_i^{1,4}}]$) and $[\mathbf{u_{i+1}^2}]$ (computed from $[\mathbf{u_i^{2,1}}] \ldots [\mathbf{u_i^{2,4}}]$)

are guaranteed to enclose the solution at t_{i+1}. Thus their intersection is guaranteed to enclose the solution as well. Consequently

$$[\mathbf{u_{i+1}}] = \bigcap_{1\leq k\leq n} [\mathbf{u^k_{i+1}}] \tag{3.61}$$

where $n = 2$ is the dimension of the tube and

$$[\mathbf{u^k_{i+1}}] = \bigcup_{1\leq j\leq p} [\mathbf{u^{k,j}_{i+1}}] \tag{3.62}$$

with $p = 4$ is the number of pieces considered on each dimension.

3.4.2 Consistency

The consistency is a property concerning the computed enclosures, it was introduced by [Deville et al., 2000]. A pruning method can be defined that reduces the overestimation of local enclosures when the consistency property is combined with the guarantee property provided by the previously introduced *LocalEnclosure* method in both forward and backward directions of the integration.

3.4.2.1 Definition: Consistency

Let $[\mathbf{u_i}]$ and $[\mathbf{u_j}]$ two boxes which are guaranteed to enclose the solution of the ODE system Θ at time-steps t_i and t_j. These boxes are consistent if at least one solution $\mathbf{u}^*$ exists, such that $\mathbf{u}^*(t_i) \in [\mathbf{u_i}]$ and $\mathbf{u}^*(t_j) \in [\mathbf{u_j}]$. In other words:

$$[\mathbf{u}](t_i, [\mathbf{u_i}], t_j) \cap [\mathbf{u_j}] \neq \emptyset \tag{3.63}$$

$$[\mathbf{u}](t_j, [\mathbf{u_j}], t_i) \cap [\mathbf{u_i}] \neq \emptyset \tag{3.64}$$

Otherwise, these enclosures are inconsistent, thus proving there are no solutions to such systems.

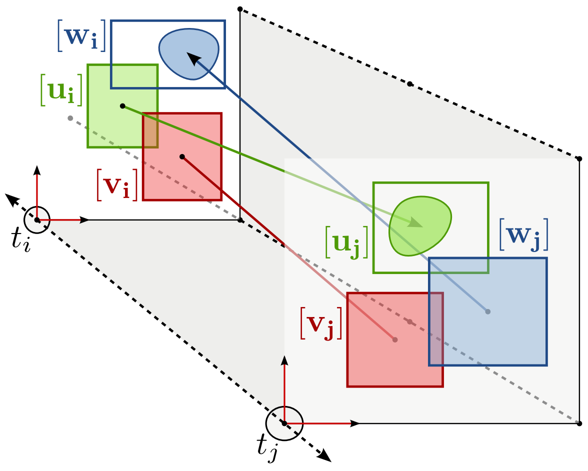
3.4.2.2 Analysis of Figure 3.13a:

On this figure, the guaranteed enclosures $[\mathbf{u}]$, $[\mathbf{v}]$ and $[\mathbf{w}]$ are represented along with the two time-steps t_i and t_j.

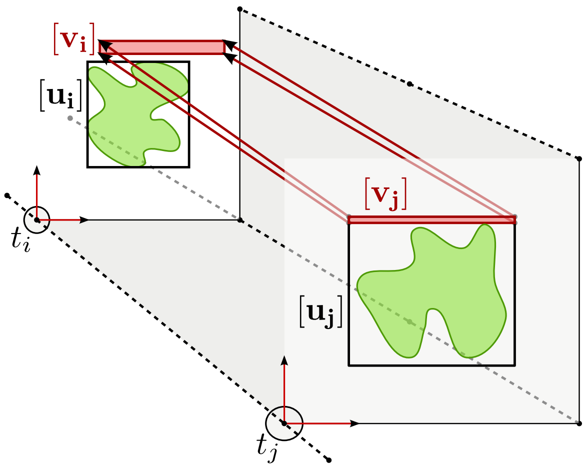
Here the enclosures $[\mathbf{u}]$ and $[\mathbf{v}]$ are inconsistent because at t_j the enclosure $[\mathbf{u_j}]$ does not intersect $[\mathbf{v_j}]$. Similarly, the enclosures $[\mathbf{v}]$ and $[\mathbf{w}]$ are inconsistent because of the empty intersection at t_i between $[\mathbf{v_i}]$ and $[\mathbf{w_i}]$.

On the other side the enclosures $[\mathbf{u}]$ and $[\mathbf{w}]$ are consistent. However this does not assure that the enclosure $[\mathbf{w_j}]$ contains a solution in $[\mathbf{u}](t_i, [\mathbf{u_i}], t_j)$ — and such is also the case with the enclosure $[\mathbf{u_i}]$ and the set of solutions $[\mathbf{w}](t_j, [\mathbf{w_j}], t_i)$.

The consistency property can be used to reduce the local enclosure $[\mathbf{u_{i+1}}]$ that has been previously computed. Indeed, a part $[\mathbf{v_{i+1}}]$ of the guaranteed enclosure $[\mathbf{u_{i+1}}]$ at t_{i+1} can be removed from $[\mathbf{u_{i+1}}]$ when it is inconsistent with a guaranteed enclosure $[\mathbf{u_i}]$ at time-step t_i. The proof of the inconsistency can be computed through the use of the previously described guaranteed integration methods because they would provide a guaranteed enclosure of $[\mathbf{v}](t_j, [\mathbf{v_j}], t_i)$.



(a) Inconsistency of $[\mathbf{v}]$ with $[\mathbf{u}]$ and $[\mathbf{v}]$



(b) Erosion of $[\mathbf{u_j}]$

Figure 3.13: Consistency and inconsistency

3.4.2.3 Analysis of Figure 3.13b:

On this figure two time-steps t_i and t_j are represented along with their respective guaranteed enclosures $[\mathbf{u_i}]$ and $[\mathbf{u_j}]$. An enclosure $[\mathbf{v_j}]$ (which is not guaranteed to contain the solution) is defined at t_j from $[\mathbf{u_j}]$. Here, the box $[\mathbf{v_j}]$ is inconsistent with the enclosure $[\mathbf{u_i}]$ at t_i because the guaranteed enclosure of $[\mathbf{u}](t_j, [\mathbf{v_j}], t_i)$ denoted by $[\mathbf{v_i}]$ does not intersect $[\mathbf{u_i}]$. Consequently the enclosure $[\mathbf{v_j}]$ can be removed from $[\mathbf{u_j}]$.

3.5 Conclusion

In this chapter the guaranteed integration of ordinary differential equations has been presented. The global integration algorithm (Section 3.3) is a loop made of two methods: one computes a global enclosure (Section 3.3.2) of the solution over a continuous set of time-steps while the other one computes its contraction into a local enclosure (Section 3.3.3).

The first method is the combination of two algorithms based on two theorems: the Cauchy-Lipschitz's existence and uniqueness theorem and the Banach's fixed point theorem. A first algorithm called FOE was introduced along with its limitations. That is why a second algorithm, named HOE was presented that allows larger stepsizes during the integration process.

The second method uses the Taylor series, on which several improvements have been made such as the use of the Horner form and the Mean Value form [Caprani and Madsen, 1980]. The Taylor models are widely used to solve these integration problems because they are efficient at bounding the error term. However, other methods can be more adapted to enclose the solutions of IVPs. For instance, in [Nedialkov and Jackson, 1999] the Interval Hermite-Obreschkoff (IHO) method is discussed. The authors compare the stability between the IHO method and the previously introduced method based on the Taylor models. It appears the IHO method is more stable than the Taylor models, the overestimation is therefore reduced. Besides, it also has some interesting properties to consider on the reduction of the wrapping effect [Jackson and Nedialkov, 2002]. Nevertheless, the methods based on Taylor models must not be neglected given that positive results have recently been presented [Makino and Berz, 2011] after such models were used to limit the wrapping effect.

It has to be highlighted that, theoretically, the concatenation of all the Taylor models occurring during the integration method defines a tube which is closer to the exact solution than the set of global enclosures $[\tilde{\mathbf{u}}_\mathbf{i}]$. Nevertheless, because of the large amount of memory it would take and the complexity of computation using such data, the Taylor series are not stored in memory.

In this work, the package *Vnode-lp* [Nedialkov et al., 2001] that is a reference in the domain of the guaranteed integration based on the interval arithmetic is used to perform the guaranteed integration of ODEs. This tool has been developed in order to provide tiny local enclosures and fast computation. It implements all the improvements that have been introduced in this chapter.

4 Guaranteed Constraint Programming

In the future, every industry should be an environmental industry. In a world where energy and carbon emissions are constrained, every business must take resource productivity seriously.

– *David Miliband*

Contents

4.1 Introduction

The *Constraint Satisfaction Problems* or CSPs are defined by sets of constraints. Therefore, solutions to a CSP have to be found while taking those into account.

The CSPs are said to be combinatorial because of the numerous combinations which have to be considered before a solution respecting each constraint is found. The computation power does not suffice (in itself) to consider all possible combinations in a reasonable time; that is why it is necessary to introduce some processes that could lead the inquiry towards the most relevant solutions. These algorithms are called CSP solvers.

A non-exhaustive list gathering examples of simple problems that can be modeled as a constraint satisfaction problem would include: eight queens puzzle, map coloring problem, sudoku and many other logic puzzles.

In this chapter, basics of constraint programming are presented with some additive methods. The Constraint Satisfaction Problems and their solutions are introduced in Section 4.2 with a few examples. An extension with guaranteed arithmetics and intervals is detailed in Section 4.3 as well as a specific logic. Section 4.4 presents some specific methods to solve CSP with guaranteed methods called contractors which are combined and used to reduce the domain of the variables. These methods are improved in Section 4.5 through a Set Inversion method based on the interval analysis.

4.2 Constraint Satisfaction Problems (CSPs)

4.2.1 Definition

A *Constraint Satisfaction Problem* (CSP) [Tsang, 2014] is a mathematical problem which involves the search for one or all assignments on a set of variables $\mathcal{V}$ with a set of values $\mathcal{X}$ driven by a set of constraints $\mathcal{C}$. Then a CSP is defined from a *Constraint Network* that is made of triple $\langle \mathcal{V}, \mathcal{X}, \mathcal{C} \rangle$ where

- $\mathcal{V} = \{x_1, \ldots, x_n\}$ is the set of the n variables of the problem.
- $\mathcal{X} = \{d_1, \ldots, d_n\}$ is a set of the respective domains of the n variables, in other words:
$$\forall i \in \{1 \ldots n\},\ x_i \in d_i \tag{4.1}$$
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of m constraints.

Each variable x_i in $\mathcal{V}$ can be assigned to any value in its respective domain d_i in $\mathcal{X}$. There are several types of CSPs depending on the domain $\mathcal{X}$. Indeed CSP can be defined over integer domains (n-queens, sudoku), real domains (spreadsheets), binary domains (digital circuits) and even non-numeric domains in case of symbolic CSP (crossword puzzles).

4.2.1.1 Definition: Constraints

Every constraint $c_j \in \mathcal{C}$ is defined by a couple $(\mathcal{V}_c, \rho_c)$ where

- $\mathcal{V}_c = \{x_1^c, \ldots, x_k^c\} \subseteq \mathcal{V}$ is a subset of k variables.
- ρ_c is an k-ary relation defining the set of all assignments allowed for the set of variables $x_i \in \mathcal{V}_c$.

The set of variables involved in the constraint c is denoted by $var(c) = \mathcal{V}_c$. Variables which are involved together in a constraint are called linked variables. Possibilities are reduced by the relation of each constraint $c_j \in \mathcal{C}$ dealing with the subsets of variables.

4.2.1.2 Definition: Evaluation

An evaluation $\tilde{s}$ is a function from a subset of variables $x_i \in \mathcal{V}_{\tilde{s}} \subset \mathcal{V}$ to a particular set of values v_i in the matching subset $\mathcal{X}_{\tilde{s}} \subset \mathcal{X}$.

$$\tilde{s} : \begin{aligned} \mathcal{V}_{\tilde{s}} &\to \mathcal{X}_{\tilde{s}} \\ x_i &\mapsto v_i \end{aligned} \tag{4.2}$$

An evaluation is consistent when all constraints are satisfied by the evaluation. A constraint $c = (\mathcal{V}_c, \rho_c)$ is satisfied when $\mathcal{V}_c = (x_1^c, \ldots, x_k^c)$ and $(\tilde{s}(x_1^c), \ldots, \tilde{s}(x_k^c)) \in \rho_c$.

4.2.2 Resolution

Solving a CSP consists in finding an assignment throughout $\mathcal{X}$ for all the variables from $\mathcal{V}$ such that all the constraints in $\mathcal{C}$ are satisfied. The solution can be found using filtering algorithms (Contractors) that would remove inconsistent values from the domains d_i in $\mathcal{X}$.

4.2.2.1 Definition: Solution of a CSP

A CSP is solved when each variable $x_i \in \mathcal{V}$ is assigned to a value in its domain $d_i \in \mathcal{X}$ while respecting the set of constraints $\mathcal{C}$.

4.2.3 Example

Usually, the constraints are algebraic, and the variables are either integer or real.

$$x^2 \leq y + z \tag{4.3}$$

where x, y and z are reals, evaluated in their respective domains $x \in [-5; 5]$, $y \in [-5; 5]$ and $z \in [2; 4]$. The constraints programming allows the expression of many peculiar constraints on reals or integers such as

$$AllDifferent(x, y, z) \tag{4.4}$$

which force the variables to being assigned with non-equal values over finite domains.

In the problems that are considered, new types of variables should however be managed: the functional variables (defined by tubes) that represent unary functions. Consequently, new constraints can be written based on functional properties. For instance the constraint

$$Increase(u, x_1, x_2) \tag{4.5}$$

where u is a functional variable, x_1 and x_2 two reals tends to remove from the tube $[u]$ associated to u the functions which are not increasing between x_1 and x_2. Also, the guaranteed integration scheme previously described can be used inside the following constraint

$$Differential(\mathbf{u}, \mathbf{d}(\mathbf{x}, \mathbf{u}), t_0, t_f) \tag{4.6}$$

to contract the tube $[\mathbf{u}]$ from t_0 to t_f using the differential $\mathbf{u}'(t) = \mathbf{d}(\mathbf{x}, \mathbf{u}(t))$.

Note that in this thesis, only continuous CSP are considered. However the tool that has been implemented can be extended in order to take into account discrete CSP and it could be an improvement to deal with mixed CSP models. It would requires the implementation of a specific module that should take the advantages of existing tools such as Choco [Jussien et al., 2008]. Moreover, the filtering techniques used on continuous CSP such that 3B-coherence [Lhomme, 1993] have equivalents on finite CSP such that the singleton arc-consistency [Debruyne and Bessiere, 1997].

4.3 Guaranteed solutions of CSPs with intervals

In this section an extension of the CSP is introduced that aims to certify the solution feasibility. In this process, the guaranteed arithmetics are used to compute the operations safely.

4.3.1 Definition: Interval CSP

The *interval Constraint Satisfaction Problem* is an extension of the CSP model. In this interval extension, the domains d_i in $\mathcal{X}$ are intervals $[x_i]$ or tubes $[u_i]$ according to the set of variables considered. Then interval CSP is defined as quintuple $\langle \mathcal{V}_\mathcal{X}, \mathcal{V}_\mathcal{U}, \mathcal{X}, \mathcal{U}, \mathcal{C}\rangle$ where

- $\mathcal{V}_\mathcal{X} = \{x_1, \dots, x_p\}$ is the set of the p real variables of the problem.
- $\mathcal{V}_\mathcal{U} = \{u_1, \dots, u_n\}$ is the set of the n functional variables of the problem.
- $\mathcal{X} = \{[x_1], \dots, [x_p]\}$ is the set of the respective domains of the p real variables.
- $\mathcal{U} = \{[u_1], \dots, [u_n]\}$ is the set of the respective domains of the n functional variables.
- $\mathcal{C} = \{c_1, \dots, c_m\}$ is a set of m constraints.

such that every constraint $c_j \in \mathcal{C}$ is defined by a triple $(\mathcal{V}_{\mathcal{X}c}, \mathcal{V}_{\mathcal{U}c}, \rho_c)$ where

- $\mathcal{V}_{\mathcal{X}c} = \{x^c_1, \dots, x^c_{k_1}\} \subseteq \mathcal{V}_\mathcal{X}$ is a subset of k_1 real variables.
- $\mathcal{V}_{\mathcal{U}c} = \{u^c_1, \dots, u^c_{k_2}\} \subseteq \mathcal{V}_\mathcal{U}$ is a subset of k_2 functional variables.
- ρ_c is a k-ary relation defining the set of all assignments allowed for the set of variables $x_i \in \mathcal{V}_\mathcal{X}$ and $u_i \in \mathcal{V}_\mathcal{U}$.

The use of the interval arithmetic implies some limitations: it could make it impossible to prove whether a complete assignment function is a feasible solution or not, especially when such a function is close to the frontier of the set of solutions.

Indeed, when using the interval arithmetic in the following algebraic constraint c in $\mathcal{C}$ defined such that

$$\mathbf{h}(\mathbf{x}) \leq \mathbf{0} \tag{4.7}$$

where $\mathbf{h}$ is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$, the result of the evaluation can be ambiguous. Let $s^\triangleright$ a complete assignment function for the set of variables $\mathcal{V}_\mathcal{X}$. An interval $[\mathbf{z}] = [\mathbf{h}_\downarrow(s^\triangleright(\mathbf{x})); \mathbf{h}^\uparrow(s^\triangleright(\mathbf{x}))]$ is then computed through the use of the interval arithmetic to guarantee the operations' results.

$$\mathbf{h}(s^\triangleright(\mathbf{x})) \in [\mathbf{z}] = [\mathbf{h}_\downarrow(s^\triangleright(\mathbf{x})); \mathbf{h}^\uparrow(s^\triangleright(\mathbf{x}))] = \left\{\mathbf{z} \in \mathbb{R}^m \;\middle|\; \mathbf{h}_\downarrow(s^\triangleright(\mathbf{x})) \leq \mathbf{z} \leq \mathbf{h}^\uparrow(s^\triangleright(\mathbf{x}))\right\} \tag{4.8}$$

Depending on the values of the bounds of $[\mathbf{z}]$, three cases have to be distinguished:

- When $\mathbf{h}^{\uparrow}(s^{\triangleright}(\mathbf{x})) \leq \mathbf{0}$ then the result $\mathbf{h}(s^{\triangleright}(\mathbf{x}))$ which is guaranteed to be lower than or equal to $\mathbf{h}^{\uparrow}(s^{\triangleright}(\mathbf{x}))$ is also guaranteed to be lower than or equal to $\mathbf{0}$. In this context the assignment function defined by $s^{\triangleright}$ is guaranteed to be in the feasible domain of the constraint c.

- When $\mathbf{h}^{\downarrow}(s^{\triangleright}(\mathbf{x})) > \mathbf{0}$ then the result $\mathbf{h}(s^{\triangleright}(\mathbf{x}))$ is guaranteed to be strictly higher than $\mathbf{0}$. Thus the assignment function defined through $s^{\triangleright}$ is guaranteed to be unfeasible within the constraint c.

- When $\mathbf{h}^{\downarrow}(s^{\triangleright}(\mathbf{x})) \leq \mathbf{0} < \mathbf{h}^{\uparrow}(s^{\triangleright}(\mathbf{x}))$ then the complete assignment function $s^{\triangleright}$ cannot be certified as feasible or unfeasible because the frontier defined by $\mathbf{0}$ is contained within the interval $[\mathbf{z}]$ along with the evaluation of $\mathbf{h}(s^{\triangleright}(\mathbf{x}))$. This ambiguity cannot be removed without any further information.

In response to the third scenario, the Boolean set that is used to express the validity of the constraints has to be extended with a new value that would define indeterminate values.

4.3.1.1 Interval and tube assignment functions:

The strength of this approach is in the ability to compute the feasibility of a constraint (or a set of constraints) on a set of complete assignment functions. An interval assignment function $s_{\mathcal{X}}^{[\triangleright]}$ can be used to define a set of assignment functions from the set of variables $x_i \in \mathcal{V}_{\mathcal{X}}$ to specific intervals $[y_i]$ ($[y_i]$ being a subset of $[x_i] \in \mathcal{X}$). Similarly, a tube assignment function $s_{\mathcal{U}}^{[\triangleright]}$ is used to define a set of assignment functions from the set of variables $u_i \in \mathcal{V}_{\mathcal{U}}$ to a specific tube $[v_i]$ ($[v_i]$ being a subset of $[u_i] \in \mathcal{U}$).

$$s_{\mathcal{X}}^{[\triangleright]} : \begin{aligned} \mathcal{V}_{\mathcal{X}} &\to \mathcal{X} \\ x_i &\mapsto [y_i] \end{aligned} \quad \text{and} \quad s_{\mathcal{U}}^{[\triangleright]} : \begin{aligned} \mathcal{V}_{\mathcal{U}} &\to \mathcal{U} \\ u_i &\mapsto [v_i] \end{aligned} \tag{4.9}$$

When a variable x_i moves within an interval $[y_i]$ which represents several values, a set of possible assignments for x_i is defined. Thus an infinite number of evaluations can easily be characterized. The assignment function $s^{[\triangleright]}$ defines a box $[\mathbf{y}]$ that matches the dimension of $\mathbf{x}$

$$s^{[\triangleright]}(\mathbf{x}) = \begin{bmatrix} s^{[\triangleright]}(x_1) \\ \vdots \\ s^{[\triangleright]}(x_n) \end{bmatrix} = \begin{bmatrix} [y_1] \\ \vdots \\ [y_n] \end{bmatrix} = [\mathbf{y}] \tag{4.10}$$

and, reciprocally, a box $[\mathbf{x}]$ that matches the dimension of $\mathbf{x}$ can be used to define an interval assignment function $s^{[\triangleright]}$.

4.3.2 Three-valued logic

A set of situations has been previously exposed that favors the extension of the classic Boolean arithmetic with a third value [Mukaidono, 1986], thus modeling indeterminate values between $\top$ (*True*) and $\bot$ (*False*). This new value is noted Ξ, and the set of three logic values $\mathbb{IB}$ is defined as below:

$$\mathbb{IB} = \{\top, \bot, \Xi\} \tag{4.11}$$

It has to be noted that when dealing with this third value Ξ, the Boolean arithmetic must be extended as well, which explains why the classic logic operations $\wedge$, $\vee$, $\neg$ are overloaded. The results of the operations that include this new value Ξ are defined in table 4.1.

Operator $\wedge$	Operator $\vee$	Operation $\neg$	Operation $\Rightarrow$
$\top \wedge \Xi = \Xi$	$\top \vee \Xi = \top$		$\top \Rightarrow \Xi = \Xi$
$\Xi \wedge \top = \Xi$	$\Xi \vee \top = \top$		$\Xi \Rightarrow \top = \top$
$\Xi \wedge \Xi = \Xi$	$\Xi \vee \Xi = \Xi$	$\neg\Xi = \Xi$	$\Xi \Rightarrow \Xi = \Xi$
$\Xi \wedge \bot = \bot$	$\Xi \vee \bot = \Xi$		$\Xi \Rightarrow \bot = \Xi$
$\bot \wedge \Xi = \bot$	$\bot \vee \Xi = \Xi$		$\bot \Rightarrow \Xi = \top$

Table 4.1: Truth tables and results of classic logic operations with the indeterminate value Ξ

When using this extended set of logic values, the constraint can be certified as valid or invalid according to an interval assignment function $s^{[\triangleright]}$. Let $c \in \mathcal{C}$ a constraint which can be evaluated via the interval assignment function $s^{[\triangleright]}$ such that

$$c_j(s^{[\triangleright]}(\mathbf{x})) = c_j(s^{[\triangleright]}(x_1), \dots, s^{[\triangleright]}(x_n)) = c_j([y_1], \dots, [y_n]) \in \mathbb{IB} \tag{4.12}$$

When the constraint c is valid for all complete assignment functions $s^{\triangleright}$ in $s^{[\triangleright]}$, the constraint c is then valid through the interval assignment function $s^{[\triangleright]}$.

4.3.2.1 Validity of a constraint through an interval assignment function:

When the constraint c is certified as feasible through the interval assignment function $s^{[\triangleright]}$, it means that all the assignment functions $s^{\triangleright}$ it contains are valid for this constraint c.

$$c(s^{[\triangleright]}(\mathbf{x})) = \top \quad \Rightarrow \quad \forall s^{\triangleright} \in s^{[\triangleright]},\ c(s^{\triangleright}(\mathbf{x})) = \top \tag{4.13}$$

On the other hand, when the constraint c is certified as unfeasible by the interval assignment function $s^{[\triangleright]}$, then no complete assignments $s^{\triangleright}$ are feasible within $s^{[\triangleright]}$.

$$c(s^{[\triangleright]}(\mathbf{x})) = \bot \quad \Rightarrow \quad \nexists s^{\triangleright} \in s^{[\triangleright]},\ c(s^{\triangleright}(\mathbf{x})) = \top \tag{4.14}$$

In the case where $c(s^{[\triangleright]}(\mathbf{x})) = \Xi$, then the constraint does not provide any information about the complete assignment functions $s^{\triangleright}$ which are contained in $s^{[\triangleright]}$.

4.3.2.2 Definition: Characterization of an interval assignment function within a guaranteed CSP

Let $[\mathbf{x}]$ a box that describes a set of possibilities for the variables $\mathbf{x}$. The box $[\mathbf{x}]$ can be associated to an interval assignment function $s^{[\triangleright]}$ such that

$$s^{[\triangleright]} : x_i \mapsto [\mathbf{x}]_i \tag{4.15}$$

The feasibility of the interval assignment function $s^{[\triangleright]}$ is given by the conjunction of the feasibility of each constraint.

$$feasible(s^{[\triangleright]}([\mathbf{x}])) = \bigwedge_{c \in \mathcal{C}} c(s^{[\triangleright]}(\mathbf{x})) \tag{4.16}$$

When all the constraints $c \in \mathcal{C}$ are evaluated as $\top$ through the interval assignment function $s^{[\triangleright]}$, any complete assignment $s^{\triangleright}$ within $s^{[\triangleright]}$ is a guaranteed solution of the CSP.

$$s^{\triangleright} : x_i \mapsto v_i \qquad \text{with } v_i \in s^{[\triangleright]}(x_i) \tag{4.17}$$

On the contrary, when at least one constraint $c \in \mathcal{C}$ is evaluated as $\perp$ through the interval assignment function $s^{[\triangleright]}$, it is then guaranteed that there is no complete assignment $s^{\triangleright}$ within $s^{[\triangleright]}$ that could be a solution of the CSP.

Otherwise, there is no constraint $c \in \mathcal{C}$ that is evaluated as $\perp$ through the interval assignment function $s^{[\triangleright]}$; but at least one indeterminate constraint does exist (that is evaluated as Ξ). Consequently, the solution existence (or nonexistence) cannot be guaranteed.

4.4 Contractors

Derived from constraint programming and filtering algorithms, contractor programming [Chabert and Jaulin, 2009a] aims at reducing the domain of boxes (Interval vectors, $\mathbb{IR}^n$) and tubes ($\mathbb{IF}^n$) in order to filter continuous domains [Sam-Haroud and Faltings, 1996]. Two types of contractors can be considered. Atomic contractors and meta-contractors that result from operations on contractors. As much atomic contractors as there are constraints can be defined.

4.4.1 Definition: Contractor [Chabert and Jaulin, 2009a]

Let $\mathbb{X}$ a feasible domain subset of $\mathbb{R}^n$ and $C_{\mathbb{X}}$ an operator. If for all $[\mathbf{x}]$ in $\mathbb{IR}^n$ the equations 4.18 *(Contraction)* and 4.19 *(Completeness)* are satisfied, then $C_{\mathbb{X}}$ is a contractor on $[\mathbf{x}]$ associated to the set $\mathbb{X}$.

$$C_{\mathbb{X}}([\mathbf{x}]) \subseteq [\mathbf{x}] \tag{4.18}$$
$$[\mathbf{x}] \cap \mathbb{X} \subseteq C_{\mathbb{X}}([\mathbf{x}]) \tag{4.19}$$

The first property Eq. (4.18) means that the contractor returns a subdomain of the input. The second Eq. (4.19) asserts that all of the feasible solutions from the initial domain $[\mathbf{x}]$ are preserved by the contraction and are also included in the final domain $C_{\mathbb{X}}([\mathbf{x}])$.

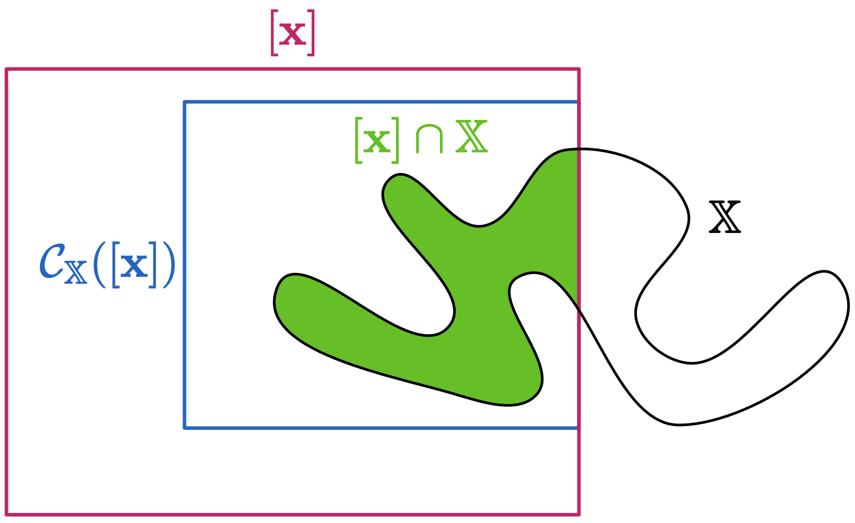

$[\mathbf{x}]$
$[\mathbf{x}] \cap \mathbb{X}$
$\mathcal{C}_{\mathbb{X}}([\mathbf{x}])$
$\mathbb{X}$

Figure 4.1: Contraction and completeness of the operator $\mathcal{C}_{\mathbb{X}}$

On Figure 4.1, the feasible domain $\mathbb{X}$ is represented by the shape drawn with black stroke and the initial box to contract $[\mathbf{x}]$ is purple. The green area is the part of the feasible domain $\mathbb{X}$ which is accessible from (inside) the box $[\mathbf{x}]$ whereas the feasible domain $\mathbb{X}$ situated outside the box is not accessible. Such a phenomenon happens when several constraints are used to reduce the domain of the box $[\mathbf{x}]$. On the figure the result of the contraction made by the contractor $\mathcal{C}_{\mathbb{X}}$ is the box bounded by the blue stroke. Note that the result $\mathcal{C}_{\mathbf{X}}([\mathbf{x}])$ is a subset of the initial enclosure $[\mathbf{x}]$ (Contraction: Eq. (4.18)) and encloses the accessible feasible domain $\mathbb{X} \cap [\mathbf{x}]$ (Completeness: Eq. (4.19)).

4.4.1.1 Definition: Tube contractors

Let $\mathbb{U}$ a feasible set of functions from $\mathbb{R}$ to $\mathbb{R}^n$ and $[\mathbf{u}]$ an n-dimensional tube in $\mathbb{IF}^n$. The operator $\mathcal{C}_{\mathbb{U}}$ is a tube contractor if

$$\forall x \in \mathbb{R},\ \ C_{\mathbb{U}}([\mathbf{u}])(x) \subseteq [\mathbf{u}](x) \tag{4.20}$$

$$[\mathbf{u}] \cap \mathbb{U} \subseteq C_{\mathbb{U}}([\mathbf{u}]) \tag{4.21}$$

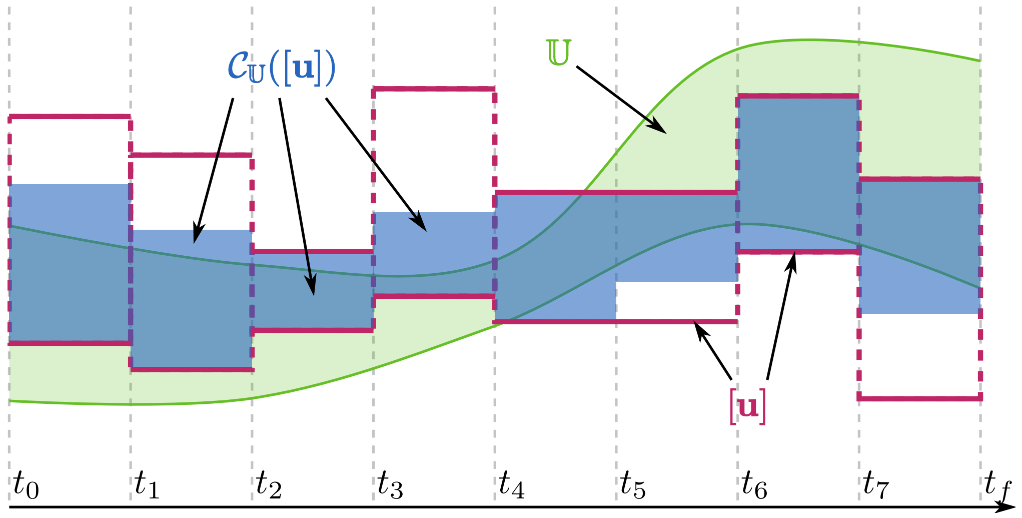


Figure 4.2: Contraction and completeness of the operator $\mathcal{C}_{\mathbb{U}}$

Figure 4.2 represents the contraction of a tube $[\mathbf{u}]$ toward the domain $\mathbb{U}$ with the contractor $\mathcal{C}_{\mathbb{U}}$. The initial tube $[\mathbf{u}]$ intersects the set of feasible functions $\mathbb{U}$ inside the contracted tube $\mathcal{C}_{\mathbb{U}}([\mathbf{u}])$.

4.4.1.2 Definition: Properties

The contractors may have additional properties.

- Minimal : $\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}_{\mathbb{X}}([\mathbf{x}]) = \mathbb{X} \cap [\mathbf{x}]$
- Monotonic : $\forall [\mathbf{x}] \subseteq [\mathbf{y}] \in \mathbb{IR}^n, \mathcal{C}_{\mathbb{X}}([\mathbf{x}]) \subseteq \mathcal{C}_{\mathbb{X}}([\mathbf{y}])$
- Idempotent : $\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}_{\mathbb{X}}(\mathcal{C}_{\mathbb{X}}([\mathbf{x}])) = \mathcal{C}_{\mathbb{X}}([\mathbf{x}])$
- Convergent : $\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}_{\mathbb{X}}([\mathbf{x}]) \underset{width([\mathbf{x}])\to 0}{\longrightarrow} \mathbb{X} \cap [\mathbf{x}]$

Note that when a contractor is minimal, it is convergent, idempotent and monotonic.

4.4.2 Atomic contractors

The atomic contractors are associated to the constraints which can be found in the CSPs. In the next paragraphs, examples of atomic contractors are introduced both on real (intervals) and functional (tubes) variables.

4.4.3 The Forward-Backward method and the algebraic constraints

The following equations are named algebraic constraints

$$(x_1 - x_2)^2 \leq x_1 x_2$$
$$h_1(\mathbf{x}) = \mathbf{0}$$
$$\mathbf{u}(t_0) \in \mathbf{g}(\mathbf{x_0})$$

These expressions can be used in order to restrain the values for x_1, x_2, $\mathbf{x}$, $\mathbf{u}$, t_0 and x_0.

The most basic contractor is the atomic contractor *HC4* which consists of a sequence of two phases [Messine, 1997] [Benhamou et al., 1999].

Let the constraint c in $\mathcal{C}$ such that $\mathbf{h}(\mathbf{x}) \leq \mathbf{0}$ and $s^{[\triangleright]}$ an interval assignment function for the variables $\mathbf{x}$.

$$s^{[\triangleright]} : \begin{cases} \mathcal{V} \to \mathbb{I}\mathcal{X} \\ x_i \mapsto [v_i] \end{cases} \quad \text{where} \quad [v_i] \subseteq d_i \in \mathcal{X} \tag{4.22}$$

The first step is the forward propagation, a process that computes rigorous bounds for the enclosure $[\mathbf{z}]$ of the evaluation of the function $\mathbf{h}$ over the box $[\mathbf{x}]$.

$$[\mathbf{z}] = [\underline{\mathbf{z}}; \overline{\mathbf{z}}] = [\mathbf{h}_{\downarrow}([\mathbf{x}]); \mathbf{h}^{\uparrow}([\mathbf{x}])] \tag{4.23}$$

Using the lower or equal constraint, it is known that $[\mathbf{z}]$ has to be lower than $\mathbf{0}$, thus implying that $\overline{\mathbf{z}}$ must be lower than $\mathbf{0}$. This is the second step.

$$[\mathbf{z}] \leq \mathbf{0} \Leftrightarrow [\underline{\mathbf{z}}; \overline{\mathbf{z}}] \leq \mathbf{0} \quad \Rightarrow \quad \overline{\mathbf{z}} \leq \mathbf{0} \Leftrightarrow \mathbf{h}^{\uparrow}([\mathbf{x}]) \leq \mathbf{0} \quad \Rightarrow \quad [\mathbf{x}] \subseteq \mathbf{h}^{-1}([\underline{\mathbf{z}}; \mathbf{0}]) \tag{4.24}$$

4.4.3.1 Example:

$$(x_1 - x_2)^2 - x_1 x_2 \leq 0 \tag{4.25}$$

The expression $(x_1 - x_2)^2 - x_1 x_2$ can be divided in a series of primitive operations (each one of them being a node in the expression tree).

$$e_1 = x_1 - x_2 \tag{4.26}$$
$$e_2 = x_1 x_2 \tag{4.27}$$
$$e_3 = e_1^2 \tag{4.28}$$
$$e_4 = e_3 - e_2 = (x_1 - x_2)^2 - x_1 x_2 \tag{4.29}$$

Assuming that the current interval assignment function is such that $x_1 \in s^{[\triangleright]}(x_1) = [x_1] = [-3;3]$ and $x_2 \in s^{[\triangleright]}(x_2) = [x_2] = [2;3]$, the enclosure of each node is computed with the forward evaluation step.

$$\begin{array}{rclcccc}
e_1 & \in & [x_1]-[x_2] = [-3;3]-[2;3] & = & [-6;1] & = & [e_1] \\
e_2 & \in & [x_1]\times[x_2] = [-3;3]\times[2;3] & = & [-9;9] & = & [e_2] \\
e_3 & \in & [e_1]^2 = [-6;1]^2 & = & [0;36] & = & [e_3] \\
e_4 & \in & [e_3]-[e_2] = [0;36]-[-9;9] & = & [-9;45] & = & [e_4]
\end{array}$$

The constraint is used to restrain the interval $[e_4]$ that contains all the accessible values for the root e_4. Because $e_4 = (x_1-x_2)^2 - x_1x_2$ and $(x_1-x_2)^2 - x_1x_2 \leq 0$ all the positive values inside $[e_4]$ are inconsistent. In order to respect this constraint the interval $[e_4]$ is contracted into a new one noted $[e_4]'$.

$$(e_4 \leq 0) \wedge (e_4 \in [e_4] = [-9;45]) \quad \Rightarrow \quad e_4 \in [-9;0] = [e_4]' \tag{4.30}$$

Similarly, all the nodes of the tree can recursively be contracted when processing the backward propagation (by using the specific contractors for each operation). For instance, from $e_4 = e_3 - e_2$, once $[e_4]$ is computed into $[e_4]' \subseteq [e_4]$, new enclosures $[e_3]'$ and $[e_2]'$ are computed. First the operations are reversed in order to isolate each node from the previously contracted nodes.

$$\begin{array}{rcccc}
e_4 = e_3 - e_2 & \Rightarrow & \left\{\begin{array}{l} e_3 = e_4 + e_2 \\ e_2 = e_3 - e_4 \end{array}\right. & \Rightarrow & \left\{\begin{array}{l} e_3 \in [e_4]' + [e_2] \\ e_2 \in [e_3] - [e_4]' \end{array}\right. \\
e_3 = e_1^2 & \Rightarrow & e_1 = \pm\sqrt{e_3} & \Rightarrow & e_1 \in \sqrt{[e_3]'} \cup -\sqrt{[e_3]'}
\end{array}$$

As a consequence, having knowledge that $e_3 \in [e_3]$, $e_2 \in [e_2]$ and $e_1 \in [e_1]$, the new enclosures $[e_3]'$, $[e_2]'$ and $[e_1]'$ can be computed as below

$$\begin{array}{rclcccc}
e_3 & \in & ([e_4]'+[e_2]) \cap [e_3] = [-18;9] \cap [0;36] & = & [0;9] & = & [e_3]' \\
e_2 & \in & ([e_3]-[e_4]') \cap [e_2] = [0;45] \cap [-9;9] & = & [0;9] & = & [e_2]' \\
e_1 & \in & \pm\sqrt{[e_3]'} \cap [e_1] = [-3;3] \cap [-6;1] & = & [-3;1] & = & [e_1]'
\end{array}$$

Ultimately, the enclosures $[x_1]$ and $[x_2]$ can be contracted using the expressions $e_1 = x_1 - x_2$, $e_2 = x_1x_2$ and the contracted enclosures $[e_1]'$, $[e_2]'$.

$$\begin{array}{rcccc}
e_1 = x_1 - x_2 & \Rightarrow & \left\{\begin{array}{l} x_1 = e_1 + x_2 \\ x_2 = x_1 - e_1 \end{array}\right. & \Rightarrow & \left\{\begin{array}{l} x_1 \in [e_1]' + [x_2] \\ x_2 \in [x_1] - [e_1]' \end{array}\right. \\
e_2 = x_1x_2 & \Rightarrow & \left\{\begin{array}{l} x_1 = e_2/x_2 \\ x_2 = e_2/x_1 \end{array}\right. & \Rightarrow & \left\{\begin{array}{l} x_1 \in [e_2]'/[x_2] \\ x_2 \in [e_2]'/[x_1] \end{array}\right.
\end{array}$$

$$\begin{array}{lllll}
x_1 & \in ([e_1]'+[x_2]) \cap ([e_2]'/[x_2]) \cap [x_1] & = [-1;4] \cap [0;9/2] \cap [-3;3] & = [0;3] & = [x_1]' \\
x_2 & \in ([x_1]-[e_1]') \cap ([e_2]'/[x_1]) \cap [e_2] & = [-4;6] \cap [-\infty;+\infty] \cap [2;3] & = [2;3] & = [x_2]'
\end{array}$$

In a nutshell, when x_1 in $[-3;3]$, x_2 in $[2;3]$ and $(x_1-x_2)^2 - x_1x_2 \leq 0$ the forward-backward contraction process proves that x_1 necessarily is in $[0;3]$ and x_2 is in $[2;3]$. Note that in this example, at the last step, the division has been used to inverse the multiplication. In practice, a peculiar operator is defined to inverse the multiplication in order to avoid any division by 0 (for example, when $x * y = z$ with $[y] = [0]$, the contraction of $[x]$ should not be computed with $[z]/[0]$) [Goualard, 2007].

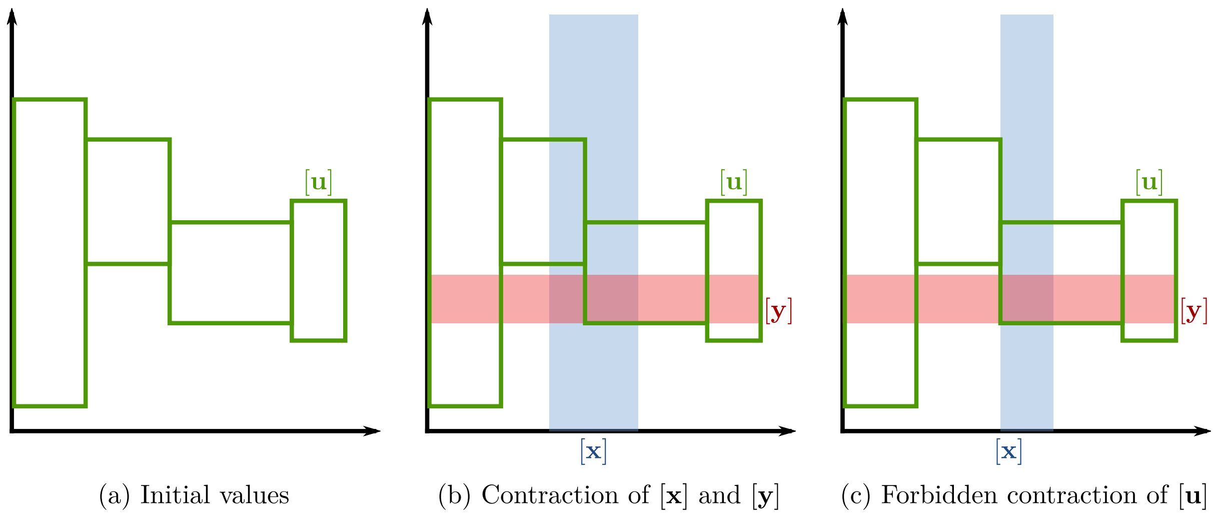
4.4.3.2 Warning on the contraction on tubes

The contraction that goes along with the forward-backward method can be used on tubes [Jaulin, 2002]. But such a contraction process should be used carefully on tubes. Indeed, the formal constraint

$$\mathbf{u}(\mathbf{x}) \leq \mathbf{y} \tag{4.31}$$

where $\mathbf{u}$ is a functional variable represented with the tube $[\mathbf{u}] = [\underline{\mathbf{u}}; \overline{\mathbf{u}}]$ and $\mathbf{x}$ (resp. $\mathbf{y}$) are real variables represented with the interval $[\mathbf{x}] = [\underline{\mathbf{x}}; \overline{\mathbf{x}}]$ (resp. $[\mathbf{y}] = [\underline{\mathbf{y}}; \overline{\mathbf{y}}]$) can be used in order to reduce the domains of the variables

- $\mathbf{y}$ using $\mathbf{u}(\mathbf{x})$ because the values $\mathbf{y} \in [\mathbf{y}]$ that are lower than the lower bound of $[\mathbf{u}]([\mathbf{x}])$ are inconsistent and can be removed (Fig. 4.3b).
- $\mathbf{x}$ using $\mathbf{u}$ and $\mathbf{y}$ because the values $\mathbf{x} \in [\mathbf{x}]$ such that the lower bound of $[\mathbf{u}](\mathbf{x})$ is greater than the upper bound of $[\mathbf{y}]$ can be removed (Fig. 4.3b).
- $\mathbf{u}$ using $\mathbf{x}$ and $\mathbf{y}$ when $[\mathbf{x}]$ is a degenerate interval that is to say when $\underline{\mathbf{x}} = \overline{\mathbf{x}}$. This condition is important, and on Figure 4.3b the interval $[\mathbf{x}]$ is large, then the contraction of the tube $[\mathbf{u}]$ that is represented on Figure 4.3c is forbidden. The reason the contraction is forbidden is detailed in the next paragraph.



(a) Initial values (b) Contraction of $[\mathbf{x}]$ and $[\mathbf{y}]$ (c) Forbidden contraction of $[\mathbf{u}]$

Figure 4.3: Contraction of $[\mathbf{x}]$ and $[\mathbf{y}]$

Considering $\mathbf{x}$, $\mathbf{y}$ in the boxes $[\mathbf{x}]$, $[\mathbf{y}]$ and $\mathbf{u}$ in the tube $[\mathbf{u}]$, the constraint

$$\mathbf{u}(\mathbf{x}) - \mathbf{y} \leq 0 \tag{4.32}$$

can be rewritten (with the interval extension)

$$[\mathbf{u}]([\mathbf{x}]) - [\mathbf{y}] \leq 0 \tag{4.33}$$

The forward-backward process tends to the functions $\mathbf{u} \in [\mathbf{u}]$ and the values $\mathbf{x} \in [\mathbf{x}]$ and $\mathbf{y} \in [\mathbf{y}]$ such that $\mathbf{u}(\mathbf{x})$ is lower than or equal to the value $\mathbf{y}$.

The restriction of the tube $[\mathbf{u}]$ should not occur over $[\mathbf{x}]$, except when the latter is a degenerate interval. When $\mathbf{x}$ is represented with an interval $[\mathbf{x}]$, it means *"a value $\mathbf{x}$ exists in $[\mathbf{x}]$ such that $[\mathbf{u}](\mathbf{x})$ is lower than $[\mathbf{y}]$"* which is very different than *"for all $\mathbf{x}$ in $[\mathbf{x}]$, $[\mathbf{u}](\mathbf{x})$ is lower than $[\mathbf{y}]$"*. Nevertheless, when the interval $[\mathbf{x}]$ is degenerate, then a single and unique value $\mathbf{x}$ does exist in $[\mathbf{x}]$ and, consequently, the contraction can be applied on the tube.

On the contrary, the box $[\mathbf{y}]$ can be contracted using the forward-backward contraction process while the box $[\mathbf{x}]$ is contracted depending on the tube $[\mathbf{u}]$ such that $[\mathbf{u}]([\mathbf{x}]) \leq [\mathbf{y}]$ (the values $\mathbf{x}$ in $[\mathbf{x}]$ such that the tube $[\mathbf{u}]$ evaluated over $\mathbf{x}$ is greater than $[\mathbf{y}]$ are inconsistent and can be removed).

A forward-backward approach that deals with algebraic constraints has been described. Unfortunately, some complex problems contain non-algebraic constraints which require the use of specific contractors. They can express some functional constraints like monotonicity, periodicity or differential constraints.

4.4.3.1 Increase, Decrease and Monotony

The terms *increase*, *decrease* and *monotony* are properties used to describe transformations performed between two ordered sets such that the order is preserved or reversed. These properties can be considered as constraints to filter any set of unary functions (which are defined as piecewise interval functions).

4.4.3.1 Definition: Increasing and decreasing piecewise interval functions
Let $[u]$ a tube defined from the domain $\mathbb{D}_{[u]}$ to the domain $\mathbb{R}$ with a piecewise interval function. If and only if for all pieces $([x_i] \mapsto [u_i])$ and $([x_j] \mapsto [u_j])$ in $[u]$ with $\overline{x_i} \leq \underline{x_j}$:

- $\overline{u_i} \leq \overline{u_j} \ \wedge \ \underline{u_i} \leq \underline{u_j}$ then the tube $[u]$ is increasing.
- $\overline{u_i} \geq \overline{u_j} \ \wedge \ \underline{u_i} \geq \underline{u_j}$ then the tube $[u]$ is decreasing.

Thus, assuming that the pieces of $[u]$ are ordered by the increasing values of the domain ($\overline{x_i} \leq \underline{x_{i+1}}$), the contraction of the tube $[u]$ with

- the contractor $\mathcal{C}_{increase}$ on any interval piece $[x_i] \mapsto [u_i]$ is done as follows (Fig. 4.4):

$$[u_i] \leftarrow [u_i] \cap [\underline{u_{i-1}}; \overline{u_{i+1}}] \tag{4.34}$$

- the contractor $\mathcal{C}_{decrease}$ on any interval piece $[x_i] \mapsto [u_i]$ is done as follows (Fig. 4.5):

$$[u_i] \leftarrow [u_i] \cap [\underline{u_{i+1}}; \overline{u_{i-1}}] \tag{4.35}$$

The monotonicity is the property of a transformation performed between two ordered sets, thus preserving or reversing the order. As a constraint, it can be applied to any set of unary functions and puts away the functions which are neither increasing nor decreasing.

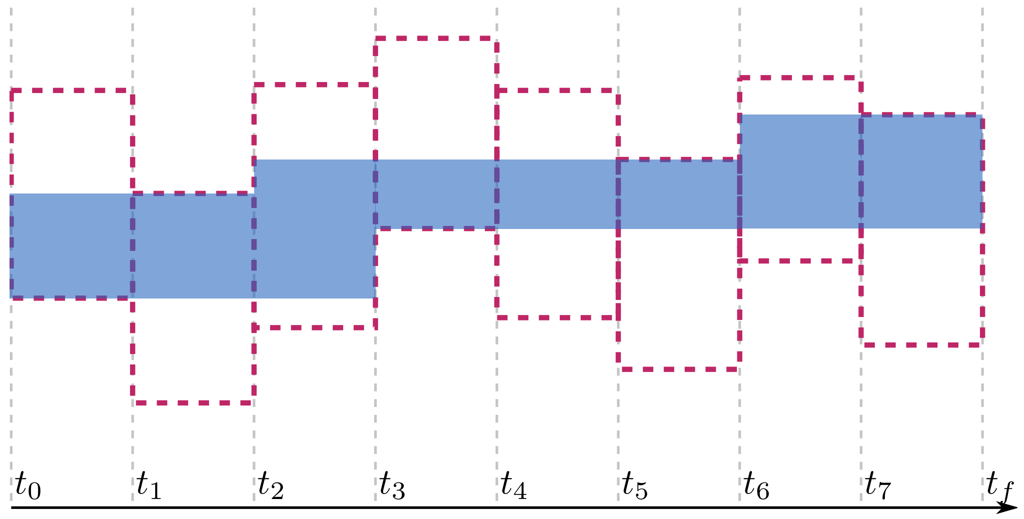


Figure 4.4: Contraction of the tube $[u]$ with the $\mathcal{C}_{increase}$

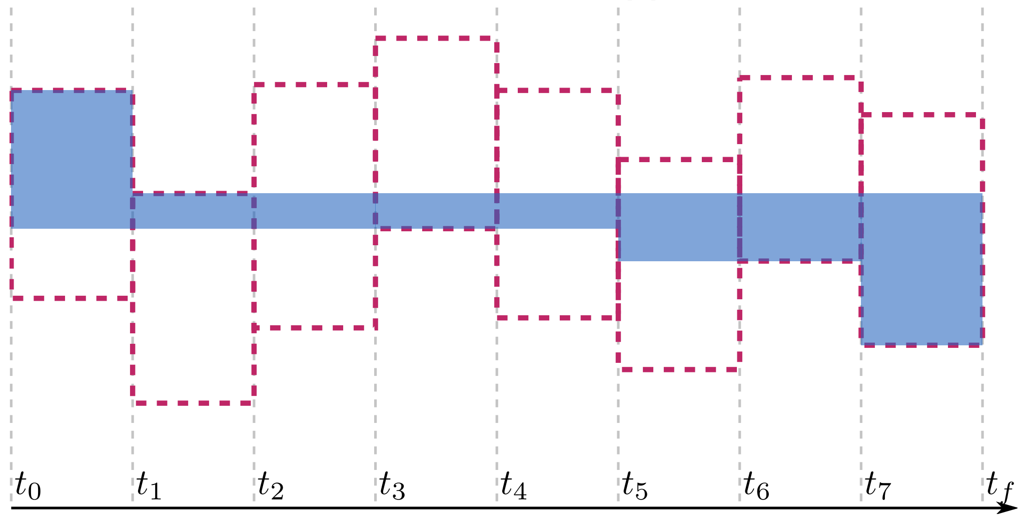


Figure 4.5: Contraction of the tube $[u]$ with the $\mathcal{C}_{decrease}$

4.4.3.2 Definition: Monotonicity

A function u, from an ordered set $(\mathbb{E}, \leq_{\mathbb{E}})$ to another ordered set $(\mathbb{F}, \leq_{\mathbb{F}})$, is monotonic iff u is either increasing or decreasing.

$$\forall (e_1, e_2) \in \mathbb{E}, \quad e_1 \leq_{\mathbb{E}} e_2 \ \Rightarrow \ u(e_1) \leq_{\mathbb{F}} u(e_2) \tag{4.36}$$

Thus, monotonicity can be applied on unary functions $u : x \mapsto u(x)$ such that $\mathcal{D}_u$ and $\mathcal{IM}_u$ are two ordered domains

$$\mathcal{D}_u = \{x \mid u(x) \text{ is defined}\}$$
$$\mathcal{IM}_u = \{u(x) : x \in \mathcal{D}_u\}$$

Let $\mathcal{U}$ a collection of functions, then, the monotonic contractor tends toward the subset $\mathcal{U}_{monotonic}$ of the monotonic functions in $\mathcal{U}$. Various implementations of this contractor is used depending on the knowledge it has gathered on the function $\mathbf{u}$.

The constraints of monotonicity, increase and decrease on a function $u(x)$ represented by a tube $[u]$ from the value $x = a$ to the value $x = b$ are expressed as follows

$$increase([u], [a; b]) \tag{4.37}$$
$$decrease([u], [a; b]) \tag{4.38}$$
$$monotonic([u], [a; b]) \tag{4.39}$$

Here, the attention is focused on the filtering of tubes that are defined as piecewise interval functions. The contractor $\mathcal{C}_{monotony}$ uses the contraction of the contractors $\mathcal{C}_{increase}$ and $\mathcal{C}_{decrease}$ depending on the situation. To begin with, a first strategy is to reduce the monotonicity constraint toward the increasing or decreasing constraint in order to perform a more efficient contraction (tinier enclosure and fast computation). This statement requires to find two pieces that would disjoint the values $[u_i]$ and $[u_j]$.

$$[u_i] \cap [u_j] = \emptyset$$

Assuming such pieces are found such that $[x_i]$ is lower than $[x_j]$ then if

- $[u_i] < [u_j]$ the tube $[u]$ is increasing and the contractor $\mathcal{C}_{increase}$ can be used.
- $[u_i] > [u_j]$ the tube $[u]$ is decreasing and the contractor $\mathcal{C}_{decrease}$ can be used.

When such pieces do not exist, the contractor $\mathcal{C}_{monotonic}$ that is associated to the monotonicity constraint of $[u]$ on the domain $[a; b] \subseteq \mathbb{D}_{[u]}$ noted $\mathcal{C}_{monotonic}([u], [a; b])$ can be defined as the union of the contractor $\mathcal{C}_{increase}$ and $\mathcal{C}_{decrease}$ (Fig. 4.6). The union is an operator that is used to build a meta-contractor which is described later.

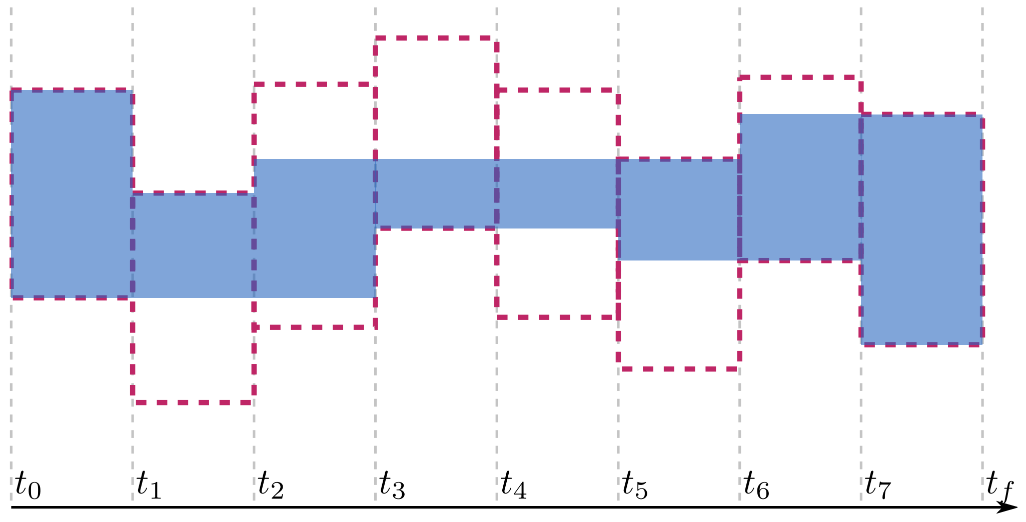


Figure 4.6: Contraction of the tube $[u]$ with the $\mathcal{C}_{monotonic}$

Note that when a tube is increasing (resp. decreasing), it can contains functions that are not necessarily increasing (resp. decreasing). The goal of these contractors is to filter some of the functions from the initial tubes that are not increasing (resp. decreasing).

4.4.4 Contraction with Ordinary Differential Equations

A function $\mathbf{u}$ represented by a tube $[\mathbf{u}] \in \mathbb{IF}^n$ can be contracted with differential constraints [Cruz and Barahona, 2003]

$$\mathbf{u}'(t) = \mathbf{d}(\mathbf{x}, \mathbf{u}(t)) \qquad \forall t \in [t_0; t_f] \tag{4.40}$$

using the specific contractor $\mathcal{C}_{diff}([\mathbf{u}], \mathbf{d}([\mathbf{x}], [\mathbf{u}]), [t_0; t_f])$.

The tube $[\mathbf{u}]$ is contracted using the guaranteed integration algorithm previously described in order to remove some parts of the enclosure $[\mathbf{u}]$ which are guaranteed not to contain functions that respect the differential constraints (and the tube restrictions). In this context, when the tube $[\mathbf{u}]$ is not restrictive enough (for example when $[\mathbf{u}]$ is defined such that $[\mathbf{u}] = \{[t_0; t_f] \mapsto [-\infty; +\infty]\}$), then the differential constraints cannot be used to contract the tube.

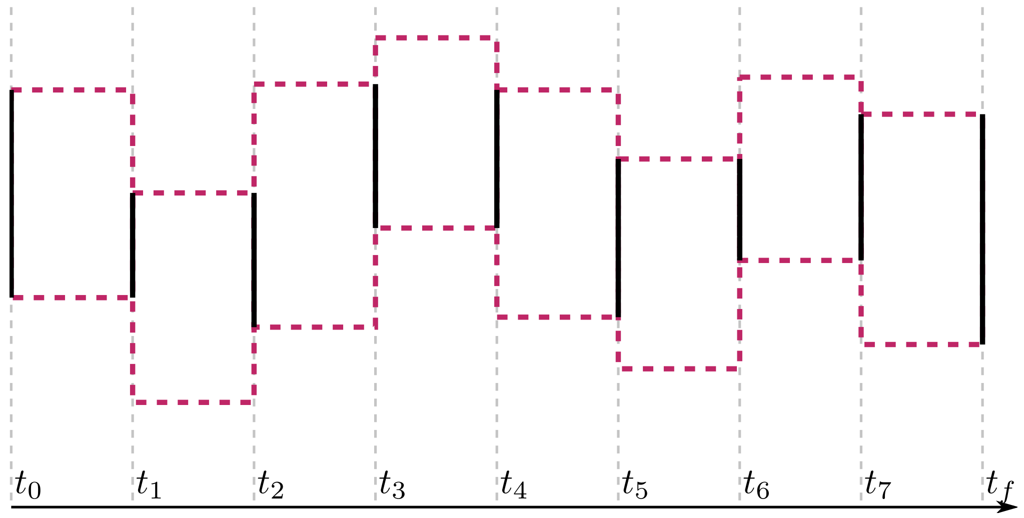


Figure 4.7: The tube $[u]$ encloses all the functions

To begin with, all the time-steps t_i from t_0 to t_f which bound the application domain of the pieces $[\underline{x_i}; \overline{x_i}] \mapsto [u_i]$ that constitute the tube $[u]$ are defined (Fig. 4.7).

$$\begin{aligned} t_0 &= \underline{x_0} \\ t_1 &= \overline{x_0} = \underline{x_1} \\ t_2 &= \overline{x_1} = \underline{x_2} \\ &\dots \end{aligned}$$

Then an initial value problem (noted $\mathcal{P}_i$) is defined for each value t_i.

$$\begin{aligned} &\mathbf{v}'(t) = \mathbf{d}(\mathbf{x}, \mathbf{v}(t)) \qquad \forall t \in [t_0; t_f] \\ &\mathbf{v}(t_i) \in [\mathbf{u_{i-1}}] \cap [\mathbf{u_i}] \end{aligned}$$

The contraction of the initial tube $[\mathbf{u}]$ can be performed by intersecting it with the tube $[\mathbf{v_i}]$ that is provided by the guaranteed integration over $[t_0; t_f]$ for each problem $\mathcal{P}_i$. They

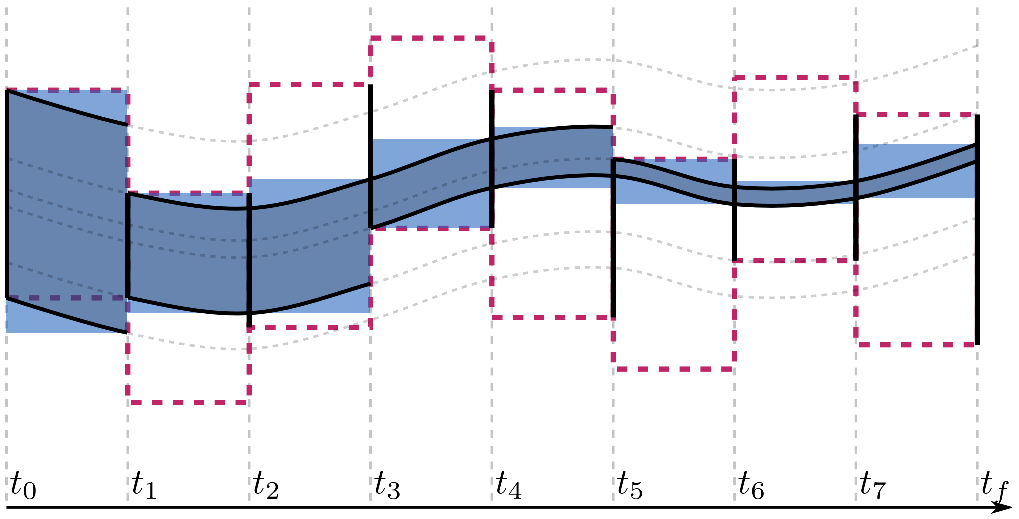


Figure 4.8: Forward integration

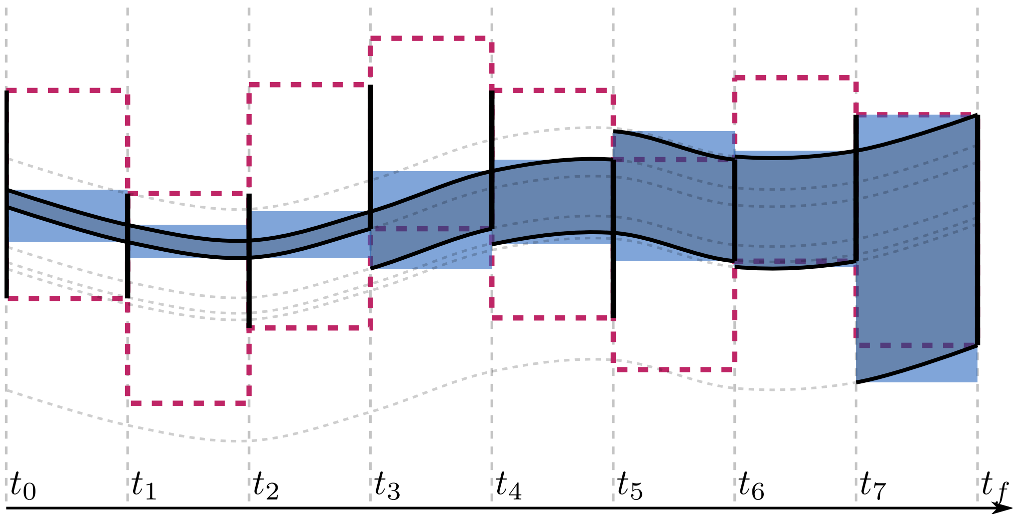


Figure 4.9: Backward integration

can be solved independently and, in order to have a maximal contraction through this process, it is required to have as many complete integrations as there are problems.

Because each integration costs memory and time processing, a limitation of the number of complete integrations into one contraction will be beneficial. One improvement is to use a forward and backward scheme (Fig. 4.10): it is based on two complete integrations, one from t_0 to t_f (Fig. 4.8) and the other from t_f to t_0 (Fig. 4.9), in order to decrease the complexity (that is to say the number of computations). From one piece $[t_i; t_{i+1}]$ to another $[t_{i+1}; t_{i+2}]$, the integration takes into account all the restrictions on each previous pieces $[t_0; t_{i+1}]$. Moreover, the contraction can be speed up by performing the integration over the pieces that can be reduced while ignoring the unmodified pieces, when using a dedicated lazy contractor (Section 4.4.5.5).

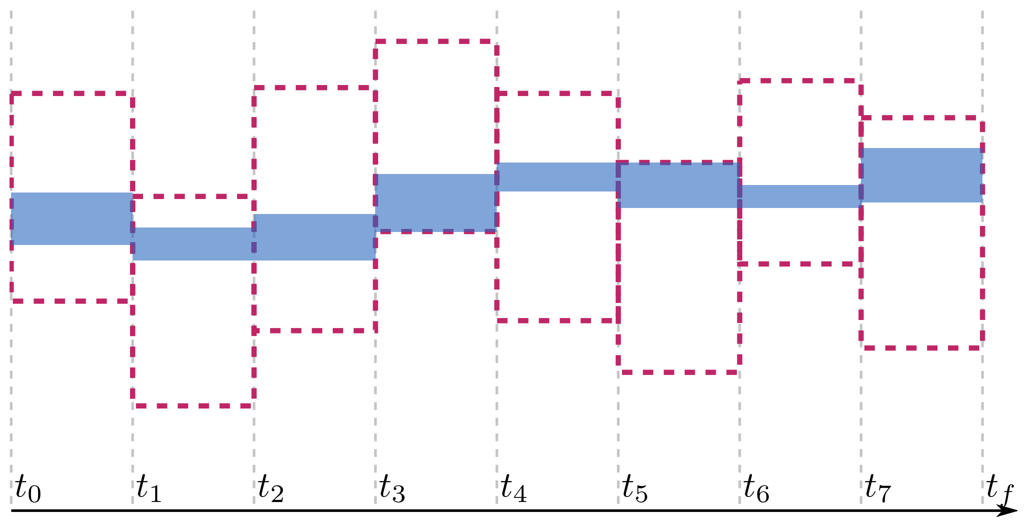


Figure 4.10: Intersection of the Forward and Backward integrations

4.4.5 Meta-Contractors

The Meta-Contractors [Chabert and Jaulin, 2009a] are operators on contractors, they are high order contractors that need sub-contractors in order to be operational. Note that meta-contractor can be applied on both interval contractors and tube contractors. Let three contractors $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ defined on intervals $[\mathbf{x}]$ and tubes $[\mathbf{u}]$.

$$\mathcal{C}_1([\mathbf{x}],[\mathbf{u}]) \quad , \quad \mathcal{C}_2([\mathbf{x}],[\mathbf{u}]) \quad \text{and} \quad \mathcal{C}_3([\mathbf{x}],[\mathbf{u}])$$

All the meta-contractors that are defined in the next paragraphs are represented through examples (Fig. 4.11, 4.12, 4.13, 4.14 and 4.15) by the contractors $\mathcal{C}_{\mathbb{X}}$, $\mathcal{C}_{\mathbb{X}_1}$, $\mathcal{C}_{\mathbb{X}_2}$, $\mathcal{C}_{\mathbb{X}_3}$. These contractors are applied on boxes and they are paired with the respective feasible domains $\mathbb{X}$, $\mathbb{X}_1$, $\mathbb{X}_2$ and $\mathbb{X}_3$.

$$\mathcal{C}_{\mathbb{X}}([\mathbf{x}]) \quad , \quad \mathcal{C}_{\mathbb{X}_1}([\mathbf{x}]) \quad , \quad \mathcal{C}_{\mathbb{X}_2}([\mathbf{x}]) \quad \text{and} \quad \mathcal{C}_{\mathbb{X}_3}([\mathbf{x}])$$

4.4.5.1 Intersection, Union and Composition

The intersection and union operators can be extended to contractors. Basically, the intersection of two contractors (Fig. 4.12a) creates a new contractor which is associated to the intersection of the feasible domains $\mathbb{X}_1$ and $\mathbb{X}_2$. Another contractor that characterizes the hull of the domains $\mathbb{X}_1$ and $\mathbb{X}_2$ can be built when using the union (Fig. 4.12b) of these contractors. These two contractors are defined as below

$$\mathcal{C}_{intersection}(\mathcal{C}_1,\mathcal{C}_2)([\mathbf{x}],[\mathbf{u}]) = (\mathcal{C}_1 \cap \mathcal{C}_2)([\mathbf{x}],[\mathbf{u}]) = \mathcal{C}_1([\mathbf{x}],[\mathbf{u}]) \cap \mathcal{C}_2([\mathbf{x}],[\mathbf{u}]) \tag{4.41}$$
$$\mathcal{C}_{union}(\mathcal{C}_1,\mathcal{C}_2)([\mathbf{x}],[\mathbf{u}]) = (\mathcal{C}_1 \cup \mathcal{C}_2)([\mathbf{x}],[\mathbf{u}]) = \mathcal{C}_1([\mathbf{x}],[\mathbf{u}]) \cup \mathcal{C}_2([\mathbf{x}],[\mathbf{u}]) \tag{4.42}$$

Another operation that links two contractors is the composition (Fig. 4.13). It builds a contractor from a chain made of two other contractors.

$$\mathcal{C}_{composition}(\mathcal{C}_1,\mathcal{C}_2)([\mathbf{x}],[\mathbf{u}]) = \mathcal{C}_1 \circ \mathcal{C}_2([\mathbf{x}],[\mathbf{u}]) = \mathcal{C}_2(\mathcal{C}_1([\mathbf{x}],[\mathbf{u}])) \tag{4.43}$$

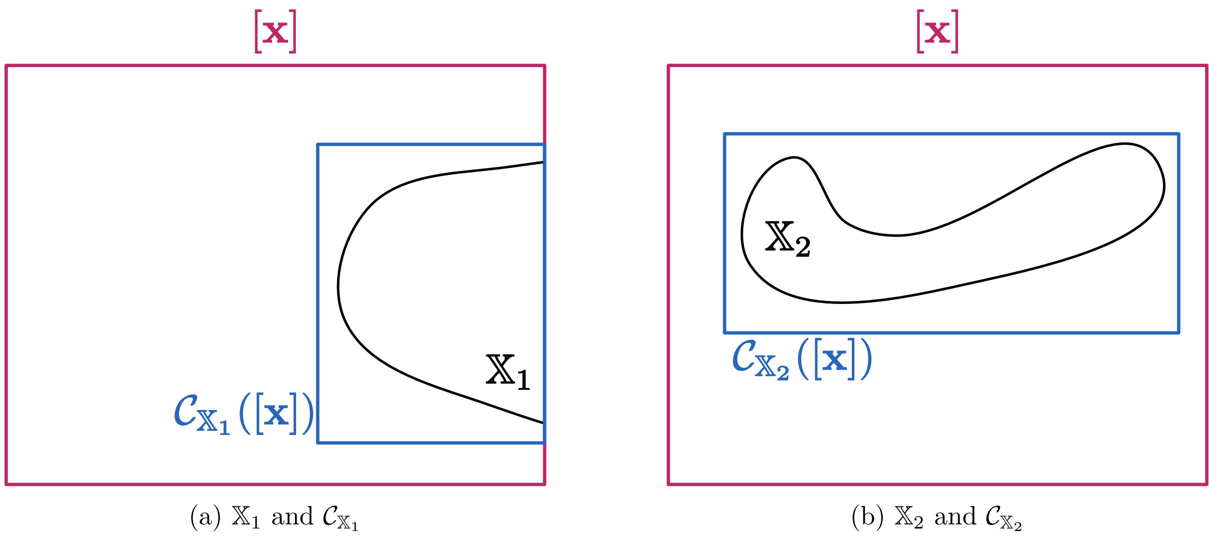

[x]
$\mathcal{C}_{\mathbb{X}_1}([\mathbf{x}])$
$\mathbb{X}_1$
(a) $\mathbb{X}_1$ and $\mathcal{C}_{\mathbb{X}_1}$
[x]
$\mathbb{X}_2$
$\mathcal{C}_{\mathbb{X}_2}([\mathbf{x}])$
(b) $\mathbb{X}_2$ and $\mathcal{C}_{\mathbb{X}_2}$

Figure 4.11: Domains $\mathbb{X}_1$, $\mathbb{X}_2$ with their associated contractors

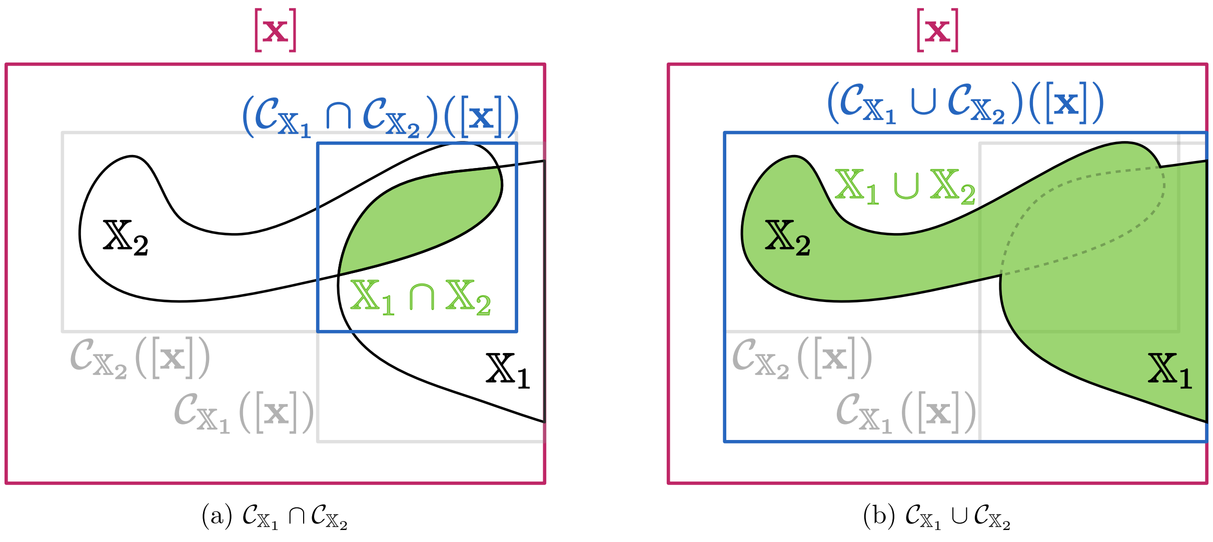
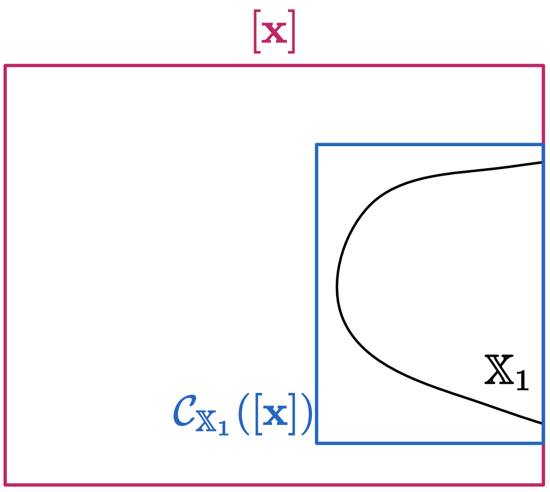

[x]
$(\mathcal{C}_{\mathbb{X}_1} \cap \mathcal{C}_{\mathbb{X}_2})([\mathbf{x}])$
$\mathbb{X}_2$
$\mathbb{X}_1 \cap \mathbb{X}_2$
$\mathcal{C}_{\mathbb{X}_2}([\mathbf{x}])$
$\mathcal{C}_{\mathbb{X}_1}([\mathbf{x}])$
$\mathbb{X}_1$
(a) $\mathcal{C}_{\mathbb{X}_1} \cap \mathcal{C}_{\mathbb{X}_2}$
[x]
$(\mathcal{C}_{\mathbb{X}_1} \cup \mathcal{C}_{\mathbb{X}_2})([\mathbf{x}])$
$\mathbb{X}_1 \cup \mathbb{X}_2$
$\mathbb{X}_2$
$\mathcal{C}_{\mathbb{X}_2}([\mathbf{x}])$
$\mathcal{C}_{\mathbb{X}_1}([\mathbf{x}])$
$\mathbb{X}_1$
(b) $\mathcal{C}_{\mathbb{X}_1} \cup \mathcal{C}_{\mathbb{X}_2}$

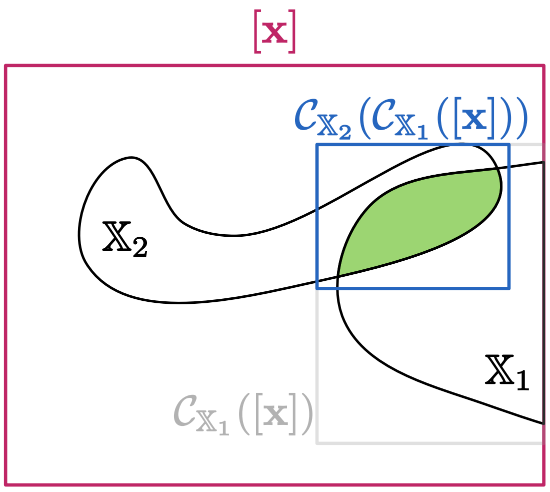
Figure 4.12: Intersection and union of the contractors $\mathcal{C}_{\mathbb{X}_1}$ and $\mathcal{C}_{\mathbb{X}_2}$

Usually the composition is preferred to the intersection because the convergence is faster while the feasible domains represented by these contractors remain identical. However the intersection can sometimes be more relevant when using parallel computing on $\mathcal{C}_1([\mathbf{x}],[\mathbf{u}])$ on one side and $\mathcal{C}_2([\mathbf{x}],[\mathbf{u}])$ on the other side.

In the next paragraphs are introduced various contractors which can be applied on a single variable x_i from the set of variables $\mathcal{V}$ that is defined in the CSP. Their application on sets of variables is made possible through the creation of a contraction chain (using the composition and the intersection of contractors).



(a) $\mathcal{C}_{\mathbb{X}_1}$



$[\mathbf{x}]$
$\mathcal{C}_{\mathbb{X}_2}(\mathcal{C}_{\mathbb{X}_1}([\mathbf{x}]))$
$\mathbb{X}_2$
$\mathbb{X}_1$
$\mathcal{C}_{\mathbb{X}_1}([\mathbf{x}])$

(b) $\mathcal{C}_{\mathbb{X}_1} \circ \mathcal{C}_{\mathbb{X}_2}$

Figure 4.13: Composition of the contractors $\mathcal{C}_{\mathbb{X}_1}$ and $\mathcal{C}_{\mathbb{X}_2}$

4.4.5.2 Piecewise Contractor

The piecewise contractor is inspired from the Constructive Interval Disjunction Contractor (CID) [Trombettoni and Chabert, 2007] which is based on the piecewise evaluation properties [Moore, 1966] and the use of another contractor to perform the sub-contractions. The piecewise contractor relies on the split of one dimension i (x_i), and on the subsequent contraction of the sub-nodes such as

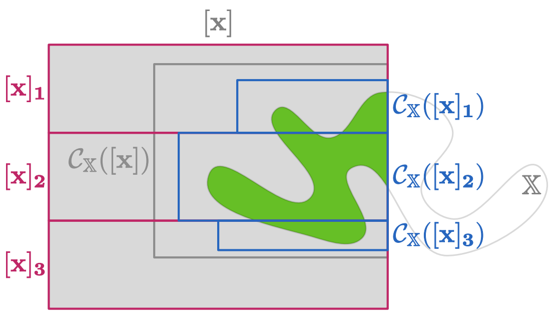
$$\mathcal{C}_{piecewise}(\mathcal{C}_3, x_i, k)([\mathbf{x}], [\mathbf{u}]) = \mathcal{C}_3\left(\begin{pmatrix} [x_1] \\ \vdots \\ [x_i]_1 \\ \vdots \\ [x_n] \end{pmatrix}, [\mathbf{u}]\right) \cup \cdots \cup \mathcal{C}_3\left(\begin{pmatrix} [x_1] \\ \vdots \\ [x_i]_k \\ \vdots \\ [x_n] \end{pmatrix}, [\mathbf{u}]\right) \tag{4.44}$$

where $[x_i] \subseteq \bigcup_{j=1}^{j\leq k} [x_i]_j$.

The use of piecewise contractors usually improves the contraction. When such is not the case it is because of the non-monotonicity of the initial contractor $\mathcal{C}_3$. One solution is to build a chain using the intersection or the composition contractor with the piecewise contractor and $\mathcal{C}_3$:

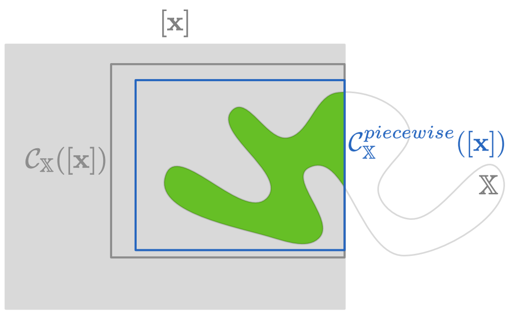
$$\mathcal{C}_{composition}(\mathcal{C}_{piecewise}(\mathcal{C}_3, x_i, k), \mathcal{C}_3)$$
$$\mathcal{C}_{intersection}(\mathcal{C}_{piecewise}(\mathcal{C}_3, x_i, k), \mathcal{C}_3)$$

Also, the contractor $\mathcal{C}_{piecewise}(\mathcal{C}_3, x_i, k)$ requires k times more operations than $\mathcal{C}_3$. Attention should be focused on the relative position of the piecewise contractor inside the contractive chain. Indeed, depending on which meta-contractor $\mathcal{C}_3$ is used to create the chain, the computational complexity of the contraction may become exponential.



$[\mathbf{x}]$
$[\mathbf{x}]_1$
$[\mathbf{x}]_2$
$[\mathbf{x}]_3$
$\mathcal{C}_{\mathbb{X}}([\mathbf{x}])$
$\mathcal{C}_{\mathbb{X}}([\mathbf{x}]_1)$
$\mathcal{C}_{\mathbb{X}}([\mathbf{x}]_2)$
$\mathcal{C}_{\mathbb{X}}([\mathbf{x}]_3)$
$\mathbb{X}$

(a) contractions



$[\mathbf{x}]$
$\mathcal{C}_{\mathbb{X}}([\mathbf{x}])$
$\mathcal{C}_{\mathbb{X}}^{piecewise}([\mathbf{x}])$
$\mathbb{X}$

(b) final

Figure 4.14: Piecewise meta-contractor over $\mathcal{C}_{\mathbb{X}}$

4.4.5.1 Remark:

In this example (Fig. 4.14) it is assumed the contractor $\mathcal{C}_{\mathbb{X}}$ is monotonic. The initial domain $[\mathbf{x}]$ is split in three parts $[\mathbf{x}]_1$, $[\mathbf{x}]_2$ and $[\mathbf{x}]_3$ following one dimension that is defined by the variable x_i. The result $\mathcal{C}_{piecewise}(\mathcal{C}_{\mathbb{X}}, x_i, 3)([\mathbf{x}])$ of the contraction noted $\mathcal{C}_{\mathbb{X}}^{piecewise}([\mathbf{x}])$ on Figure 4.14b is the union of each contraction $\mathcal{C}_{\mathbb{X}}([\mathbf{x}]_1)$, $\mathcal{C}_{\mathbb{X}}([\mathbf{x}]_2)$ and $\mathcal{C}_{\mathbb{X}}([\mathbf{x}]_3)$. Because of the piecewise evaluation property on intervals, the result $\mathcal{C}_{\mathbb{X}}^{piecewise}([\mathbf{x}])$ is more accurate than $\mathcal{C}_{\mathbb{X}}([\mathbf{x}])$.

4.4.5.3 Erosion

The goal of the erosion that is inspired from the shaving operator [Chabert and Jaulin, 2009a] is to erase the extremities of the domain $[x_i]$ associated to the variable x_i. On each interval $[x_i]_j$ from the multi-interval value $[x_i]$ two external oriented slices $[x_i]_{jL}$ and $[x_i]_{jR}$ are considered. When using the subcontractor $\mathcal{C}_3$, whenever an inconsistency is found within any slice, then the slice can be removed from the domain $[x_i]$. Otherwise, when none is found, the size of the slice is only reduced and a new iteration is performed. The contractor $\mathcal{C}_{erode}$ can be defined as follows:

$$\mathcal{C}_{erode}(\mathcal{C}_3, x_i, k)([\mathbf{x}], [\mathbf{u}]) = ([\mathbf{x}], [\mathbf{u}]) \setminus (\mathcal{C}_{erodeL}(\mathcal{C}_3, x_i, k) \cup \mathcal{C}_{erodeR}(\mathcal{C}_3, x_i, k))\,([\mathbf{x}], [\mathbf{u}])$$

where $\mathcal{C}_{erodeL}$ and $\mathcal{C}_{erodeR}$ return the parts the domain $[x_i]$ that are inconsistent via the contractor $\mathcal{C}_3$.

$$\mathcal{C}_{erodeL}(\mathcal{C}_1, x_i, k)([\mathbf{x}], [\mathbf{u}]) = \begin{cases} \mathcal{C}_{erodeL}(\mathcal{C}_1, x_i, k-1)([\mathbf{x}]_\mathbf{L}, [\mathbf{u}]) & \text{\textit{if} } \mathcal{C}_1([\mathbf{x}]_\mathbf{L}, [\mathbf{u}]) \neq \emptyset \\ ([\mathbf{x}]_\mathbf{L}, [\mathbf{u}]) \cup \mathcal{C}_{erodeL}(\mathcal{C}_1, x_i, k-1)([\mathbf{x}]_\mathbf{R}, [\mathbf{u}]) & \textit{otherwise} \end{cases}$$

$$\mathcal{C}_{erodeR}(\mathcal{C}_1, x_i, k)([\mathbf{x}], [\mathbf{u}]) = \begin{cases} \mathcal{C}_{erodeR}(\mathcal{C}_1, x_i, k-1)([\mathbf{x}]_\mathbf{R}, [\mathbf{u}]) & \text{\textit{if} } \mathcal{C}_1([\mathbf{x}]_\mathbf{R}, [\mathbf{u}]) \neq \emptyset \\ ([\mathbf{x}]_\mathbf{R}, [\mathbf{u}]) \cup \mathcal{C}_{erodeR}(\mathcal{C}_1, x_i, k-1)([\mathbf{x}]_\mathbf{L}, [\mathbf{u}]) & \textit{otherwise} \end{cases}$$

with $[\mathbf{x}]_\mathbf{L} = ([x_1], \dots, [x_i]_L, \dots, [x_n])$ and $[\mathbf{x}]_\mathbf{R} = ([x_1], \dots, [x_i]_R, \dots, [x_n])$ with $[x_i]_L = [\underline{x_{iL}}; \overline{x_{iL}}]$ such that $\underline{x_{iL}} = \underline{x_i}$ and $[x_i]_R = [\underline{x_{iR}}; \overline{x_{iR}}]$ such that $\overline{x_{iR}} = \overline{x_i}$. This is a recursive method, with decreasing value of k. When $k = 0$ the recursion stop.

$$\mathcal{C}_{erodeL}(\mathcal{C}_1, x_i, 0)([\mathbf{x}], [\mathbf{u}]) = \emptyset$$
$$\mathcal{C}_{erodeR}(\mathcal{C}_1, x_i, 0)([\mathbf{x}], [\mathbf{u}]) = \emptyset$$

An improvement of the 3B-coherence presented here is the partition-one arc consistency (POAC) that is discussed in [Bennaceur and Affane, 2001], [Balafrej et al., 2014]. It speeds up the contraction because it is able to prune values on several variable, which is not the case with the operator that is described here.

4.4.5.4 Fixed point

Let $\mathcal{C}_{\mathbb{X}}$ a contractor for $\mathbb{X}$. The fixed point contractor $\mathcal{C}_{\infty}(\mathcal{C}_3)$ is equivalent to an infinite composition of $\mathcal{C}_3$ [Chabert and Jaulin, 2009a].

$$\mathcal{C}_{\infty}(\mathcal{C}_3)([\mathbf{x}],[\mathbf{u}]) = \mathcal{C}_3 \circ \cdots \circ \mathcal{C}_3([\mathbf{x}],[\mathbf{u}]) \tag{4.45}$$

The fixed point meta-contractor $\mathcal{C}_{\infty}$ on any contractor $\mathcal{C}_3$ is idempotent (see §4.4.1.2). If $\mathcal{C}_3$ is minimal, then $\mathcal{C}_{\infty}(\mathcal{C}_3)$ remains its equivalent and is therefore useless.

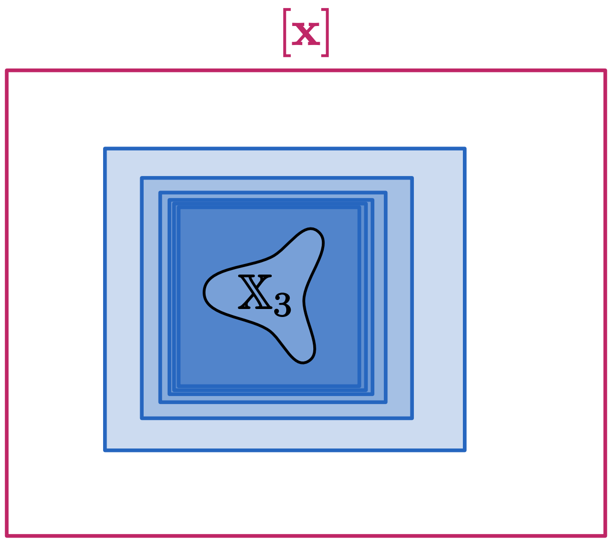


Figure 4.15: fixed point contractor

In practice, even though the fixed point contractor consists in an infinite computation of the composition, it only computes the right amount of iterations to approach the convergent point. Basically, in order to avoid an infinite computation, the contractor is not called for when unnecessary (the last computation is not contractive enough) and/or a limit is set on the iteration number. On Figure 4.15, the contraction appears to be slower at each iteration.

4.4.5.5 Lazy

The role of the lazy contractor is to avoid useless contractions in order to save computation time. It is made of a set of variables $\mathcal{V}_l \subseteq \mathcal{V}$, and a sub contractor $\mathcal{C}_3$. Each time the contractor $\mathcal{C}_{lazy}$ is called, the current domains $[x_i]$ which are associated to each variable x_i from $\mathcal{V}_l$ are compared to the ones that were used in the last contraction $[x_i]_{prev}$ performed. The contractor $\mathcal{C}_3$ is called as long as one of the domain is significantly contracted (meaning it goes beyond a minimal contraction limit once compared to the previous iteration).

$$\mathcal{C}_{lazy}(\mathcal{C}_3, \mathcal{V}_l, [\mathbf{x}]_{\mathbf{prev}}, [\mathbf{u}]_{\mathbf{prev}}, \alpha)([\mathbf{x}], [\mathbf{u}]) = \begin{cases} \mathcal{C}_3([\mathbf{x}], [\mathbf{u}]) & \text{if } \exists x_i \in \mathcal{V}_l, \dfrac{width([x_i])}{width([x_i]_{prev})} < \alpha \\ ([\mathbf{x}], [\mathbf{u}]) & \text{otherwise} \end{cases}$$

When the contraction happens, the domains $[\mathbf{x}]_{\mathbf{prev}}$ and $[\mathbf{u}]_{\mathbf{prev}}$ are updated with the values returned by $\mathcal{C}_3([\mathbf{x}], [\mathbf{u}])$. Such a contractor is useful to avoid computation from complex contractors when these are not efficient enough. It can be combined with the fixed point to limit its infinite number of contraction.

4.5 Set Inversion Via Interval Analysis

The contraction process was defined in the previous section. It is used to converge towards solutions of one or more sets of constraints. Unfortunately, what results from the intervals computed through the contraction processes previously described may remain inconsistent with the constraints they are associated with. The desire to get the tiniest enclosure possible leads to the use of another tool: the Set Inversion Via Interval Analysis algorithm (SIVIA) [Jaulin and Walter, 1993].

The method has been developed to get the inverse image $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{y})$ such that $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ where $\mathbf{f}$ is a function from $\mathbb{X}$ to $\mathbb{Y}$. SIVIA can also be used to specify some of the feasible and unfeasible parts of the domains according to a constraint (a part of the domain still remains ambiguous and draws a line between the feasible and unfeasible subdomains).

The initial enclosure $[\mathbf{x}]$ of $\mathbf{f}^{-1}([\mathbf{y}])$ is computed with interval and tube arithmetics using the results of several evaluations of the interval function $\mathbf{f}$ over subdomains $[\mathbf{x}]'$ from $\mathbb{X}$. Then, depending on the intersection with $[\mathbf{y}]$, the respective results $[\mathbf{y}]'$ are used to define the initial domain $[\mathbf{x}]'$. This process can be seen on Figures 4.16, 4.17 and 4.18.

In Figure 4.16, three initial boxes $[\mathbf{x_1}]$, $[\mathbf{x_2}]$ and $[\mathbf{x_3}]$ picture different possibilities for $[\mathbf{x}]'$. These values are associated to the respective boxes $\mathbf{f}([\mathbf{x_1}])$, $\mathbf{f}([\mathbf{x_2}])$ and $\mathbf{f}([\mathbf{x_3}])$ through the function $\mathbf{f}$.

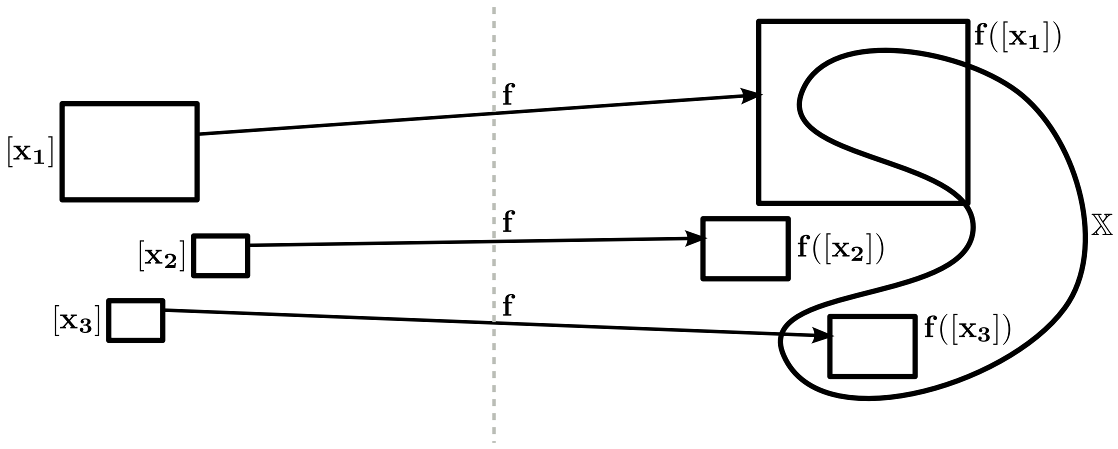


Figure 4.16: Sivia results - step 1: evaluation with $\mathbf{f}$

Three distinctive situations are pictured depending on the relative positions of the final boxes ($\mathbf{f}([\mathbf{x_1}])$, $\mathbf{f}([\mathbf{x_2}])$, $\mathbf{f}([\mathbf{x_3}])$) in relation with the validity of the constraint domain $\mathbb{X}$ (Fig. 4.17):

- When the final box is guaranteed to be out of the domain $\mathbb{X}$, it is red: $\mathbf{f}([\mathbf{x_2}])$.
- When the final box is guaranteed to be inside of the domain $\mathbb{X}$, it is green: $\mathbf{f}([\mathbf{x_3}])$.
- Otherwise, the situation is unclear. A part of the final box goes inside the domain $\mathbb{X}$ while the other part goes out of it. The box is gray: $\mathbf{f}([\mathbf{x_1}])$.

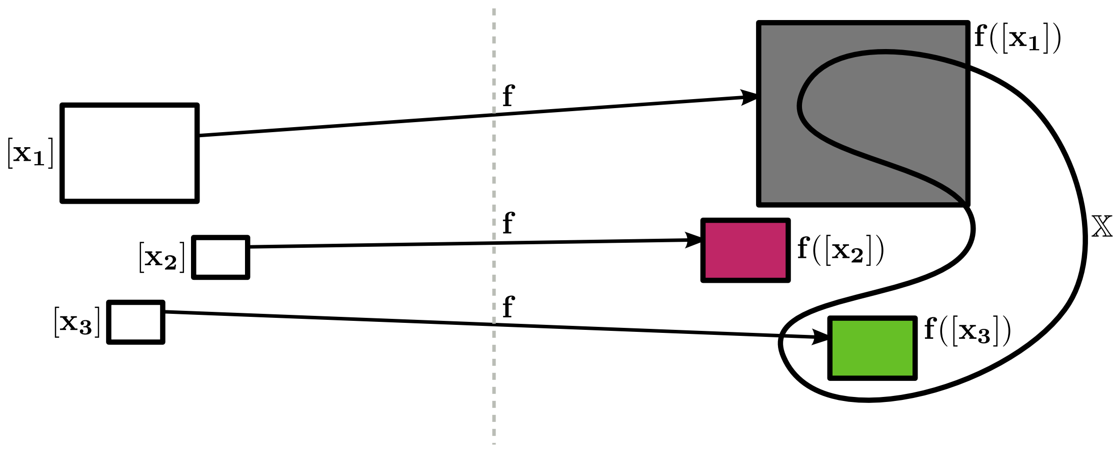


Figure 4.17: Sivia results - step 2: information on $\mathbf{f}([\mathbf{x}])$

At last, the colors assigned to $\mathbf{f}([\mathbf{x_1}])$, $\mathbf{f}([\mathbf{x_2}])$ and $\mathbf{f}([\mathbf{x_3}])$ are used to characterize the initial domains $[\mathbf{x_1}]$, $[\mathbf{x_2}]$ and $[\mathbf{x_3}]$ (Fig. 4.18).

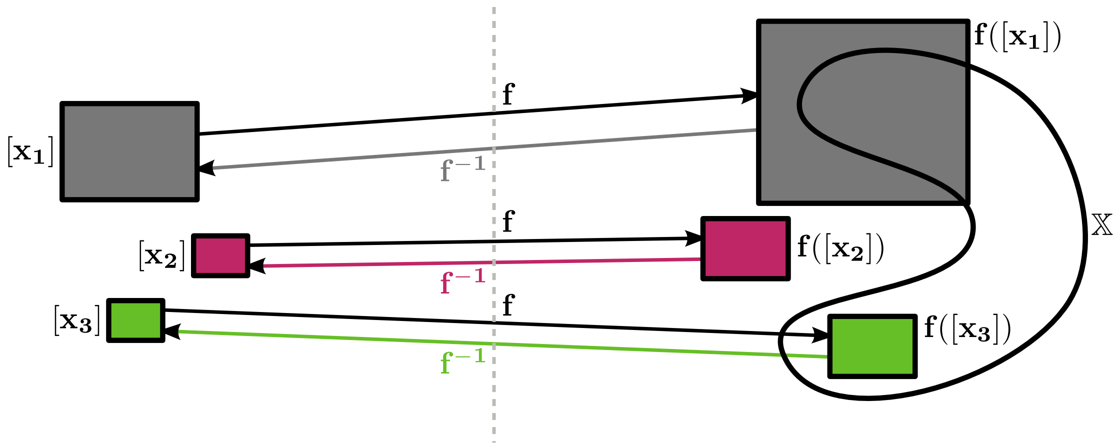


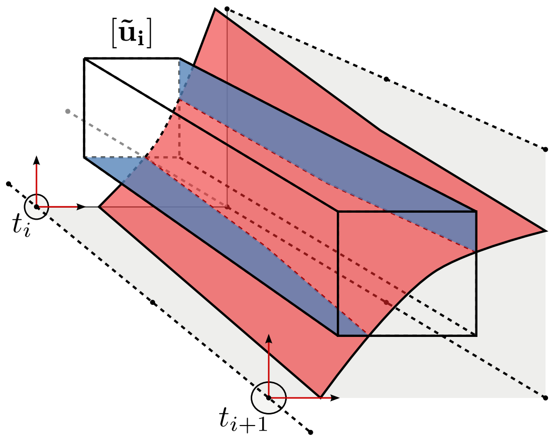
Figure 4.18: Sivia results - step 3: backward propagation of information with $\mathbf{f}^{-1}$

Let $[\mathbf{x}]'$ an enclosure which is guaranteed to enclose $\mathbf{f}^{-1}([\mathbf{y}])$, then the global algorithm consists in a succession of subdivisions over the domain $[\mathbf{x}]'$. While $[\mathbf{y}]' = \mathbf{f}([\mathbf{x}]')$ is not guaranteed to be exclusively inside or exclusively outside the validity domain $\mathbb{X}$ (like $\mathbf{f}([\mathbf{x_1}])$ on Figure 4.18), $[\mathbf{x}]'$ is split into several pieces on which the SIVIA is applied. This process is represented on Figure 4.19.

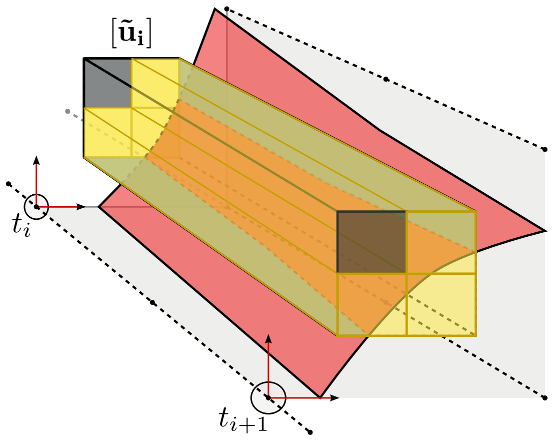
On Figure 4.19, the SIVIA is applied on a tube $[\mathbf{u}](t)$ that is restrained to a single piece:

$$[t_i; t_{i+1}] \mapsto [\tilde{\mathbf{u}}_\mathbf{i}]$$

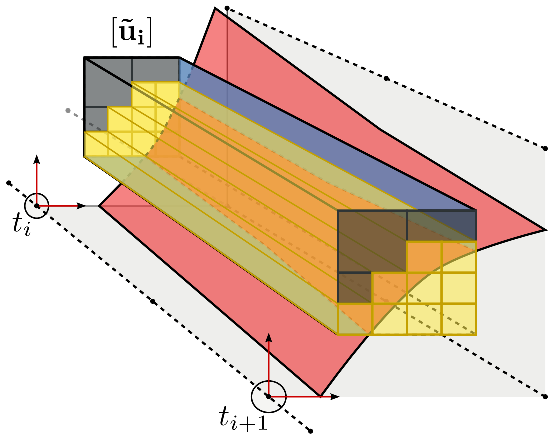
A function that is represented with the red surface that crosses the tube (Fig. 4.19a). The red section splits the tube in two parts: the upper parts on the top-left corner and the lower part on the bottom-right corner. The cross section can be defined with an explicit multi-dimensional functions $[\mathbf{f}]([\mathbf{u}], t)$ and used as a constraint on the tube to



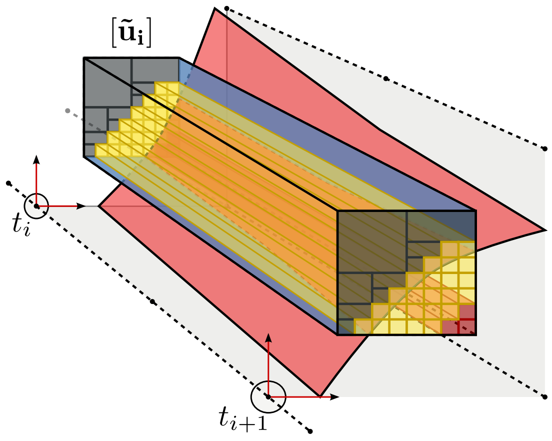
(a) initial step



(b) step 2



(c) step 4



(d) final step

Figure 4.19: Application of the SIVIA on a tube

remove the bottom-right corner of $[\tilde{\mathbf{u}}_\mathbf{i}]$.

$$[\mathbf{u}](t) \geq [\mathbf{f}]([\mathbf{u}], t) \quad \forall t \in [t_i; t_{i+1}]$$

The contractors are not able to contract the tube $[\mathbf{u}]$, but the SIVIA can be used to define parts of the tube that are inconsistent.

At the step two (Fig. 4.19b), the initial tube is split into four parts. One part of the tube is guaranteed to be consistent and is represented in black. The yellow tubes do not provide relevant information. Consequently, they are split one more time into tinier tubes, this is the step four (Fig. 4.19c) and the information is better because three parts of the tubes are guaranteed to be consistent with the constraint (black tubes). The other domains of the tube $[\mathbf{u}]$ (yellow tubes) require to be split again in order to extract more information according to the consistency with the constraint.

At the final step (Fig. 4.19d) the three tubes in red on the bottom-right corner are guaranteed to be inconsistent. Note that in this example, the SIVIA only splits the tube

$[\mathbf{u}]$ on its domain $[\tilde{\mathbf{u}}_\mathbf{i}]$ but the time windows $[t_i; t_{i+1}]$ can be split as well.

The SIVIA can be applied in several situations such that state estimation, robust control, robotics [Jaulin et al., 2001]. In this work, SIVIA can be used when it comes to integrate piecewise ordinary differential equations (Chapter 5), because they are defined with several dynamics paired with guards. On Figure 4.19d the yellow part of the tube is the intersection between the tube $[\mathbf{u}]$ that could be the guaranteed enclosure of an ODE and a guard on $[t_i; t_{i+1}]$. Unfortunately, this algorithm is costly in computational time, that is why it has to be used carefully: to limit the number of recursive iterations is an easy and efficient trick to limit the complexity of the algorithm.

4.5.1 IBEX library and plugins

The Interval Based EXplorer (IBEX - `http://www.ibex-lib.org/`) [Chabert, 2007] is a library based on several layers that are the guaranteed arithmetics layer (with the interval arithmetic and the affine arithmetic presented in Chapter 2) and the constraint programming layer with contractor programming [Chabert and Jaulin, 2009a] that has been introduced in this chapter.

A plugin named IbexSolve has been developed and solves rigorously non-linear equations systems. All the contractors discussed in this chapter are implemented with other specific contractors in order to improve the efficiency of the solver. The constructive interval disjunction contractor introduced in [Trombettoni and Chabert, 2007] has been improved ACID in [Neveu and Trombettoni, 2013] and [Neveu et al., 2015]. The hull consistency (HC4) is improved with monotonicity properties in [Chabert and Jaulin, 2009b], [Araya et al., 2010]. A contractor based on the convexification of the constraints is described in [Araya et al., 2012]. A specific contractor that is used to compute efficiently the q-relaxed intersection of several boxes is detailed in [Carbonnel et al., 2014].

Recently, a specific plugin named DynIbex that is under development aims to solve initial value problems of ordinary differential equations (Chapter 3) with Runge-Kutta schemes and affine arithmetic [Alexandre dit Sandretto and Chapoutot, 2016a] to guarantee the integration.

A global optimization plugin named IbexOpt [Trombettoni et al., 2011], [Ninin, 2015] is detailed in Section 6.3.

4.6 Conclusion

This chapter introduced the basics of constraint satisfaction problems (Section 4.2) and constraint programming on intervals (Section 4.3) and tubes by using the interval arithmetic computation as well as the tube arithmetic and contractors (Section 4.4).

Essential bricks have been developed to deal with specific constraints using atomic contractors. With HC4 is the classic approach to deal with algebraic constraints, but monotonic or differential contractors can deal with some functional and differential constraints. Afterwards, high level contractors are used to get more accurate results. This entire process leads to the creation of a chain made of several contractors such as

$$\mathcal{C}_\infty\left(\mathcal{C}_{piecewise}\left(\mathcal{C}_3, x_1, 8\right) \cap \mathcal{C}_{lazy}\left(\mathcal{C}_1 \cap \mathcal{C}_2, \{x_2, x_4\}, 10^{-2}\right)\right) \cap \cdots \cap \mathcal{C}_{erode}(\mathcal{C}_4, x_2, 6)$$

These chains are more efficient than atomic contractors because they exploit some specific properties of the interval arithmetic. Nevertheless, it should be underlined that there are some contrasts between these results and the increase of computation time and of memory requirements.

Part III

Toward the Guaranteed Integration of Piecewise Ordinary Differential Equations and Global Optimization

5 Complex Ordinary Differential Equations

Abandon the urge to simplify everything, to look for formulas and easy answers, and to begin to think multidimensionally, to glory in the mystery and paradoxes of life, not to be dismayed by the multitude of causes and consequences that are inherent in each experience – to appreciate the fact that life is complex.

– M. Scott Peck

Contents

5.1 Introduction

In the previous chapters, some specific methods have been introduced in order to perform rigorous computation with intervals and tubes. In the problems that are considered, many constraints are used to bound the feasible domain. From these ones, the algebraic constraints are treated with the atomic contractor HC4 and the differential constraints are propagated using the differential contractor that is made of the guaranteed integration scheme that has been detailed previously. However, the method to take into account the piecewise differential constraints that are the specificity of the problems that are considered has not been developed yet. Thus, the objective of this chapter is to detail such a method as well as to proof that the results provided with the integration algorithm are guaranteed.

The initial optimization problem that is tackled in this section includes some piecewise differential equations. Thus, it must be noted that each piece is defined by an Ordinary Differential Equation (ODE) and that the complexity resides in the fact that all of the pieces intersect each other at some point.

First, a definition of the piecewise-ODE system and of some metrics are introduced in Section 5.2. Afterwards, the uncertain time initial value problems (which is a variant of the initial value problem) are presented along with the process of guaranteed integration in Section 5.3. Ultimately the guaranteed integration process that is required to solve piecewise-ODE is discussed and demonstrated in Section 5.4.

5.2 Piecewise Ordinary Differential Equations

The Piecewise Ordinary Differential Equation systems allow the creation of dynamic systems that are much more complex than simple ODEs. Thanks to ODEs, the regular dynamic systems can be described, but the systems that are considered in this thesis imply that several dynamics could be applied on the same solution along its resolution. The passage from one dynamic to another is performed on a breaking point.

Many solutions can be applied depending on what is known about the breaking point to take it into consideration. When the break occurs at an accurate time-step that is independent of the integration, then multiple ODEs can be used to solve the different parts of the solution (before and after the breaking point). In the model that has been introduced at the beginning of the thesis, some complex dynamic systems were presented that may possess several breaking points. Moreover, these breaking points cannot be planned out because their creations may differ according to the resolution, the initial values, the dynamics and the guards the dynamics are paired with.

5.2.1 Definitions

5.2.1.1 Definition: Piecewise-ODE

A piecewise-ODE (also noted p-ODE) is defined as a set of ODEs guarded by logical expressions $c_p(t, \mathbf{u}(t))$ with $p = 1, \ldots, q$ Eq. (5.1).

$$\mathbf{u}'(t) = \begin{cases} \mathbf{d_1}(t, \mathbf{u}(t)) & \text{if} \quad c_1(t, \mathbf{u}(t)) \\ & \vdots \\ \mathbf{d_q}(t, \mathbf{u}(t)) & \text{if} \quad c_q(t, \mathbf{u}(t)) \end{cases} \tag{5.1}$$

where $\mathbf{u} \in \mathbb{R} \to \mathbb{R}^n$ and for all p from 1 to q, $\mathbf{d_p} \in \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ and $c_p \in \mathbb{R} \times \mathbb{R}^n \to \mathbb{B}$.

5.2.1.2 Important note:

Systems such as this one Eq. (5.1) are hardly solvable due to their non-continuous behavior: the solution can oscillate between many states (from 1 to q) being driven by several differential functions and because of the trajectory changes happening at unknown times (depending on the resolution).

5.2.1.3 Hybrid Systems

The hybrid systems [Ábrahám and Schupp, 2012] are dynamic systems that combines continuous and discrete dynamic behaviors. They are defined with a state machine in which each state is paired with a dynamic and a set of transition states that model the changes from one state to another. These transition states can be controlled with an external operator, discrete conditions, or the state of the dynamic.

Consequently, the piecewise-ODE that are considered in this thesis correspond to a specific case of hybrid systems such that

- the states s_p of the automaton are paired with the pieces $p \in \{1 \dots q\}$ of the p-ODE,
- the dynamic applied on each state s_p is the dynamic $\mathbf{d_p}$ of the piece p it is associated with.
- all the states of the automaton are connected together through transition states and the state could change from the state s_i to another state s_j when the guard c_j is activated.

The reachability problem[1] on hybrid automata is undecidable [Alur et al., 1993]. A lot of tools and results exist on specific cases of hybrid systems such as timed automata or linear hybrid automata. Hyson [Bouissou et al., 2012] is a set based simulation tool that computes a precise simulation, which is not guaranteed, of uncertain systems. DynIbex [Alexandre dit Sandretto and Chapoutot, 2016a] is able to compute the guaranteed integration of ODEs and can control, using tools provided by the Ibex framework, that the solutions stay inside a given box. SpaceEx [Frehse et al., 2011] implements several methods in order to compute the reachable space for linear hybrid systems. Also, the reachable space for non-linear hybrid systems can be computed with Flow* [Chen et al., 2012]. In this chapter an alternative method is presented, which is based on interval and tube arithmetics, in order to compute the reachable space for peculiar hybrid systems that are considered in this thesis: piecewise-ODEs.

In the next paragraphs, many metrics and notations are defined in order to simplify the analysis of p-ODE systems, solutions, theorems and proofs of guaranteed integration.

5.2.1.4 Differential discontinuity in solution of p-ODE

Let $\mathbf{u}^*$ a solution of p-ODEs system. $\mathbf{u}^*$ is a continuous function, but the differential $\mathbf{u}^{*\prime}$ of $\mathbf{u}^*$ is not necessarily continuous if several dynamics are involved on $[t_0; t_f]$. Let $\mathbf{d_{p_1}}$, $\mathbf{d_{p_2}}$ two of the dynamics involved by $\mathbf{u}^*$ and t_j a transition time-step such that at $t_j - \varepsilon$ the first dynamic $\mathbf{d_{p_1}}$ is considered Eq. (5.2), and then at $t_j + \varepsilon$ the second one is used Eq. (5.3).

$$\mathbf{u}^{*\prime}(t_j - \varepsilon) =_{\varepsilon \to 0} \mathbf{d_{p_1}}(t_j - \varepsilon, \mathbf{u}^*(t_j - \varepsilon)) \tag{5.2}$$

$$\mathbf{u}^{*\prime}(t_j + \varepsilon) =_{\varepsilon \to 0} \mathbf{d_{p_2}}(t_j + \varepsilon, \mathbf{u}^*(t_j + \varepsilon)) \tag{5.3}$$

As a consequence, if the values of the functions $\mathbf{d_{p_1}}$ and $\mathbf{d_{p_2}}$ are not equal when ε tends to 0

$$\mathbf{d_{p_1}}(t_j - \varepsilon, \mathbf{u}^*(t_j - \varepsilon)) \neq_{\epsilon \to 0} \mathbf{d_{p_2}}(t_j + \varepsilon, \mathbf{u}^*(t_j + \varepsilon))$$

then the differential of $\mathbf{u}^*$ is not continuous at t_j

$$\mathbf{u}^{*\prime}(t_j - \varepsilon) \neq_{\varepsilon \to 0} \mathbf{u}^{*\prime}(t_j + \varepsilon)$$

[1]The reachability problem which is a central problem in hybrid-system verification is to decide if there exists an execution of the automata from a given initial location l_0 to a given final location l_f [Raskin, 2011].

Nevertheless, the left differential that leads to t_j ($\mathbf{d_{p_1}}$) and the right differential which starts from t_j ($\mathbf{d_{p_2}}$) are well known. Therefore the differential $\mathbf{d_{p_1}}$ (resp. $\mathbf{d_{p_2}}$) can still be used to integrate $\mathbf{u}^*$ to t_j (resp. from t_j) using the continuity property of $\mathbf{u}^*$.

$$\begin{aligned}&\lim_{\varepsilon\to 0} \mathbf{u}^*(t_j-\varepsilon)+\varepsilon\times\mathbf{d_{p_1}}(t_j-\varepsilon,\mathbf{u}^*(t_j-\varepsilon))\\ &=\lim_{\varepsilon\to 0} \mathbf{u}^*(t_j+\varepsilon)-\varepsilon\times\mathbf{d_{p_2}}(t_j+\varepsilon,\mathbf{u}^*(t_j+\varepsilon))\\ &=\mathbf{u}^*(t_j)\end{aligned} \tag{5.4}$$

5.2.1.5 Definition : Validity Domain of Dynamics

In case of p-ODEs, the validity domain of each piece p from 1 to q is noted $\mathcal{V}_p$. The set $\mathcal{V}_p \subseteq \mathbb{R}\times\mathcal{U}$ is defined by the guard c_p that is linked with the dynamic $\mathbf{d_p}$.

$$\mathcal{V}_p = \{(t,\mathbf{u}) \in \mathcal{T}_p \times \mathcal{U} \mid c_p(t,\mathbf{u}(t))\} \tag{5.5}$$

where $\mathcal{T}_p$ is the validity time domain of the piece p. The set of functions $\mathbf{u}\in\mathcal{U}$ such as $\mathbf{u}(t)$ is within the boundaries of the validity domain of the piece p is noted $\mathcal{V}_p^t$ ($t\in\mathcal{T}_p$). In that sense, the guard that is associated with the piece p on the function $\mathbf{u}$ at time-step t is active.

$$\mathcal{V}_p^t = \{\mathbf{u}\in\mathcal{U} \mid (t,\mathbf{u})\in\mathcal{V}_p\}$$

This notation is extended with quantifiers and interval time-steps so as to get $\mathcal{V}_p^{\exists\mathcal{T}}$ and $\mathcal{V}_p^{\forall\mathcal{T}}$ with $\mathcal{T}\subseteq\mathcal{T}_p$. It defines two sets of unary-functions $\mathbf{u}\in\mathcal{U}$.

$$\begin{aligned}\mathcal{V}_p^{\exists\mathcal{T}} &= \{\mathbf{u}\in\mathcal{U} \mid \exists t\in\mathcal{T},\ (t,\mathbf{u})\in\mathcal{V}_p\} = \bigcup_{t\in\mathcal{T}} \mathcal{V}_p^t\\ \mathcal{V}_p^{\forall\mathcal{T}} &= \{\mathbf{u}\in\mathcal{U} \mid \forall t\in\mathcal{T},\ (t,\mathbf{u})\in\mathcal{V}_p\} = \bigcap_{t\in\mathcal{T}} \mathcal{V}_p^t\end{aligned}$$

Choice is made that the evaluation of the validity domain $\mathcal{V}_p$ on a piece p at a time-step $t\in\mathcal{T}_p$ and a set of time-steps $\mathcal{T}\subseteq\mathcal{T}_p$ are defined as below.

$$\mathcal{V}_p(t) = \bigcup_{\mathbf{u}\in\mathcal{V}_p^t} \mathbf{u}(t) \qquad \text{and then} \qquad \mathcal{V}_p(\mathcal{T}) = \bigcup_{t\in\mathcal{T}} \mathcal{V}_p(t) \tag{5.6}$$

On Figure 5.1 several curves are drawn, they represent unary functions from $\mathcal{U}$. The blue area at the top of the figure defines the validity domain $\mathcal{V}_1$ whereas its complement is the validity domain $\mathcal{V}_0$. The gray vertical domain represents a set of time-steps $\mathcal{T}$. According to this domain, the two functions $\mathbf{u_1}$ and $\mathbf{u_2}$ (green curves at the top) are elements of the set $\mathcal{V}_1^{\forall\mathcal{T}}$, the function $\mathbf{u_3}$ (black curve on the middle) is an element of the set $\mathcal{V}_1^{\exists\mathcal{T}}$ and the solution $\mathbf{u_4}$ (red curve at the bottom) is out of the validity domain $\mathcal{V}_1$ over $\mathcal{T}$.

5.2.1.6 Remark

Let $\mathbf{u}^*$ a solution of a p-ODE system, and $\mathcal{T}_p$ the set of values such that $\mathbf{u}^*\in\mathcal{V}_p^{\forall\mathcal{T}_p}$, then

$$\forall t\in\mathcal{T}_p,\ \mathbf{u}^*(t)\in\mathcal{V}_p(t) \tag{5.7}$$

and for all $t\in\mathcal{T}_p$, the solution $\mathbf{u}^*$ is driven by the dynamic $\mathbf{d_p}$.

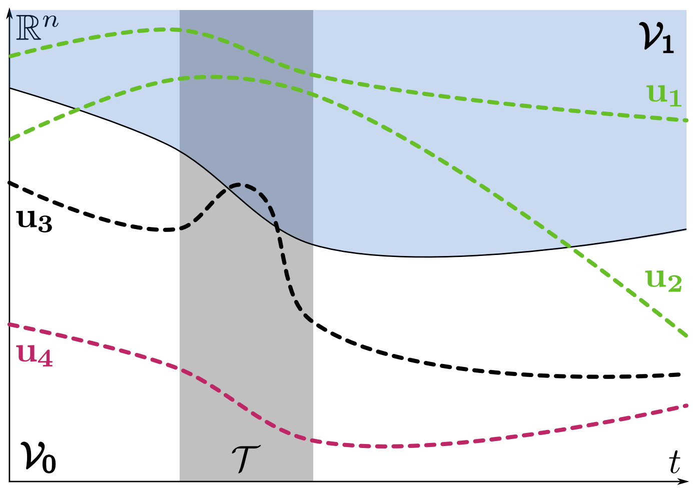


Figure 5.1: Validity domains $\mathcal{V}_0$ and $\mathcal{V}_1$ of dynamics

5.2.1.7 Definition : Pieces used by a solution

A solution $\mathbf{u}^*$ of the p-ODE uses one and only one dynamic at a time-step t. The dynamic that is used respects the validity domains. The main difficulty is to deal with non continuous states and to guarantee the numeric computation with tubes and intervals. This is why the times $t-\varepsilon$ and $t+\varepsilon$ around t are considered. Let $\mathbf{u}^*$ a solution of a p-ODE system, then the piece used by $\mathbf{u}^*$ at time t to reach $\mathbf{u}^*(t-\varepsilon)$ (resp. $\mathbf{u}^*(t+\varepsilon)$) is noted $\mathcal{B}^{t-}_{\mathbf{u}^*}$ (resp. $\mathcal{B}^{t+}_{\mathbf{u}^*}$), with $\varepsilon > 0$, $\varepsilon \to 0$:

$$\mathcal{B}^{t_i-}_{\mathbf{u}^*} = p \in \{1 \dots q\} \textit{ such that } \begin{cases} \mathbf{v}(t_i) &= \mathbf{u}^*(t_i) & \\ \mathbf{v}(t_i-\varepsilon) &= \mathbf{u}^*(t_i-\varepsilon) & \varepsilon>0,\ \varepsilon\to 0 \\ \mathbf{v}'(t) &= \mathbf{d_p}(t,\mathbf{v}(t)) & \forall t\in[t_i-\varepsilon;t_i] \\ \mathbf{u}^* &\in \mathcal{V}_p^{\forall[t_i-\varepsilon;t_i]} & \end{cases} \tag{5.8}$$

$$\mathcal{B}^{t_i+}_{\mathbf{u}^*} = p \in \{1 \dots q\} \textit{ such that } \begin{cases} \mathbf{v}(t_i) &= \mathbf{u}^*(t_i) & \\ \mathbf{v}(t_i+\varepsilon) &= \mathbf{u}^*(t_i+\varepsilon) & \varepsilon>0,\ \varepsilon\to 0 \\ \mathbf{v}'(t) &= \mathbf{d_p}(t,\mathbf{v}(t)) & \forall t\in[t_i;t_i+\varepsilon] \\ \mathbf{u}^* &\in \mathcal{V}_p^{\forall[t_i;t_i+\varepsilon]} & \end{cases} \tag{5.9}$$

By extension, the set of pieces used by $\mathbf{u}^*$ on the interval $[t_1;t_2]$ defines $\mathcal{B}^{[t_1;t_2]}_{\mathbf{u}^*}$:

$$\mathcal{B}^{[t_1;t_2]}_{\mathbf{u}^*} = \bigcup_{t\in[t_1;t_2[} \mathcal{B}^{t+}_{\mathbf{u}^*} = \bigcup_{t\in]t_1;t_2]} \mathcal{B}^{t-}_{\mathbf{u}^*} \tag{5.10}$$

The obstacle is to guarantee the integration process of the solution $\mathbf{u}^*$ on time windows $[t_1;t_2]$ such that $\mathcal{B}^{[t_1;t_2]}_{\mathbf{u}^*}$ contains several pieces.

5.2.1.8 Definition : ($\mathcal{E}$-)Boundary of validity domain

For p-ODEs, $\mathcal{F}^{\varepsilon}_p$ defines the ε-boundary of the validity domain $\mathcal{V}_p$ for each dynamic $\mathbf{d_p}$.

The set $\mathcal{F}_p^\varepsilon \subseteq \mathcal{V}_p$ is defined as below :

$$\mathcal{F}_p^\varepsilon = \left\{ (t,\mathbf{u}) \in \mathcal{V}_p \;\middle|\; \exists \mathbf{k} \in \mathbb{R}^{n+1}, \begin{array}{c} |\mathbf{k}| < \varepsilon \\ (t,\mathbf{u}) + \mathbf{k} \notin \mathcal{V}_p \end{array} \right\} \tag{5.11}$$

When ε is close to 0, the ε-boundary $\mathcal{F}_p^\varepsilon$ defines the boundary $\mathcal{F}_p$ of $\mathcal{V}_p$.

$$\mathcal{F}_p^\varepsilon =_{\varepsilon \to 0} \mathcal{F}_p \tag{5.12}$$

5.2.1.9 Definition : Sets of Transition Times and States

Let p_1 and p_2 two different pieces from the p-ODE $\boldsymbol{\Theta}$. The set of transition times between several pieces is noted $\mathcal{T}_\nabla$ (and $\mathcal{T}_{p_1 \nabla p_2}$ between p_1 and p_2). Similarly, the set of transition states is noted $\mathcal{U}_\nabla$ (and $\mathcal{U}_{p_1 \nabla p_2}$).

$$\mathcal{U}_\nabla = \{\mathbf{u} \in \mathcal{U} \mid \exists t \in \mathcal{T}_\nabla,\ (\mathbf{u},t) \in \mathcal{V}_{p_1} \cap \mathcal{V}_{p_2}\} \tag{5.13}$$

$$\mathcal{T}_\nabla = \{t \in \mathcal{T}_{p_1} \cap \mathcal{T}_{p_2} \mid \exists u \in \mathcal{U}_\nabla,\ (\mathbf{u},t) \in \mathcal{V}_{p_1} \cap \mathcal{V}_{p_2}\} \tag{5.14}$$

5.2.1.10 Definition : Guards Disjunction (Partial and Total)

Let Φ a piecewise ODE. The total disjunction between the two dynamics $\mathbf{d_i}$, $\mathbf{d_j}$ is qualified by the empty intersection of their validity domains $\mathcal{V}_i$ with $\mathcal{V}_j$.

$$\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$$

The domains are totally disjointed: the dynamic from one domain cannot reach the other domain. The guaranteed integration method previously described can be used separately on each dynamic. The partial disjunction has more subtlety to it because the dynamics must share a common set of validity domains. Then the intersection of $\mathcal{V}_i$ with $\mathcal{V}_j$ exists and is contained within the boundaries of the validity domains $\mathcal{F}_i$ and $\mathcal{F}_j$ Eq. (5.15).

$$\mathcal{V}_i \cap \mathcal{V}_j \subseteq \mathcal{F}_i \cap \mathcal{F}_j \tag{5.15}$$

These definitions can be extended to sets of multiple dynamics when applied on all distinct pairs of dynamics.

On the opposite, guards can intersect each other inside a volume that is larger than their respective frontiers. For example, let Φ the following piecewise ODEs system Eq. (5.16) with two dynamics $\mathbf{d_1}$, $\mathbf{d_2}$:

$$\mathbf{u}'(t) = \begin{cases} \mathbf{d_1}(t,\mathbf{u}(t)) & \text{if } \mathbf{u}(t) \leq 10 \\ \mathbf{d_2}(t,\mathbf{u}(t)) & \text{if } 0 \leq \mathbf{u}(t) \end{cases} \tag{5.16}$$

Then a state set of the system can induce more than one dynamic because of the non-disjointed guards. The set of pairs $(t,\mathbf{u}(t))$ such that $0 \leq \mathbf{u}(t) \leq 10$ represents all the states thus making both of the dynamics valid simultaneously while the respective frontiers are $\mathbf{u}(t) = 10$ for the first piece and $\mathbf{u}(t) = 0$ for the second one.

These intersections on guards are critical points since the solution goes from one dynamic to another. Sometimes the breaking point is uncertain and many dynamics may therefore be valid in theory (although in reality only one dynamic at a time is valid). The behavior of the dynamics is important and many situations may happen. This situation is used to model physic states whose limits are known with a limited accuracy. For example the limit between the atmosphere and the empty space is not clearly defined, also the limit between the hydrosphere and the atmosphere is moving in many areas because of waves and is unpredictable. However these environments can be bounded: at a level of 15 meters above the average level of the ocean it is the atmosphere, on the contrary at a level below -15 meters it is the hydrosphere, between these two extrema the environment is uncertain and both environments have to be considered in order to guarantee the position of an object that falls down into the ocean.

5.2.1.11 Definition : Cooperation of the dynamics

The term cooperative is used to describe a consistent global behavior that is shared by many dynamics of a piecewise ODE Φ, over a period of time $\mathcal{T}_\triangledown$ and a set of solutions $\mathcal{U}_\triangledown \subseteq \mathcal{U}$. Let $\mathcal{H}_o$ a subset of dynamics extracted from the piecewise ODE Φ and $\mathbf{d_i}$ another dynamic from Φ. The set $\mathcal{H}_o$ is cooperative with $\mathbf{d_i}$ on $\mathcal{T}_\triangledown \times \mathcal{U}_\triangledown$ if and only if:

- All the dynamics $\mathbf{d_j} \in \mathcal{H}_o$ are purgative, that is to say they drive all the solutions $\mathbf{u} \in \mathcal{U}_\triangledown$ out of their domains $\mathcal{V}_j$ over $t \in \mathcal{T}_\triangledown$.

$$\forall (t_1, \mathbf{u}) \in \mathcal{V}_j \cap (\mathcal{T}_\triangledown \times \mathcal{U}_\triangledown),\ \exists (t_2, \mathbf{v}) \in \mathcal{T}_\triangledown \times \mathcal{U},\ \left[\begin{array}{rrcl} & t_1 & \leq & t_2 \\ \wedge & (t_2, \mathbf{v}) & \notin & \mathcal{V}_j \\ \wedge & \mathbf{v}'(t) & = & \mathbf{d_j}(t, \mathbf{v}(t)) \\ \wedge & \mathbf{v}(t_1) & = & \mathbf{u}(t_1) \end{array}\right] \quad (5.17)$$

- The dynamic $\mathbf{d_i}$ is conservative and keeps all the solutions $\mathbf{u} \in \mathcal{U}_\triangledown$ in its domain $\mathcal{V}_i$ over $t \in \mathcal{T}_\triangledown$.

$$\forall (t_1, \mathbf{u}) \in \mathcal{V}_i \cap (\mathcal{T}_\triangledown \times \mathcal{U}_\triangledown),\ \nexists (t_2, \mathbf{v}) \in \mathcal{T}_\triangledown \times \mathcal{U},\ \left[\begin{array}{rrcl} & t_1 & \leq & t_2 \\ \wedge & (t_2, \mathbf{v}) & \notin & \mathcal{V}_i \\ \wedge & \mathbf{v}'(t) & = & \mathbf{d_i}(t, \mathbf{v}(t)) \\ \wedge & \mathbf{v}(t_1) & = & \mathbf{u}(t_1) \end{array}\right] \quad (5.18)$$

5.2.1.12 Cooperative dynamics (Fig. 5.2):

Cooperative dynamics guarantee that the solutions $\mathbf{u}^*$ that are elements of $\mathcal{U}_\triangledown$ go from one dynamic to another on the time window $\mathcal{T}_\triangledown$. The complexity of this situation comes from the bounding of the time window $\mathcal{T}_\triangledown$ to get an upper bound of the time to end the integration with $\mathbf{d_1}$ and a lower bound of the time to begin the integration with $\mathbf{d_2}$.

5.2.1.13 Competitive dynamics – Conservative vs Purgative (Fig. 5.3):

Competitive dynamics can induce distinctive situations that depend on the conservative or the purgative behavior of these dynamics:

- When all the dynamics are conservative, the solutions from each piece stay inside the validity domain of this piece (Fig. 5.3a).

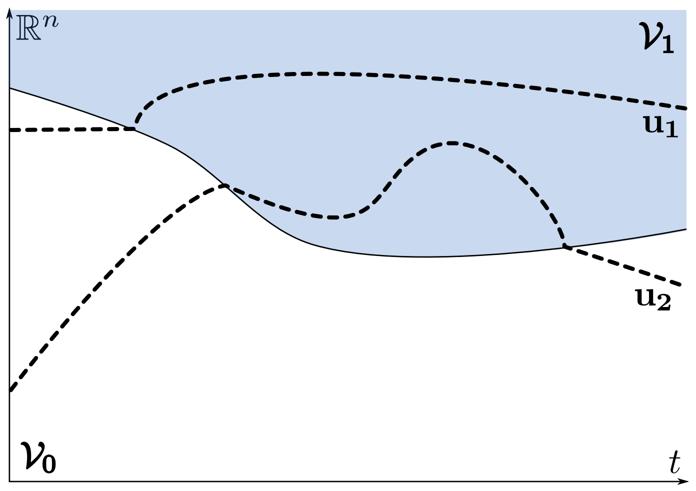
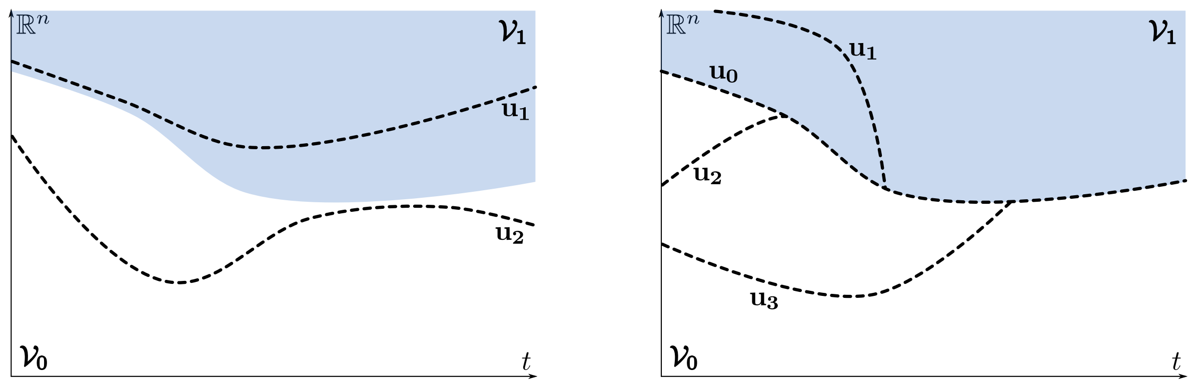


Figure 5.2: Cooperative dynamics

- When all the dynamics involved drive the solutions out of their domains (purgative), the solutions will follow the intersection between the guards in play (Fig. 5.3b).



(a) Conservative (b) Purgative

Figure 5.3: Competitive dynamics

As a consequence, the guaranteed integration can diverge largely depending on the set of solutions (Fig. 5.4) and because of the over-approximation when the enclosure is led to intersect several pieces. Starting from the set of transition times, more than one dynamic might have to be considered at each time-step. In order to get tinier enclosures when it happens, multi-tubes are used to bound solutions.

In the previous paragraphs several cases (cooperative and competitive) have been developed. On these examples the domains $\mathcal{V}_1$ and $\mathcal{V}_2$ are partially disjointed, meaning that the intersection between the domains is contained inside the intersection of their frontiers $\mathcal{F}_1 \cap \mathcal{F}_2$.

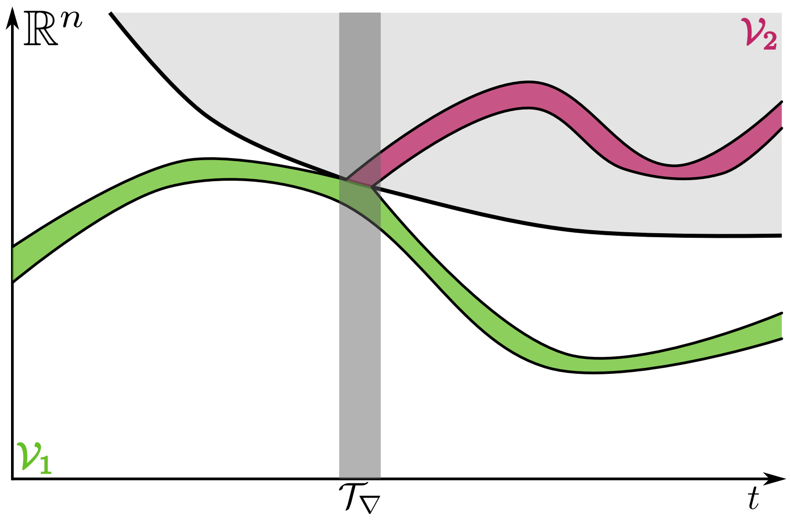


Figure 5.4: Conservative dynamics on a set of solutions

A more complex case has to be considered: the existence of non-disjointed guards when the domains $\mathcal{V}_1$ and $\mathcal{V}_2$ intersect one another outside of their frontiers $\mathcal{F}_1$ and $\mathcal{F}_2$ (Fig. 5.5). Although it is mathematically impossible for a state $\mathbf{u}^*(t) \in [\mathbf{u}](t)$ to have multiple differential values at a given time-step t in $\mathcal{T}_\nabla$, in such a situation it is necessary to take into account all the plausible dynamics in order to guarantee that the enclosure computed encloses all the solutions.

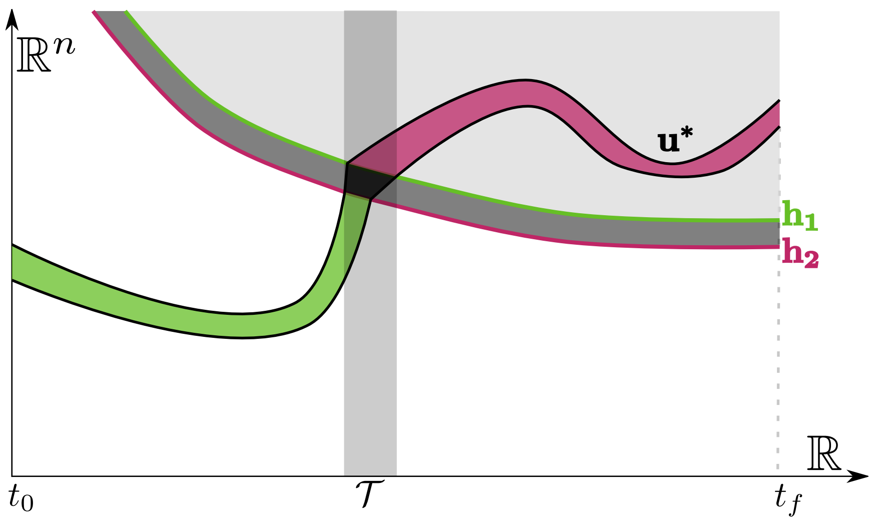


Figure 5.5: Set of solutions of a p-ODE with non-disjointed guards in a cooperative situation

For example, when in a transition time window $\mathcal{T}_\nabla$ that implicates two dynamics $\mathbf{d_1}$ and $\mathbf{d_2}$, all of the possibilities have to be taken into account for all the couples of time-steps (t_1, t_2) in $\mathcal{T}_\nabla^2$:

- $\mathbf{d_1}$ at time $t_1 \in \mathcal{T}_\nabla$ and $t_2 \in \mathcal{T}_\nabla$

- $\mathbf{d_2}$ at time $t_1 \in \mathcal{T}_\nabla$ and $t_2 \in \mathcal{T}_\nabla$
- $\mathbf{d_1}$ at time $t_1 \in \mathcal{T}_\nabla$ and $\mathbf{d_2}$ at time $t_2 \in \mathcal{T}_\nabla$
- $\mathbf{d_2}$ at time $t_1 \in \mathcal{T}_\nabla$ and $\mathbf{d_1}$ at time $t_2 \in \mathcal{T}_\nabla$

In a situation such as this one, when the dynamics are cooperative (or competitive and purgative), the solutions remain enclosed in a relatively tiny enclosure that depends on the size of the intersection of the validity domains $\mathcal{V}_1 \cap \mathcal{V}_2$. On the opposite, the set of solutions becomes dramatically huge in case of competitive and conservative dynamics.

5.2.1.14 Reduction through non-piecewise ODE systems with additional constraints

This paragraph is a theoretical consideration about the characterization of an exact solution $\mathbf{u}^*$ on $[t_i, t_{i+1}]$ of a partially disjointed n-dimensional first-order p-ODE system $\boldsymbol{\Theta}$ defined by q pieces Eq. (5.19) from a single initial value Eq. (5.20):

$$\forall t \in [t_i; t_{i+1}],\ \mathbf{u}'(t) = \begin{cases} \mathbf{d_1}(t, \mathbf{u}(t)) & \text{if} \quad c_1(t, \mathbf{u}(t)) \\ & \vdots \\ \mathbf{d_q}(t, \mathbf{u}(t)) & \text{if} \quad c_q(t, \mathbf{u}(t)) \end{cases} \tag{5.19}$$

$$\mathbf{u}(t_i) \in [\mathbf{u_i}] \tag{5.20}$$

where for each piece p, $\mathbf{d_p}$ is continuous and has first order partial derivatives on $[t_i; t_{i+1}]$. If a piece p exists such that the solution $\mathbf{u}^*$ is exclusively driven by the dynamic $\mathbf{d_p}$ from t_i to $t_\nabla \in [t_i; t_{i+1}]$, then $\mathbf{u}^*$ is also a solution of the classic ODE system:

$$\forall t \in [t_0; t_f],\ \mathbf{u}'(t) = \mathbf{d_p}(t, \mathbf{u}(t)) \tag{5.21}$$

$$\mathbf{u}(t_0) \in [\mathbf{u_0}] \tag{5.22}$$

with the additional constraint:

$$t_0 = t_i \ \wedge\ t_f = t_\nabla \ \wedge\ [\mathbf{u_0}] = [\mathbf{u_i}] \ \wedge\ \forall t \in [t_0; t_f],\ c_p(t, \mathbf{u}(t)) \tag{5.23}$$

The solution $\mathbf{u}^*$ of the p-ODE from t_0 to t_f can then be considered as a succession of solutions of several ODEs. The guaranteed integration of the IVP Eq. (5.21) and (5.22) with these additional constraints Eq. (5.23) is also a part of the guaranteed integration of the p-ODE IVP Eq. (5.19) and (5.20) because uncertainty reside in the value of t_∇ and t_i when it comes from a previous dynamic.

In this section piecewise-ODE and some metrics (validity domains $\mathcal{V}$, frontiers $\mathcal{F}$, cooperative dynamics...) were introduced. The guaranteed integration process on such dynamic systems is described in the next sections. The first section will consider *Uncertain Time Initial Value Problem* on ODE (ODE-UTIVP), a more global ODE-IVP system (with a single piece, not a piecewise ODE) with a variation about the initial value and a resolution strategy that consists in a reduction towards a classic ODE-IVP system. Then, this reduction will be used in the resolution process of piecewise-ODE.

5.3 Uncertain Time Initial Value Problems

This section is dedicated to the definition and the resolution of more generic cases than IVPs previously introduced. ODE systems are considered, with an initial value which is contained in an interval. The specificity of this problem is that the initial time on which the initial value is valid is also uncertain and is contained in an interval. In the context of this thesis the main advantage of this model is that it can be used to solve piecewise-ODEs. Moreover, this model still possesses some other positive aspects, such as its robustness when it comes to integrate ODEs with guarantees - an approximation over the time-step of the initial value thus being a loss of guarantee. Consequently, an interval value to enclose the initial time-step is an efficient way to get the guarantee back.

5.3.1 Integration

5.3.1.1 Definition : Uncertain Time Initial Value Problem

The initial value problem Eq. (3.7) is extended with uncertain time over the initial values enclosure Eq. (5.25). The system is modeled as below:

$$\forall t \in [t_0; t_f],\ \mathbf{u}'(t) = \mathbf{d}(t, \mathbf{u}(t)) \tag{5.24}$$

$$\exists t \in [t_0; t_k],\ \mathbf{u}(t) \in [\mathbf{u_0}] \tag{5.25}$$

where $t_0 \le t_k \le t_f$. Then the initial value $\mathbf{u_0} \in [\mathbf{u_0}]$ exists at a time-step $t \in [t_0; t_k]$. Note that many pairs $(t, \mathbf{u_0})$ are equivalent and lead to identical solutions $\mathbf{u}^*$.

The resolution process (guaranteed integration) of the ODE-UTIVPs is based on the resolution of the ODEs as previously detailed. The trick is to build an ODE-IVP from the ODE-UTIVP. The next paragraphs describe the reduction method.

5.3.1.2 Reduction from the Uncertain Time IVPs to the IVPs

In order to get an IVP from an uncertain time IVP, one should first focus on the peculiar case of UTIVP in which $t_f = t_k$.

$$\forall t \in [t_0; t_k],\ \mathbf{u}'(t) = \mathbf{d}(t, \mathbf{u}(t)) \tag{5.26}$$

$$\exists t \in [t_0; t_k],\ \mathbf{u}(t) \in [\mathbf{u_0}] \tag{5.27}$$

Theorem 5.3.1 (Extension of the Picard operator)
Let $\mathbf{u}^*$ a solution of the system and t_i in $[t_0; t_k]$ such that $\mathbf{u}^*(t_i) \in [\mathbf{u_0}]$. Let $[\tilde{\mathbf{u}}_\mathbf{i}]^+$ an interval vector such that $[\mathbf{u_0}] \subseteq [\tilde{\mathbf{u}}_\mathbf{i}]^+$. A new Picard operator is defined where the time-step bounds are larger than the original ones.

$$\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+) = [\mathbf{u_0}] + [0; t_k -^{\uparrow} t_0][\mathbf{d}]([t_0; t_k], [\tilde{\mathbf{u}}_\mathbf{i}]^+) \tag{5.28}$$

The operator that is defined this way has the same properties than the original one. If $\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+) \subseteq [\tilde{\mathbf{u}}_\mathbf{i}]^+$ then

1. *The system $\mathbf{\Theta}$ with the initial value $\mathbf{u}(t_i) \in [\mathbf{u_0}]$ has a unique solution $\mathbf{u}^*$ on $[t_i, t_k]$.*

2. The box $\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}])$ is an enclosure of $\mathbf{u}^*$ on $[t_i, t_k]$ with respect to $\mathbf{u}^*(t_i) \in [\mathbf{u_0}]$.

The application of the operator $\mathbf{\Phi}^+$ allows to build and find an enclosure of the solution $\mathbf{u}^*$ on $[t_i; t_k]$ for all t_i in $[t_0; t_k]$.

5.3.1.3 Proof

Using the classic Picard operator $\mathbf{\Phi}$ that is defined in Chapter 3,

$$\mathbf{\Phi}([\tilde{\mathbf{u}}_\mathbf{i}]^+) = [\mathbf{u_0}] + [0; t_k -^{\uparrow} t_i][\mathbf{d}]([t_i; t_k], [\tilde{\mathbf{u}}_\mathbf{i}]^+)$$

it is known that if $\mathbf{\Phi}([\tilde{\mathbf{u}}_\mathbf{i}]^+) \subseteq [\tilde{\mathbf{u}}_\mathbf{i}]^+$ then $\mathbf{\Phi}([\tilde{\mathbf{u}}_\mathbf{i}]^+)$ is an enclosure of $\mathbf{u}^*$ on $[t_i; t_k]$. Using the interval inclusion Eq. (5.29) and (5.30) and the monotonicity inclusion properties of interval arithmetic, it can be proven that $\mathbf{\Phi}([\tilde{\mathbf{u}}_\mathbf{i}]^+) \subseteq \mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+)$:

$$[0; t_k -^{\uparrow} t_i] \subseteq [0; t_k -^{\uparrow} t_0] \tag{5.29}$$

$$[t_i; t_k] \subseteq [t_0; t_k] \tag{5.30}$$

$$\mathbf{\Phi}([\tilde{\mathbf{u}}_\mathbf{i}]^+) \subseteq [\mathbf{u_k}] + [0; t_k -^{\uparrow} t_0][\mathbf{d}]([t_0; t_k], [\tilde{\mathbf{u}}_\mathbf{i}]^+) = \mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+) \tag{5.31}$$

If an interval vector $[\tilde{\mathbf{u}}_\mathbf{i}]^+$ is found such that $\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+) \subseteq [\tilde{\mathbf{u}}_\mathbf{i}]^+$, then $\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+)$ is an enclosure of $\mathbf{\Phi}([\tilde{\mathbf{u}}_\mathbf{i}]^+)$ which is an enclosure of $\mathbf{u}^*$ on $[t_i; t_k]$.

5.3.1.4 Generalization

Although $t_k \neq t_f$ in the general case, an enclosure of the solution can be integrated from the restriction to $t_f = t_k$. Indeed, the enclosure $\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+)$ of the solution can be computed from any t_i in $[t_0; t_k]$ to t_k with the operator $\mathbf{\Phi}^+$. This enclosure guarantees that the solution is enclosed in $\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{i}]^+)$ at t_k. Thus, the classic IVP defined as below can be solved.

$$\forall t \in [t_k; t_f],\ \mathbf{u}'(t) = \mathbf{d}(t, \mathbf{u}(t)) \tag{5.32}$$

$$\mathbf{u}(t_k) \in [\tilde{\mathbf{u}}_\mathbf{i}]^+ \tag{5.33}$$

The difficulty is that the time-step $[t_0; t_k]$ can be large and the stepsize should not be decreased in order to guarantee the enclosure. Therefore the enclosure $\mathbf{\Phi}^+([\tilde{\mathbf{u}}_\mathbf{k}]^+)$ cannot be computed or, when computed, is irrelevant and should not be used as an initial value at t_k to define the classic IVP. In order to face this issue, one could divide the time window $[t_0; t_k]$ into a set of intervals $[t_{k_1}] \dots [t_{k_n}]$ with $\underline{t_{k_1}} = t_0$, $\overline{t_{k_i}} = \underline{t_{k_{i+1}}}$ and $\overline{t_{k_n}} = t_f$ that defines several UTIVPs. They can be solved separately and then merged into a single enclosure using the classic integration scheme from each $\overline{t_{k_i}}$ to t_f.

In this section, a method to compute the guaranteed forward integration of UTIVPs has been detailed. Similarly, in case of backward integration, the Picard operator $\mathbf{\Phi}^-$ can be defined.

5.4 Guaranteed Integration of Piecewise ODEs

Piecewise ODEs can be solved thanks to the previous results. Indeed the guaranteed enclosure of the solution is reached when all the possible dynamics are visited. However, in the case where a single piece is used, the resolution of a p-ODE can be reduced to the guaranteed integration of an ODE. Otherwise, when more than one piece are involved, the resolution is more complex because there are transitions from one dynamic to another. When situated outside of these transitions areas, the resolution of a p-ODE is equivalent to the integration of a classic ODE. Therefore, what matters the most in this context and that will be dealt with in the following paragraphs is the resolution of the p-ODEs through the specific treatment of the transition areas.

5.4.1 The relationship between ODE, UTIVP and p-ODE

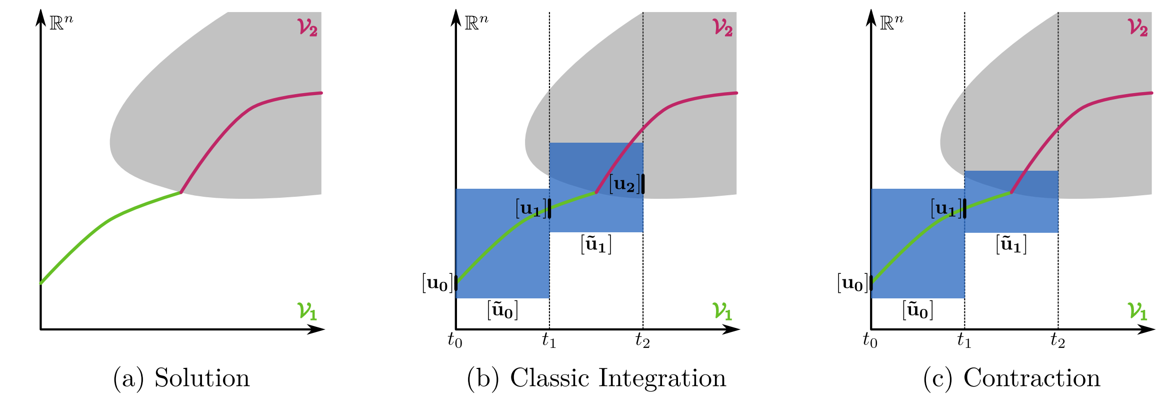
The relation resides in the strategy that is used to solve p-ODEs. Solving the p-ODEs begins with the resolution of the ODEs that correspond to the pieces that were enabled by the initial value. For each one of those pieces, as long as their dynamics are enabled by the guard they are associated with, the integration process goes on. At the end of the resolution of each piece, the intersections between the solutions and the other domains are computed. The results of the non-empty intersections have to be considered as initial values to the other dynamics involved. These new initial values are uncertain time initial values.

On Figure 5.6a, the solution of a p-ODE is driven by the two dynamics $\mathbf{d_1}$ and $\mathbf{d_2}$, with the respective validity domains $\mathcal{V}_1$ and $\mathcal{V}_2$.

The enclosure computed exclusively using the first branch $\mathbf{d_1}$ is not guaranteed to be a safe enclosure of the solution of the p-ODE on $[t_0; t_f]$ (Fig. 5.6b). However, the global enclosure $[\tilde{\mathbf{u}}_\mathbf{0}]$ is a safe enclosure of the solution as far as the first dynamic on $[t_0; t_1]$, $[\mathbf{u_0}]$ at t_0 and $[\mathbf{u_1}]$ at t_1 is considered. On the contrary, the solution is clearly not enclosed in $[\tilde{\mathbf{u}}_\mathbf{1}]$ on the second time window $[t_1; t_2]$ and only a part of the solution (the one using the first dynamic $\mathbf{d_1}$) is guaranteed to be enclosed on the interval $[t_1; t_\bigtriangledown]$ where $t_\bigtriangledown$ is the transition time. It can be noted that although the second part that is driven by $\mathbf{d_2}$ is partially enclosed, the enclosure is not guaranteed and the enclosure $[\mathbf{u_2}]$ at t_2 does not contain the solution.

The main difficulties lie in knowing how to compute the crossing time of the boundary and in knowing which new initial value must be used in order to compute a second enclosure of the solution driven by the second dynamic $\mathbf{d_2}$. The example that is developed, is deliberately easy because it has only one crossing time (and not a set of those) and a punctual value $\mathbf{u}^*(t_\bigtriangledown)$. However simple it may look it has no effect on the complexity of the solving process since tubes are used to enclose the solution.

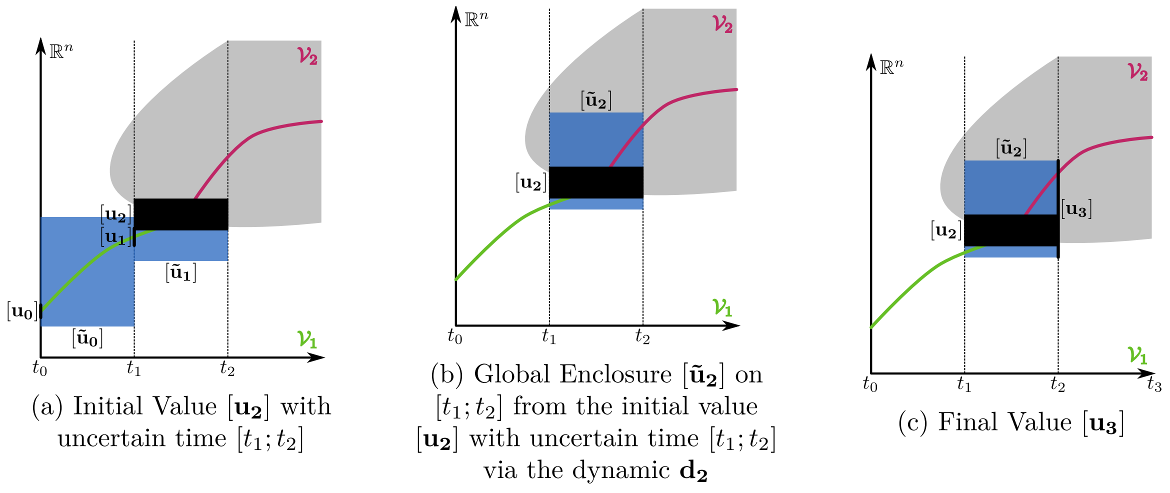
The solution developed in this thesis is to consider the enclosure $[\tilde{\mathbf{u}}_\mathbf{1}]$ as an uncertain time initial value $[\mathbf{u_2}]$ that would initiate the integration with the second dynamic $\mathbf{d_2}$ (Fig. 5.7a). From this UTIVP, a new bounding box can be computed using the second dynamic $\mathbf{d_2}$ over the time window $[t_1; t_2]$ (Fig. 5.7b). A new and classic initial value problem is built using the bounding box $[\tilde{\mathbf{u}}_\mathbf{2}]$ as the initial value $[\mathbf{u_3}]$ at time-step t_2 (Fig.



(a) Solution (b) Classic Integration (c) Contraction

Figure 5.6: integration of a piecewise ODE - classic integration

5.7c). Now the guaranteed integration method that was described previously can be used from $(t_2, [\mathbf{u_3}])$ to t_3.



(a) Initial Value $[\mathbf{u_2}]$ with uncertain time $[t_1; t_2]$

(b) Global Enclosure $[\tilde{\mathbf{u}}_\mathbf{2}]$ on $[t_1; t_2]$ from the initial value $[\mathbf{u_2}]$ with uncertain time $[t_1; t_2]$ via the dynamic $\mathbf{d_2}$

(c) Final Value $[\mathbf{u_3}]$

Figure 5.7: computing the new initial value

5.4.2 Issues

The integration process that was developed seems to be sufficient enough to compute a guaranteed integration of piecewise-ODEs. However, it appears that the process is not reliable enough. Indeed, as depicted on the set of Figures 5.7, a piece of the uncertain time initial value $[\mathbf{u_2}]$, as well as a piece of the global bounding box $[\tilde{\mathbf{u}}_\mathbf{2}]$ and of the initial value $[\mathbf{u_3}]$ intersect the domain $\mathcal{V}_1$. Moreover, this method does not guarantee that the solution does not return from the second dynamic $\mathbf{d_2}$ to the first one $\mathbf{d_1}$. This critical issue underlines how necessary the following theorem is; It guarantees the enclosure of the solution along the integration even when the solutions go back and forth over several dynamics.

The theorem requires to build several uncertain time initial values, one for each piece that is involved in the transition. All these pieces must meet specific requirements over these uncertain time initial values and the results of the Picard operator each piece is associated with.

Theorem 5.4.1 (Global enclosure for p-ODEs on transition time windows) *Let $\boldsymbol{\Theta}$ a n-dimensional first-order p-ODE system defined by q pieces where for each piece p, $\mathbf{d_p}$ is continuous and has first order partial derivatives on $[t_i] = [\underline{t_i}; \overline{t_i}]$. Let $[\mathbf{u_i}]$ an enclosure of the initial value of the solution $\mathbf{u}^*$ on the same uncertain interval $[t_i]$.*

$$\mathbf{u}'(t) = \begin{cases} \mathbf{d_1}(t, \mathbf{u}(t)) & \textit{if} \quad c_1(t, \mathbf{u}(t)) \\ & \vdots \\ \mathbf{d_q}(t, \mathbf{u}(t)) & \textit{if} \quad c_q(t, \mathbf{u}(t)) \end{cases}$$
$$\exists t_i \in [t_i],\ \mathbf{u}^*(t_i) = \mathbf{u_i} \in [\mathbf{u_i}]$$

Let the set of initial values $\{[\mathbf{u_i}]_\mathbf{1} \dots [\mathbf{u_i}]_\mathbf{q}\}$ consistent with each piece p:

$$\forall p \in \{1 \dots q\},\ [\mathbf{u_i}]_\mathbf{p} = [\mathbf{u_i}] \cap \mathcal{V}_p([t_i]) \tag{5.34}$$

Let $\{[\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{1} \dots [\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{q}\}$ a set of q interval vectors. For each piece p, the Picard operator noted Φ^+_p is defined as:

$$\Phi^+_p([\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p}) = \Big([\mathbf{u_i}]_\mathbf{p} + [0; \overline{t_i} - \underline{t_i}][\mathbf{d_p}]([t_i], [\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p})\Big) \cap \mathcal{V}_p([t_i]) \tag{5.35}$$

where $[\mathbf{d_p}]$ is an interval extension of $\mathbf{d_p}$.

If the following conditions are satisfied

$$\forall p \in \{1 \dots q\},\ \Phi^+_p([\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p}) \subseteq [\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p} \tag{5.36}$$
$$\forall (p_1, p_2)_{p_1 \neq p_2} \in \{1 \dots q\}^2,\ \Phi^+_{p_1}([\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p_1}) \cap \mathcal{V}_{p_2}([t_i]) \subseteq [\mathbf{u_i}]_\mathbf{p_2} \tag{5.37}$$

then $[\tilde{\mathbf{u}}_\mathbf{i}]^+$ is an enclosure of $\mathbf{u}^*$ on $[t_i; \overline{t_i}]$, provided the solution exists, where $[\tilde{\mathbf{u}}_\mathbf{i}]^+$ is defined as below.

$$[\tilde{\mathbf{u}}_\mathbf{i}]^+ = \bigcup_{p \in \{1 \dots q\}} \Phi^+_p([\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p}) \tag{5.38}$$

5.4.2.1 Proof:

Let t_i and t_j in $[t_i] = [\underline{t_i}; \overline{t_i}]$ with $t_i \leq t_j$. Let $\mathbf{u}^*$ a solution on $[t_i]$ of a p-ODE system $\boldsymbol{\Theta}$. From the proof 5.3.1.3, it is known that for all the pieces p, if

$$\mathbf{u}^*(t_i) \in [\mathbf{u_i}]_\mathbf{p} \tag{5.39}$$
$$\forall t \in [t_i; t_j[,\ \mathbf{u}^{*\prime}(t) = \mathbf{d_p}(t, \mathbf{u}^*(t)) \tag{5.40}$$

then $\Phi^+_p([\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p})$ encloses $\mathbf{u}^*$ on $[t_i; t_j]$. In particular $\Phi^+_p([\tilde{\mathbf{u}}_\mathbf{i}]_\mathbf{p})$ encloses $\mathbf{u}^*(t_j)$:

$$\mathbf{u}^*(t_j) \in \Phi^+_p([\tilde{\mathbf{u}}_\mathbf{i}]^+_\mathbf{p}) \tag{5.41}$$

Two cases have to be considered depending on the value of t_j, $t_j = \overline{t_i}$ and $t_j \leq \overline{t_i}$.

In the first case, when t_j is equal to $\overline{t_i}$ the solution $\mathbf{u}^*$ is enclosed by $\Phi_p^+([\mathbf{\tilde{u}_i}]_\mathbf{p}^+) \subseteq [\mathbf{\tilde{u}_i}]^+$ on $[t_i; \overline{t_i}]$.

In the second case, considering $t_j < \overline{t_i}$, let $\varepsilon > 0$, then at the time-step t_j the following property is achieved:

$$\mathbf{u}^{*\prime}(t_j - \varepsilon) =_{\varepsilon\to 0} \mathbf{d_p}(t_j, \mathbf{u}^*(t_j)) \neq_{\varepsilon\to 0} \mathbf{u}^{*\prime}(t_j + \varepsilon) \tag{5.42}$$

Since $\mathbf{u}^*$ is a solution on $[t_i]$, it exists on $[t_j; \overline{t_i}]$. Then a piece $r \neq p$ exists in the set $\{1 \ldots p-1, p+1 \ldots q\}$ with $(t_j, \mathbf{u}^*) \in \mathcal{V}_r$ such that

$$\mathbf{u}^{*\prime}(t_j + \varepsilon) =_{\varepsilon\to 0} \mathbf{d_r}(t_j, \mathbf{u}^*(t_j)) \tag{5.43}$$

It is known from $(t_j, \mathbf{u}^*) \in \mathcal{V}_r$ that $\mathbf{u}^*(t_j) \in \mathcal{V}_r^{t_j}(t_j) \subseteq \mathcal{V}_r^{[t_i]}([t_i])$ and from Eq. 5.41 that $\mathbf{u}^*(t_j) \in \Phi_p^+([\mathbf{\tilde{u}_i}]_\mathbf{p}^+)$. Ultimately, once the theorem condition Eq. (5.37) is applied on the pieces p and r it is known that

$$\Phi_p^+([\mathbf{\tilde{u}_i}]_\mathbf{p}^+) \cap \mathcal{V}_r^{[t_i]}([t_i]) \subseteq [\mathbf{u_i}]_\mathbf{r} \tag{5.44}$$

$$\mathbf{u}^*(t_j) \in [\mathbf{u_i}]_\mathbf{r} \tag{5.45}$$

Then $\mathbf{u}^*$ is enclosed by $[\mathbf{\tilde{u}_i}]^+$ on $[t_i; \overline{t_i}]$.

Similarly, following the same steps it can be proven that $\mathbf{u}^*$ is enclosed by $[\mathbf{\tilde{u}_i}]^-$ on the time windows $[\underline{t_i}; t_i]$.

5.4.3 Improvements

In order to integrate the output dynamics, the uncertain time initial value should be the tiniest one. Indeed, when it is not tiny enough then the enclosure could quickly explode and the final enclosure could be irrelevant. Several methods can be used together.

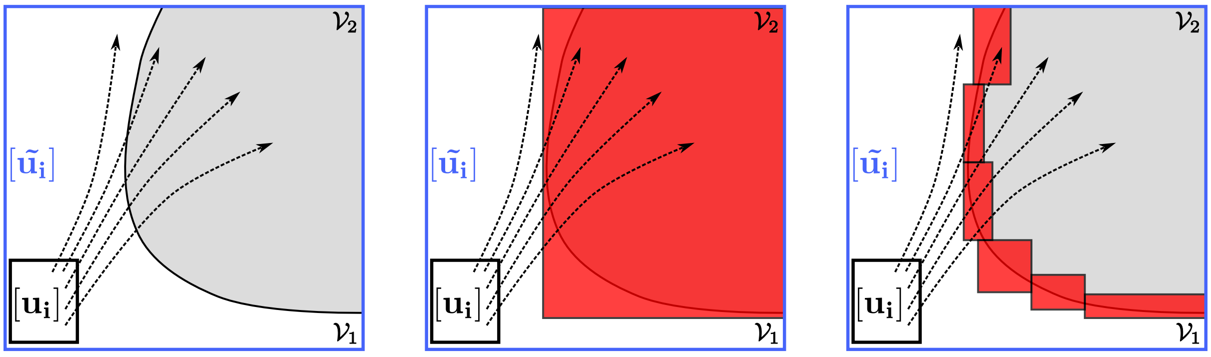
1. First, the presence of another domain $\mathcal{V}_2$ can be used to contract the tube so as to get a tinier enclosure. Because of its inconsistency with the domain $\mathcal{V}_2$, a part of the tube enclosing the solution driven by the first dynamic $\mathbf{d_1}$ can be removed (Fig. 5.6c).
$$[\mathbf{\tilde{u}_i}] \leftarrow [\mathbf{\tilde{u}_i}] \cap \mathcal{V}_1$$

2. Second, it has to be noted that the enclosure $[\mathbf{u_2}]$ that is associated with the first dynamic $\mathbf{d_1}$ does not exist at t_2 anymore. This is an important piece of information; it implies that the solution that is only driven by $\mathbf{d_1}$ necessarily crosses the guard, and does not exist at t_2. Then the time-step can be reduced to decrease the size of the box along the time dimension.

3. Once the global enclosure is contracted, the uncertain time initial value can also be contracted using the inconsistency argument with the original domain $\mathcal{V}_1$.
$$[\mathbf{u_{i+1}}] \leftarrow [\mathbf{\tilde{u}_i}] \cap \mathcal{V}_2$$

4. Finally, a complementary method can be used on the uncertain time initial value $[\mathbf{u_{i+1}}]$ and the time window $[t_1; t_2]$: the SIVIA algorithm. It returns a set of uncertain time initial values. The uncertain time initial value $[\mathbf{u_{i+1}}]$ is subdivided into several boxes that enclose the frontier. These boxes are the uncertain time initial values which are used to continue the integration with the second dynamic $\mathbf{d_2}$ that is associated with the domain $\mathcal{V}_2$.

On Figures 5.8, there is a global bounding box $[\tilde{\mathbf{u}}_\mathbf{i}]$ which is computed from an initial value $[\mathbf{u_i}]$ at time-step t_i and that is contracted with the previous improvements (1 and 2) over the time window $[t_i; t_{i+1}]$ (Fig. 5.8a). Then an uncertain time initial value is computed (Fig. 5.8b) from this bounding box thanks to the contraction process detailed in the third improvement. Finally the SIVIA algorithm refines the uncertain time initial value (Fig. 5.8c) and a large part of the uncertain time initial value is removed and not taken into account to continue the integration.



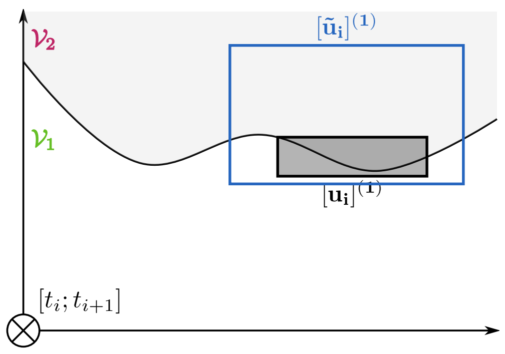
(a) Global enclosure $[\tilde{\mathbf{u}}_\mathbf{i}]$ (b) Contraction of UTIVP (c) SIVIA

Figure 5.8: Refinement of the UTIVP with contraction and SIVIA

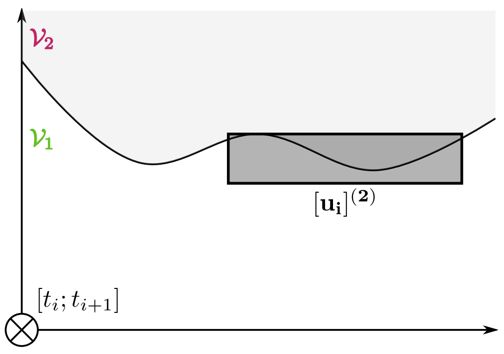
5.4.4 Limitations

This theorem guarantees that the enclosure computed safely encloses the solutions of p-ODEs from a set of uncertain time initial values $[\mathbf{u_i}]_\mathbf{p}$ on a time window $[\underline{t_i}; \overline{t_i}]$. Unfortunately its impact is limited in practice because of the over-wrapping interval arithmetic properties. Therefore, the theorem can only be used to prove the guarantee of an enclosure when the conditions are respected. However, such enclosures appear to be difficult to find (with the exception of the infinite enclosure $]-\infty; \infty[^n$ of $\mathbb{R}^n$). This limitation is drawn on Figures 5.9.

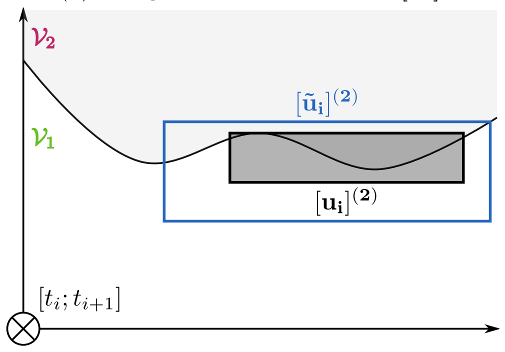
On the first figure (Fig. 5.9a) a bounding box $[\tilde{\mathbf{u}}_\mathbf{i}]'$ is computed by using the second dynamic $\mathbf{d_2}$. It is computed from the uncertain time initial value $[\mathbf{u_i}]'$ over the time window $[t_i; t_{i+1}]$. In the bounding box $[\tilde{\mathbf{u}}_\mathbf{i}]'$, a part of the enclosure is in the domain $\mathcal{V}_1$. Then, a new uncertain time initial value $[\mathbf{u_i}]^{(\mathbf{2})}$ is built (Fig. 5.9b) that replicates the process with the first dynamic. Using the first dynamic, a new bounding box is computed $[\tilde{\mathbf{u}}_\mathbf{i}]^{(\mathbf{2})}$ (Fig. 5.9c). A part of this bounding box goes back in the second dynamic and is


$\mathcal{V}_2$
$[\tilde{\mathbf{u}}_\mathbf{i}]^{(1)}$
$\mathcal{V}_1$
$[\mathbf{u}_\mathbf{i}]^{(1)}$
$[t_i; t_{i+1}]$

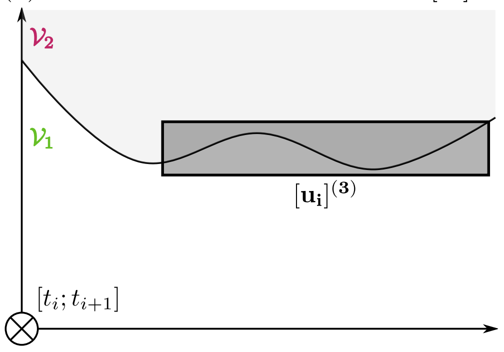
(a) Integration with $\mathbf{d_1}$ from $[\mathbf{u_i}]'$



(b) New uncertain time initial value $[\mathbf{u_i}]^{(\mathbf{2})}$


$\mathcal{V}_2$
$[\tilde{\mathbf{u}}_\mathbf{i}]^{(\mathbf{2})}$
$\mathcal{V}_1$
$[\mathbf{u}_\mathbf{i}]^{(\mathbf{2})}$
$[t_i; t_{i+1}]$

(c) Integration with $\mathbf{d_2}$ from $[\mathbf{u_i}]^{(\mathbf{2})}$



(d) New uncertain time initial value $[\mathbf{u_i}]^{(\mathbf{3})}$

Figure 5.9: Growth of the uncertain time initial value

not included in the uncertain time initial value $[\mathbf{u_i}]'$ that was previously used to compute the integration with the second dynamic. As a consequence, the theorem cannot be applied, and a new integration must be computed with the second dynamic $\mathbf{d_2}$ on a larger uncertain time initial value $[\mathbf{u_i}]^{(\mathbf{3})}$.

Thus, the number of returns allowed by the solutions in each dynamic must be controlled and is defined with a limit l. This aims to avoid an infinite enclosure and to limit the computation time that is required by the algorithm. The assumption that is made about the limit l allows the creation of a guaranteed enclosure of all the solutions of a p-ODE system with an uncertain time initial value. Indeed, it is now made possible because all of the solutions that are enclosed can oscillate between the same dynamics l times at most. The cooperative systems can be solved with guarantee by assigning the value 1 to l; similarly, by choosing a larger value l some non-cooperative systems can be dealt with.

5.5 Conclusion

In this chapter the guaranteed integration of piecewise ordinary differential equations (p-ODEs) has been presented. In the first section (Section 5.2) specific mathematical objects have been defined such as the validity domains $\mathcal{V}$ or the frontiers $\mathcal{F}$ of the dynamics. These objects were used to detail the problematic of the guaranteed integration of p-ODEs that come from the dynamics engaged and their validity domains that can be disjointed or not. Dynamics can be conservative or purgative in regards to their behaviors and their validity domains. When they are linked one another in a p-ODEs, the global behavior is induced by the properties of each dynamic and the system can be cooperative or competitive.

In the second section (Section 5.3), the uncertain time initial value problem (UTIVP) that is an extension of the initial value problem presented in Chapter 3 has been introduced. A reduction toward the initial value problem has been detailed through the use of an extension of the Picard operator.

In the last section (Section 5.4), a guaranteed integration algorithm has been developed on piecewise-ODE. This algorithm is based on the algorithm previously detailed in Chapter 3 with a specific treatment of the areas defined by the intersection of the different validity domains. Consequently, a new theorem has been developed and proved, that is based on the extension of the Picard iterator defined in Section 5.3. The practical limits of this theorem have been discussed, and a limited version has been presented with some improvements such that the use of the SIVIA previously presented at the end of Chapter 4.

The precision of the enclosures computed by the method presented in this chapter is limited by the tube representation (piecewise interval functions) used to guarantee the enclosure of solutions. The size of the enclosures may be reduced using alternative representations:

- Affine Arithmetic instead of Interval Arithmetic
  (see DynIbex [Alexandre dit Sandretto and Chapoutot, 2016a])
- Piecewise Taylor Models (based on the Interval Arithmetic) instead of Piecewise Interval Functions (see Flow* [Chen et al., 2012])

6 Application to the Optimization

Effectiveness is doing the things that get you closer to your goals. Efficiency is performing a given task (whether important or not) in the most economical manner possible. Being efficient without regard to effectiveness is the default mode of the universe.

– Timothy Ferriss

Contents

6.1 Introduction

The performance is an important aspect in our society. The optimization allows to get better results in a system, without any evolution of its model. The optimization is an important part of many industrial processes, it consists in being smarter and using tricks in order to get the full potential of the system.

The optimization process is a complex method that is used to find the best parameters inside a range of values. The range of values can be explicitly or implicitly described. In the second case, the process becomes more complex because of a subproblem to solve in order to get the explicit range of values.

This chapter is organized as follows: in Section 6.2, the optimization problem is defined. In Section 6.3 a deterministic global optimization algorithm that is guaranteed is detailed and its limits are discussed. Finally, in Section 7, the optimization algorithm is applied on several examples, as well as the guaranteed integration method that has been developed in Chapter 5.

6.2 Optimization Problems

A continuous optimization problem is composed of a real function $\mathbf{f_{cost}}$ which has to be minimized over a set of variables $\mathbf{x}$.

$$min_{\mathbf{x}\in\mathcal{D}_\mathbf{x}}\mathbf{f_{cost}}(\mathbf{x}) \tag{6.1}$$

where:
- $\mathbf{x} = (x_1, \dots, x_n)$ are the decision variables
- $\mathbf{f_{cost}} : \mathcal{D}_\mathbf{x} \subseteq \mathbb{R}^n \to \mathbb{R}^m$ is the cost function
- $\mathcal{D}_\mathbf{x}$ is the feasible domain

6.2.1 Local and Global Minima

An optimization problem is solved when a global minimum $\mathbf{x}^*$ in the set of all the feasible domain $\mathcal{D}_\mathbf{x}$ is found. Such a process is difficult because there are numerous of elements in $\mathcal{D}_\mathbf{x}$ (an infinite number if $\mathcal{D}_\mathbf{x} \subseteq \mathbb{R}^n$) and because of the function $\mathbf{f_{cost}}$'s irregularity (non-linearity, non-convexity) in which a large number of local minima exist.

6.2.1.1 Definition: Local minimum

The value $\mathbf{x} \in \mathcal{D}_\mathbf{x} \subseteq \mathbb{R}^n$ is a local minimum of the function $\mathbf{f_{cost}}$ over $\mathcal{D}_\mathbf{x}$ when the following condition is valid:

$$\forall \mathbf{y} \in \mathcal{D}_\mathbf{x} \cap \mathcal{V}, \quad \mathbf{f_{cost}}(\mathbf{x}) \leq \mathbf{f_{cost}}(\mathbf{y}) \tag{6.2}$$

where $\mathcal{V}$ is an open subset containing $\mathbf{x}$.

6.2.1.2 Definition: Global minimum

The value $\mathbf{x} \in \mathcal{D}_\mathbf{x} \subseteq \mathbb{R}^n$ is a global minimum of the function $\mathbf{f_{cost}}$ over $\mathcal{D}_\mathbf{x}$ when

$$\forall \mathbf{y} \in \mathcal{D}_\mathbf{x}, \quad \mathbf{f_{cost}}(\mathbf{x}) \leq \mathbf{f_{cost}}(\mathbf{y}) \tag{6.3}$$

6.2.2 Single objective vs. multi-objectives

In order to take different quantities into account, the cost function $\mathbf{f_{cost}}$ may take multiple forms: the single objective form or the multi-objective form. The majority of the industrial problems are expressed through multi-objective cost functions because of the conflicting criteria which have to be optimized.

6.2.2.1 Single and multi-objective problems

The evaluation of the cost function $\mathbf{f_{cost}}$ returns a value in $\mathbb{R}^m$. Then the optimization problem gets either one best value ($m = 1$) or a set of values which cannot be compared pairwise ($m \geq 2$). The set that contains the best values is known as the Pareto set.

6.2.2.2 Definition : Domination

Let $\mathbf{x_1}$ and $\mathbf{x_2}$ two elements in $\mathcal{D}_\mathbf{x}$. The first element $\mathbf{x_1}$ dominates the second $\mathbf{x_2}$ through the evaluation of the cost function $\mathbf{f_{cost}}$ when

$$\mathbf{f_{cost}}(\mathbf{x_1}) \leq \mathbf{f_{cost}}(\mathbf{x_2}) \tag{6.4}$$

Since $\mathbf{f_{cost}}$ is valued in $\mathbb{R}^m$, it can be detailed

$$\forall i \in \{1 \dots m\}, \quad \mathbf{f_{cost}}(\mathbf{x_1})_i \leq \mathbf{f_{cost}}(\mathbf{x_2})_i \tag{6.5}$$

6.2.2.3 Definition : Pareto front

When it is associated to a function $\mathbf{f_{cost}}$, the *Pareto* frontier is noted $\mathcal{P}_{\mathbf{f_{cost}}}$. It is a set of non dominated solutions through $\mathbf{f_{cost}}$.

$$\forall \mathbf{x_1}, \mathbf{x_2} \in \mathcal{P}_{\mathbf{f_{cost}}}, \quad \mathbf{f_{cost}}(\mathbf{x_1}) \not\leq \mathbf{f_{cost}}(\mathbf{x_2}) \quad \wedge \quad \mathbf{f_{cost}}(\mathbf{x_2}) \not\leq \mathbf{f_{cost}}(\mathbf{x_1}) \tag{6.6}$$

6.2.3 Optimization with regards to constraints

The optimization process of a real function $\mathbf{f_{cost}}$ onto a validity domain $\mathcal{D}_\mathbf{x}$ is a complex problem which comes from

- the large number of variables,
- the size of the validity domain $\mathcal{D}_\mathbf{x}$,
- the large number of local minima.

Usually the feasible domain $\mathcal{D}_\mathbf{x}$ is not explicitly described and has to be computed from a set of constraints. The optimization problem consequently becomes even more complex:

$$\min_{\mathbf{x} \in \mathcal{D}_\mathbf{x}} \mathbf{f_{cost}}(\mathbf{x}) \tag{6.7}$$

$$w.r.t. \quad \mathcal{C}(\mathbf{x}) \tag{6.8}$$

where $\mathcal{C}$ represents the set of constraints.

Many techniques can be used to find the best value $\mathbf{x}^*$ with respect to the set of constraints $\mathcal{C}$. If $\mathcal{C}$ is a set of linear constraints as well as the cost function $\mathbf{f_{cost}}$ such that

$$\min_{\mathbf{x} \in \mathcal{D}_\mathbf{x}} \mathbf{c}.\mathbf{x} \tag{6.9}$$

$$w.r.t. \quad A\mathbf{x} \leq b \tag{6.10}$$

then the simplex algorithm is used to compute the optimal point.

As far as optimization is concerned, many methods exist. Attention should be focused on the complete search process, because it checks all of the elements in the search area and finds out those which are feasible and, among them, the ones which are global optima.

It has to be reminded that when the area in which the search takes place contains an infinite number of elements $\mathbf{x}$, the interval representation is required. This representation can be associated with branch and bound methods in order to perform optimization.

6.3 Interval Branch and Bound Algorithm

The interval Branch and Bound algorithm (IBBA) [Messine, 1997], [Messine et al., 2001], [Messine, 2005] is an algorithm used to find the best feasible values according to an objective function and a set of constraints. The strength of this method resides in the fact that it provides a proof of the existence or of the non-existence of a feasible solution and its optimality.

This method consists in a recursive split and recursive evaluation of the domain by the objective function. At each iteration the current search space is divided into several parts which are evaluated afterwards via the objective function that has to be minimized. After a large number of iterations, the optimal value emerges when the domains considered are small enough.

The main limitation of this method is that it requires a great amount of memory as well as a long computation time, which is problematic when the search space is large. Moreover, in the problem that is tackled in this thesis, because the values are continuous, the search space may be infinitely split (it depends on the numerical precision used to represent the values).

The interval branch and bound algorithm can therefore be improved thanks to some pruning methods. Indeed they would remove the parts of the search space which are guaranteed not to contain a feasible solution or the optimal solution. An upper bound over the objective function is updated at each step and decreases all along the process. When an area of the search space is above the upper bound it can be ignored because the current solution (that is the current value that minimizes the objective function) is better than all the feasible solutions that are inside this area.

However, even with this improvement, the method suffers from an exponential complexity that increases with the number of decision variables $\mathbf{x}$ and it also requires a lot of computations in case of complex models with several local optima. A faster convergence towards some feasible domains and solutions can be reached when this algorithm is coupled with the constraint satisfaction process [Messine, 2004] in which the algebraic, functional, ODEs and p-ODEs constraints propagators are included. Finally a global algorithm based on decomposition, contraction, evaluation and pruning is built.

The algorithm is detailed in Algorithm 2. For each step of the IBBA, a node is selected according to a strategy (depth-first, width-first, best-first...) and bisected in sub-nodes on the selected dimension (round-robin, largest side...). The sub-nodes are then contracted using the constraint satisfaction process through two steps. First, algebraic constraints are used to filter search domains using hull-consistency methods (HC4) [Benhamou et al., 1999], because they are fast to compute. If the nodes still contain some feasible domains then the differential constraints are propagated (Interval based guaranteed integration). This step costs memory and time computing; they are the main negative impacts from the methods previously mentioned.

At the end of each iteration, if the node still contains some feasible solutions, a final step is usually used in the IBBA in order to update the upper bound. This part consists in finding a feasible solution in the domain described by the node. This local value can

Algorithm 2 : *Optimize*($\mathbf{x} \in \mathcal{X}$)

```
1:  Node_Collection $searchSpace \leftarrow \{createNode(\mathbf{x})\}$
2:  Node_Collection $solutions \leftarrow \emptyset$
3:  while $searchSpace \neq \emptyset$ do
4:     Node_Collection $subnodes \leftarrow cut(extract(searchSpace))$
5:     for all $node$ in $subnodes$ do
6:        $contract(node, upperBound(solutions))$
7:        if $unfeasible(node)$ or $isDominated(node, solutions)$ then
8:           $subnodes \leftarrow subnodes \setminus \{node\}$
9:        else
10:          $searchSpace \leftarrow searchSpace \cup \{node\}$
11:          Node $point \leftarrow lookForFeasiblePoint(node)$
12:          if $feasible(point)$ and not $isDominated(point, solutions)$ then
13:             $searchSpace \leftarrow removeNodesDominatedBy(searchSpace, point)$
14:             $solutions \leftarrow solutions \cup \{point\}$
15:          end if
16:       end if
17:    end for
18: end while
19: return $solutions$
```

be computed at the middle of the current area of the search space considered, or it can be computed randomly inside this area. Computing both feasibility and cost from a punctual value (and not from an interval vector) limits the over-approximation caused by Interval Arithmetic; a tinier enclosure is reached. When a feasible solution is found, it is added to the solution manager which can have many implementations according to the multi-objective properties (Pareto set, lexicographic, coefficients...). The value of the objective can also be used to reduce the search space domains, with the intention to minimize the goal function.

6.3.1 Implementation

In this work, the IBBA has been developed using several components in order to apply specific strategies to specific problems. The search space is stored inside a root node, and the algorithm creates sub-nodes from this initial root node. The components are listed on the following lines.

- The Pareto object is the component that stores the set of optimal solutions. Different Pareto objects are used according to the objective function. On a multi-objective problems, a Lexicographic-Pareto will have a different behavior that a (α, β, γ)-Pareto where coefficients are paired with objectives.

- The Contractor is the algorithm that computes the contraction of the nodes according to the model. The global contractor is a meta-contractor that is made of subcontractors so that each constraint or set of constraints may have a peculiar contractor. It allows us to treat each constraint specifically. The contractor used to contract ODEs is different from a contractor for an algebraic constraint.

- The Picker is the component that chooses which node to treat inside the set of nodes. Each Picker possesses a specific strategy. Largest-Picker selects the node that has the larger size, Queue-Picker implements a breadth first search algorithm while Stack-Picker applies the depth-first-search strategy. Also the BestNode-Picker and WorstNode-Picker select the nodes that have the lowest lower bound and the highest upper bound depending on the cost function.

- The Cutter is the module that is used to cut the nodes on one dimension. The Picker can then follow the Round-Robin strategy or it can cut the largest dimension depending on the strategy.

- The Tracker is the module that tries to find a local solution within the node. The Middle-Tracker consider the point in the middle of the node while the Random-Tracker randomly choose a point inside the node. Smarter versions are the MiddleContract and the RandomContract Trackers that use a contractor to reduce the domains of all the dimensions each and every time a dimension is fixed.

This implementation design is inspired from the IBEX global optimization plugin, and is useful to develop new modules or meta modules.

6.3.2 IbexOpt plugin

The IBEX library [Chabert, 2007] has been extended with IbexOpt that is a global optimization plugin [Trombettoni et al., 2011], [Ninin, 2015] based on the interval branch and bound that is presented in the last section with several improvements. A robust and efficient interval linearization that is used to update the lower and the upper bounds is presented in [Trombettoni et al., 2011] and extended in [Araya et al., 2014]. A filtering algorithm is detailed in [Sans et al., 2016] that improves the computation of these bounds. These specific works aim to reduce the search space, thus the computation time of the algorithm by improving the computation of the lower and the upper bounds. Another aspect that could have a strong impact on the efficiency of the global algorithm is the strategy used in the selection node. In [Neveu et al., 2016], because it is necessary to find a balance between diversification and intensification that is relevant, the authors present two node selection policies that are based on the upper bound of each node and on the feasible regions at each node of the best-first search.

6.3.3 Loss of guarantees

Basically, the IBBA is used to guarantee the results of the optimization process. If the results computed using these algorithms are guaranteed in theory, the overapproximation of the results with Interval Arithmetic is a fact that should not be neglected and that could ruin the guarantee of the results in most situations. The solution is guaranteed with a certain amount of tolerance towards the constraints. An example of a solution that is not guaranteed is developed in this paragraph.

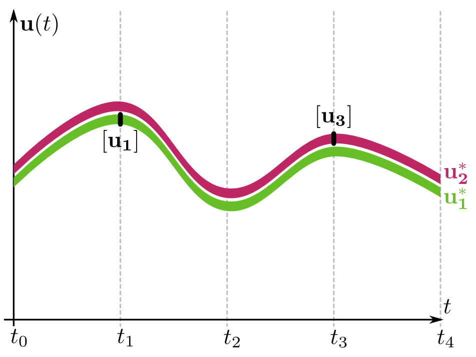
Using Interval-based guaranteed integration, the enclosure of the solution of the differential systems can be computed. In the case of an IVP model, the solution is guaranteed to exist and thus to be enclosed in a tube afterwards. The initial value problems are described in our model by an ODE or a p-ODE, and then by an algebraic constraint that describes the initial value. For such systems, the existence of the solution as well as its enclosure inside the tube are guaranteed.

However, the situation becomes critical when more than one algebraic constraint exists on one solution of a differential system (representing a system with multiple initial values). The reason for this is that the second constraint could remove the part of the enclosure that contains the solution without generating an inconsistent enclosure (due to the growing effect from the IA). The step by step integration method suffers from over-approximation, that is why an enclosure could still exist even though if the solution is not inside.

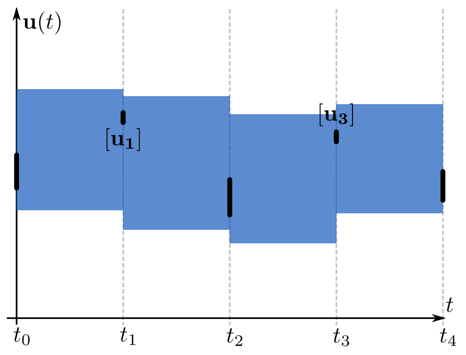
For example, on Figures 6.1, two initial value problems are considered with the same ODE.

$$\begin{aligned}\mathbf{u}'(t) &= \mathbf{d}(t,\mathbf{u}) \\ \mathbf{u}(t_1) &\in [\mathbf{u_1}]\end{aligned} \qquad \textit{and} \qquad \begin{aligned}\mathbf{u}'(t) &= \mathbf{d}(t,\mathbf{u}) \\ \mathbf{u}(t_3) &\in [\mathbf{u_3}]\end{aligned} \tag{6.11}$$

The first figure (Fig. 6.1a) represents the exact solution of each IVP. In other words, it represents the minimal tubes that contain all the solutions for each IVP.



(a) Sets of solutions for each IVP of an ODE



(b) Non empty enclosure of the multi-IVP of an ODE

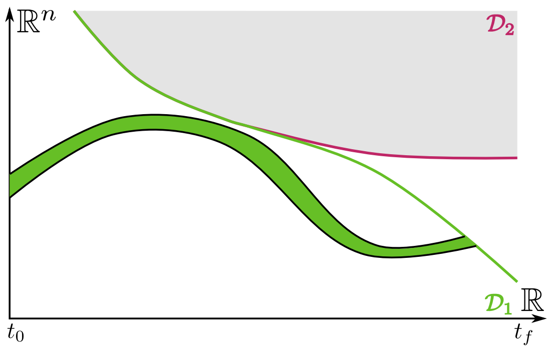
Figure 6.1: Enclosure of an unfeasible solution

Now, taking into consideration a more complex system that would be composed of the same ODE, then, with the two initial values:

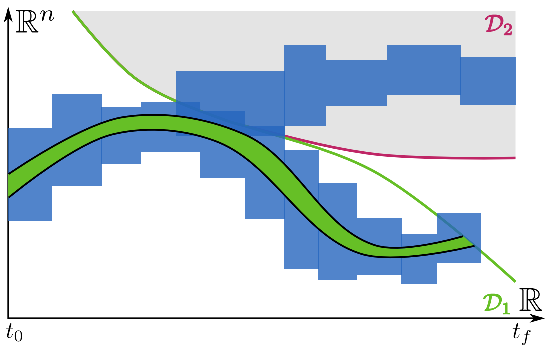
$$\begin{aligned}\mathbf{u}'(t) &= \mathbf{d}(t,\mathbf{u}) \\ \mathbf{u}(t_1) &\in [\mathbf{u_1}] \\ \mathbf{u}(t_3) &\in [\mathbf{u_3}]\end{aligned}$$

There is no solution since the intersection of the solutions of the previous IVP is empty and does not exist over $[t_1; t_3]$. The problem is that when the interval arithmetic is used to solve this complex dynamic system, the enclosure could still exist and not be empty because of the over-approximation (Fig. 6.1b). That is problematic, because this solution could be considered as feasible in the IBBA, and could be used to prune other areas of the search space that could contain feasible solutions. At worst, the algorithm could return a non-feasible solution and consider that it is feasible and optimal. As long as this remains a possibility, one way to avoid such a situation is not to model complex problems in which multiple points of the ODE are implied in constraints.

Another issue that involves p-ODEs is, similarly, the over-approximation along the resolution process. This issue is caused by the existence of totally disjointed guards in the p-ODEs, which may draw a space with no dynamic. First figure (Fig. 6.2a) represents the exact solution of a p-ODE from an initial value at t_0. Because of the guards that limit the domain $\mathcal{D}_1$ of the first dynamic, the solution does not exist at time-step t_f. However, because of the over-approximation during the guaranteed integration process (Fig. 6.2b), the enclosure that is computed intersects the domain $\mathcal{D}_2$ and an enclosure of the solution is found at time-step t_f. The enclosure is guaranteed and it encloses the solution of the p-ODE. However the value of the enclosure at time-step t_f could be used in the IBBA to compute the evaluation of the objective function. When this happens, the enclosure is considered as a feasible solution and is used to update the upper bound, to prune the other parts of the search space, and could therefore be considered as the optimal solution.



(a) Set of solutions for the IVP of a p-ODE



(b) Non-empty enclosure of the solution

Figure 6.2: Enclosure of an unfeasible solution at t_f

It seems the presence of ODEs with algebraic constraints over several time-steps or p-ODEs with domains that are not assigned to any dynamic may possibly lead to the loss of the guarantee. The previous cases dealing with the loss of guarantee with ODEs in the optimization problems may also be found along with the algebraic constraints that link the same groups of variables. Considering the fact that the main interest of the method is to certify the feasibility of a solution, and to make that solution optimal, the loss of the guarantee is a major issue.

However, in practice, this method provides good solutions and is more robust than the numerical approaches.

As it was stated earlier, and as far as optimization is concerned, this kind of approach is exponential depending on the number of variables in the system that needs to be optimized.

The amount of time required to optimize large-sized problems may still be a problem even when the contraction method is used as well as some smart heuristics to branch on the variables.

In the recent years, some progress has been achieved. The idea is to tend toward the creation of some cooperative optimization algorithms, using parallel computing along with evolutionary algorithms coupled with an IBBA [Vanaret, 2015].

6.4 Conclusion

In this chapter optimization problems have been introduced with their solutions. A deterministic algorithm that is an interval extension of the Branch and Bound algorithm has been presented. Its implementation has been detailed through the use of five modules that are:

- the solution manager that stores the optimal and feasible solution,
- the global contractor that contracts the domains of the variables,
- the picker that selects the node to treat,
- the cutter that cuts the node in subnodes,
- the tracker that tries to find a local solution.

This implementation inspired by IBEX is an advantage because it allows the user to design the global optimization that will be adapted to the instance of the model is facing. Also, new strategies can easily be implemented because it only requires to implement the module that is concerned.

In this chapter, a relevant paradox has been discussed that is the loss of the guarantee when using interval based guaranteed arithmetics on related constraints which can be both differential and algebraic constraints, because of the overapproximation of the interval arithmetic.

Results of numerical tests computed with the optimization tool discussed in this chapter are presented in Chapter 7 on several optimization problems.

7 Numerical Tests

7.1 Introduction

This chapter presents the results of the methods that were previously applied on problems of various kind. The first result that is detailed is the one of the optimization method. This method was applied on the design of a constant magnetic engine (as it was defined at the beginning of this thesis). Afterwards, the results focused on the guaranteed integration method that was developed to solve piecewise ODEs. At last, the global method was applied on an optimization problem that contains both algebraic and piecewise ODEs constraints.

7.2 Optimization of the Constant Magnetic Engine

In this section the solution returned by the proposed optimization tool on the design of a constant magnetic engine is detailed. After 50000 iterations, the algorithm returns the following solution, the objective of which is $6.0857.10^{-4}$.

Variable	Value in the solution
e	$1.0076.10^{-3}$
l_a	$5.1313.10^{-3}$
E	$3.5922.10^{-3}$
$beta$	$8.0144.10^{-1}$

The results of the test are computed within the IBBA as planned below:

- The HC4 contractor provides the contraction at each iteration
- The Largest First picker takes out the node with the largest side from the list of nodes
- The Round-Robin cutter then splits successively each dimension of the selected node

Two different strategies are tested to find a feasible solution inside the current node. The first one of those is the classic IBBA; it tests how feasible the center of the current node is. The second one is named IBBA+. As soon as the tracker fixes a dimension it also computes a set of contractions. However it does require more computation time.

After the 50000-th iterations, 52503 nodes are still waiting on the list of nodes to be treated in the IBBA+. The updates of the upper bound in both versions of the algorithm are detailed in Table 7.1.

Iteration	Classic IBBA	IBBA+
0	8.8458608342810625	10.2152959486188303
1	7.5370391369965011	8.0354003492125149
3		7.5661081382451560
7		7.2275685866515999
16	7.0732293231167245	
23	6.8221865383365666	6.6553217552697938
37	6.6507260315448171	
55		6.3977628136495173
59	6.5967377277043970	
155		6.2655233014983895
188	6.3689039770893586	
300		6.2056857181902784
315	6.3455713114835498	
1112		6.1632080592304700
1206	6.2333580492079140	
1210		6.1225309866000551
1314	6.2020896440044093	
7144		6.1082986231933748
1790	6.1911465910313181	
7798	6.1747863873140205	
7814	6.1416725529062251	
9478		6.1071718877779796
10362	6.1380297300968952	
13859		6.0980763545317182
14568	6.1377531920707386	
20454	6.1312195826489170	
42942	6.1180383295963796	
43054	6.1058220574739712	
44874		6.0857241659314132
49860	6.1054260091690201	
Time(s)	46.13	59.05

Table 7.1: Update of the upper bound (10^{-4}) during the optimization depending on the strategy used to find a feasible solution

As the tracker chooses a value on one dimension, then propagates this value on other dimensions thanks to the set of constraints, the algorithm is efficient and provides relevant upper bounds after only a few iterations.

The other side of the coin is that at each iteration, while searching for a feasible solution, a large number of contractions is performed, therefore increasing the computation time. As a consequence the limit of 50000 iterations is reached in only 46s with the classic IBBA and in 59s with IBBA+.

In this reformulated problem the search for a feasible solution is a difficult problem and is not required. However, in some more complex problems in which the computation of a feasible solution is a difficulty, this trick could be relevant. Another strategy could be to track feasible solutions at the beginning of the optimization process with this robust strategy and then to use the classic approach by trying the center of the boxes when they are small enough.

7.3 Guaranteed integration of piecewise-ODE

In this paragraph, in order to focus on the guaranteed integration of a piecewise ODE, a simple situation is modeled. It is inspired from ballistics because this science uses kinematics equations. Let an object launch upward from an initial height $x_0 \in [10; 15]$ with a velocity $v_0 \in [10; 11]$. In the first environment the friction does not exist and the object is the subject of a gravity force g. At some point in its free fall, the object reaches a lower limit $l \in [-0.5; 0.5]$ and below this level a second environment is defined that creates a friction f and slows down the free fall of the object depending on its vertical velocity.

Let $\mathbf{u} = (u_1, u_2)$ the couples of functions such that u_1 is the height of the object function of time t and u_2 is its velocity. Then the initial value $\mathbf{u}(t_0)$ is defined as following

$$\mathbf{u}(t_0) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}(t_0) \in \begin{pmatrix} h_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} [10; 15] \\ [10; 11] \end{pmatrix} \tag{7.1}$$

From this initial situation, the trajectory of the object defined with two dynamics $\mathbf{d_1}$ and $\mathbf{d_2}$

$$\mathbf{d_1}(\mathbf{u}, t) = \begin{pmatrix} u_2 \\ [-g] \end{pmatrix} \text{ and } \mathbf{d_2}(\mathbf{u}, t) = \begin{pmatrix} u_2 \\ [-g] - [2].u_2 \end{pmatrix} \tag{7.2}$$

depending on its elevation u_1 and the limit l.

$$\mathbf{u}' = \begin{cases} \mathbf{d_1}(\mathbf{u}, t) & \textbf{if} \quad u_1(t) \geq l \\ \mathbf{d_2}(\mathbf{u}, t) & \textbf{if} \quad u_1(t) \leq l \end{cases} \tag{7.3}$$

Note that the limit l is defined with the interval $[-1; 1]$ so that when the object elevation is within this range, the two dynamics are valid at the same time.

The guaranteed integration of this dynamic system is computed by the contractor developed in this research work and introduced in this thesis from the time-step $t_0 = 0$ to the final time-step $t_f = 9$. The results are presented on Figure 7.1. Knowing which model allows us to set a limit on the number of returns that are allowed for each dynamic. When the object reaches the second environment, it is not supposed to go back to the first environment. Consequently, the limit that is defined on the number of allowed returns in the previous dynamic is set to 0.

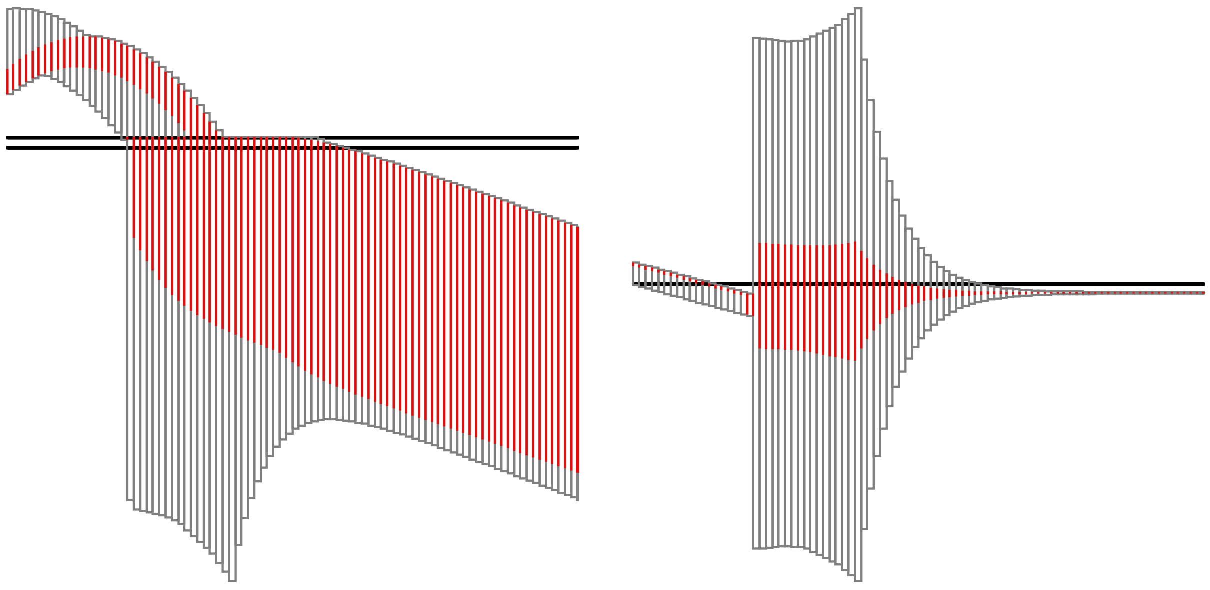

(a) $[u_1](t)$: enclosure of $u_1(t)$ (position) (b) $[u_2](t)$: enclosure of $u_2(t)$ (velocity)

Figure 7.1: Results of the guaranteed integration

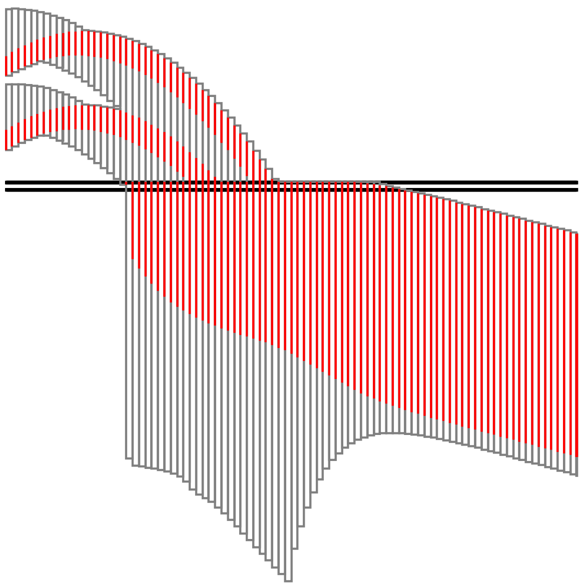
What could be relevant in a variation of this model would be if the initial value was to be composed of multi-intervals.

$$\mathbf{u}(t_0) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}(t_0) \in \begin{pmatrix} h_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} [10;15][30;35] \\ [10;11] \end{pmatrix} \tag{7.4}$$

On Figures 7.1a and 7.2a the two horizontal lines represent the boundaries of the validity domain for each dynamic. Between these lines, the dynamics $\mathbf{d_1}$ and $\mathbf{d_2}$ are valid at the same time. On Figures 7.1b and 7.2b the horizontal line underlines the moment when the value of the velocity is equal to 0. On these pictures, the gray boxes represent the global bounding boxes (the global enclosure of the solution over a time window) while the red lines represent the local enclosures at a local time-step. A relevant observation is that with no more information, the second dynamic $\mathbf{d_2}$ starts way before the local enclosure reaches the limit because of the large bounding boxes. That is because of the use of *Vnode-lp* [Nedialkov et al., 2001] to compute the guaranteed integration for each dynamic. *Vnode-lp* has been designed to provide tiny local enclosures with a minimal computation time; it was not designed to provide tiny global enclosures but the ones that are sufficient enough to compute the guaranteed integration.

A first solution to this issue is to consider that in this model, the global enclosure can be recomputed with the local enclosures such that the global enclosure from a time-step to another is the minimal enclosure that encloses these two successive local enclosures. With this method, the resolution of the piecewise-ODEs with multiple initial values is more accurate and the result is represented on Figure 7.3.

In the generic case, such an approximation is a limit to the guarantee of the results. However, for this peculiar dynamic system, only one global bounding box is not guar-


(a) $[u_1](t)$: enclosure of $u_1(t)$ (position)

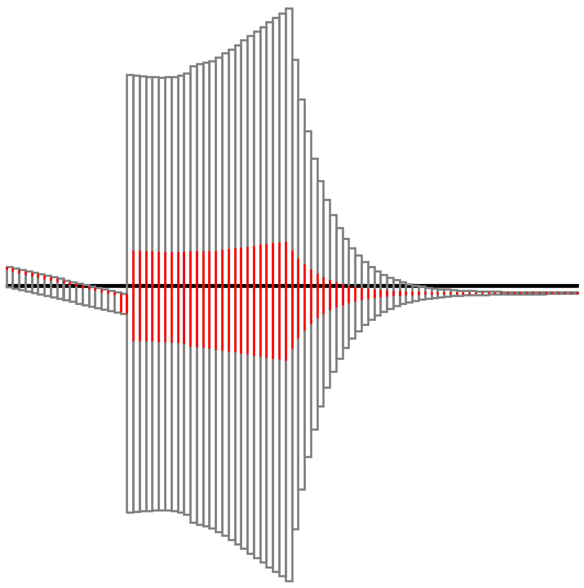
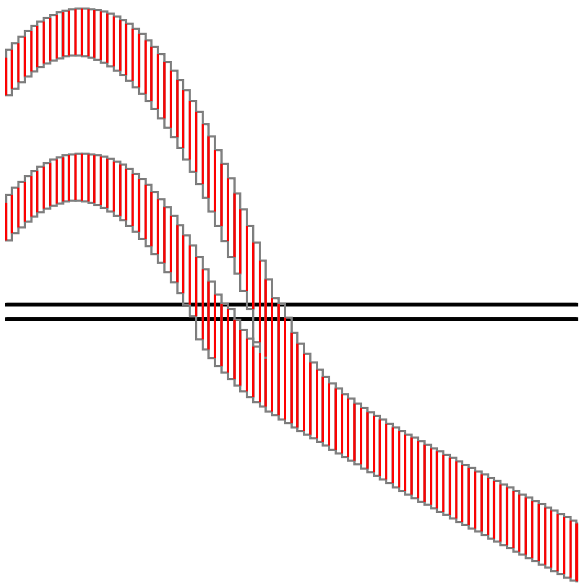

(b) $[u_2](t)$: enclosure of $u_2(t)$ (velocity)

Figure 7.2: Results of the guaranteed integration with multiple initial values


(a) $[u_1](t)$: enclosure of $u_1(t)$ (position)

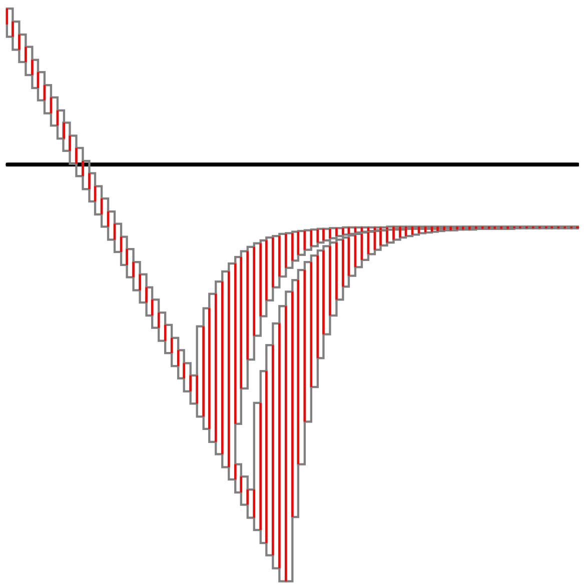

(b) $[u_2](t)$: enclosure of $u_2(t)$ (velocity)

Figure 7.3: Results of the guaranteed integration with multiple initial values and minimal global enclosure approximation

anteed to enclose all the solutions and it has no impact on the global resolution. This non-guaranteed bounding box is the one that encloses the change in the direction of the trajectory (from upward to downward). With this approximation and this specific model, the enclosure is guaranteed through the transition phase from the first to the second dynamic.

7.4 Optimisation with a piecewise-ODE system

The previous results focused on the optimization methods and then on the contractor dedicated to the contraction and the propagation of ODEs and p-ODEs constraints. Consequently, the next step is to present a global test that optimizes the problems containing p-ODEs constraints.

The optimization tool that has been developed in this work is not applied on the electromagnetic contactor because of its complexity. The large number of variables requires an exponential number of iterations on the branch and bound algorithm and each iteration would require too much time and memory to compute the guaranteed integration of the p-ODE that is contained in the model.

In order to keep things easy, the p-ODE that is contained in this optimization problem is the one that was previously tested. In this optimization problem the limit l used in the guard of the p-ODE and that defines the boundaries of each environment is a decision variable, as are the initial values h_0 and v_0 of the p-ODE system.

$$\begin{array}{lrcl}
 & \multicolumn{3}{l}{\mathbf{Minimize}_{h_0,v_0,l}\ f} \\
 & \mathbf{u}(t_0) & \in & \begin{pmatrix} h_0 \\ v_0 \end{pmatrix} \\
sc. & & & \\
 & \mathbf{u}'(t) & = & \begin{cases} \mathbf{d_1}(\mathbf{u},t) = \begin{pmatrix} u_2 \\ [-g] \end{pmatrix} & \textbf{if}\quad u_1(t) \geq l - [1] \\ \mathbf{d_2}(\mathbf{u},t) = \begin{pmatrix} u_2 \\ [-g] - [2].u_2 \end{pmatrix} & \textbf{if}\quad u_1(t) \leq l + [1] \end{cases}
\end{array} \tag{7.5}$$

where $h_0 \in [50; 100]$, $v_0 \in [20; 50]$ and $l \in [-10; 10]$. Note that the decision variable l is redundant with the variable h_0. An equivalent model can be defined in which $l = 0$, thus the variable h_0 would evolve in the interval $[40; 110]$. The objective function f is chosen to define three different problems, for each one of them the optimization process returns a specific solution that is close to the optimal. These results are computed with a limitation of 100 iterations for the IBBA+, which is enough to get relevant results. The three objective functions and their solutions are described in what follows.

- When $f = u_1(t_f)$ it minimizes the elevation of the final position.
  - $h_0 \in [62.81691269265343890993, 62.81691269265343890993]$
  - $v_0 \in [21.65528272995971192927, 21.65528272995971192927]$
  - $l \in [-8.82797157686574962554, -8.82797157686574962554]$
- When $f = abs(u_1(t_f))$ it leads the elevation toward 0 at the final position.
  - $h_0 \in [61.78823573248844525097, 61.78823573248844525097]$

- $v_0 \in [33.69914115462412240731, 33.69914115462412240731]$
- $l \in [-6.64534120897079816359, -6.64534120897079816359]$

- When $f = -u_1(t_f)$ it maximizes the elevation of the final position.
  - $h_0 \in [99.71890292204632544326, 99.71890292204632544326]$
  - $v_0 \in [49.88195981155473646140, 49.88195981155473646140]$
  - $l \in [5.42597042241877414170, 5.42597042241877414170]$

The results of the optimization depend on the objective function. The p-ODE contractor uses the approximation that was previously described and that contracts the global bounding boxes of the enclosures using successive local enclosures. The evolution of the upper bound during the optimization process is presented in the table 7.2 and the solutions it returns are on Figures 7.4.

Iteration	$f = u_1(t_f)$	$f = abs(u_1(t_f))$	$f = -u_1(t_f)$
0	-18.855	27.594	-89.396
1		22.980	-102.236
2		18.091	-147.337
3	-22.931	8.166	
4			-160.865
5	-26.593		
10			-174.900
13	-29.267		
19	-31.164		
23		3.586	
30	-32.701		-176.053
34		1.657	
38			-178.261
43			-182.683
50		1.418	
65		0.580	
73	-33.156		
74			-182.911
75	-34.387		
83	-34.388		
88			-183.619
90			-184.156

Table 7.2: Update of the upper bound of the objective during the optimization process over the problem defined in 7.5 depending on the cost function f.

On them (Fig. 7.4), the enclosure of the position u_1 is represented by red squares while the enclosure of the velocity is green. The horizontal gray line represents the horizontal axis. On Figure 7.4a the two horizontal black lines represent the boundaries of each validity domain. Between these lines the two dynamics $\mathbf{d_1}$ and $\mathbf{d_2}$ are valid.

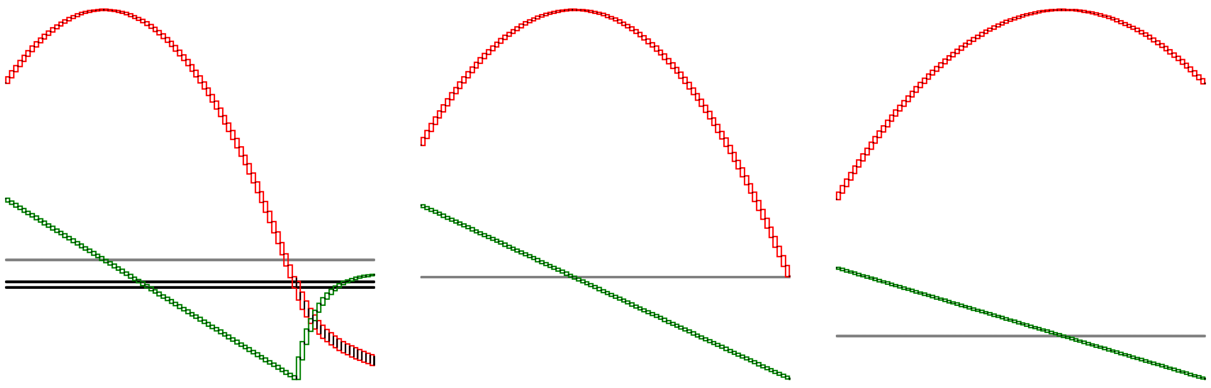

(a) $f = u_1(t_f)$ (b) $f = abs(u_1(t_f))$ (c) $f = -u_1(t_f)$

Figure 7.4: Enclosures $[u_1](t)$ and $[u_2](t)$ for each solution returned by the optimization process according to the objective function

7.5 Conclusion

In this chapter, a set of models and problems were introduced and tests were made upon them with the software GDODynS that was developed during this research work. The analysis of these results underlines the possibilities and the potential that resides in solving complex dynamic systems with guarantees. It also highlights the difficulties encountered by the method and the limitations imposed by the use of Vnode in this context.

The guaranteed integration of p-ODEs with the algorithm that was developed requires more computation time. This computation time can be minimized thanks to a set of optimizations in the algorithm. Nevertheless the cost of the algorithm that deals with p-ODE constraints will remain high but it could be the price to pay to get the guarantee.

In order to limit the complexity when dealing with such constraints, one should develop strategies within the IBBA that would limit the contraction of differential constraints. Another important aspect is the exponential complexity of IBBA. It requires a lot of computations and it cannot manage large-sized problems. This is the reason the results over the design of the electromagnetic contactor are not presented. A cooperative optimization algorithm based on genetic algorithms could be a winning strategy in order to provide good solutions faster, as well as to decrease the upper bound quickly and to prune large areas from the initial search space.

Part IV

Conclusion

In this thesis a set of methods has been presented that would optimize complex dynamic systems. During this work, an optimisation tool named GDODynS *Global and Deterministic Optimization for Dynamic Systems* has been developed. GDODynS has been developed in C++ and takes the advantages of the object-oriented programming with the development of several modules. The architecture of this optimization tool is presented on Figure 8 and all the modules have been detailed in the chapters of this thesis.

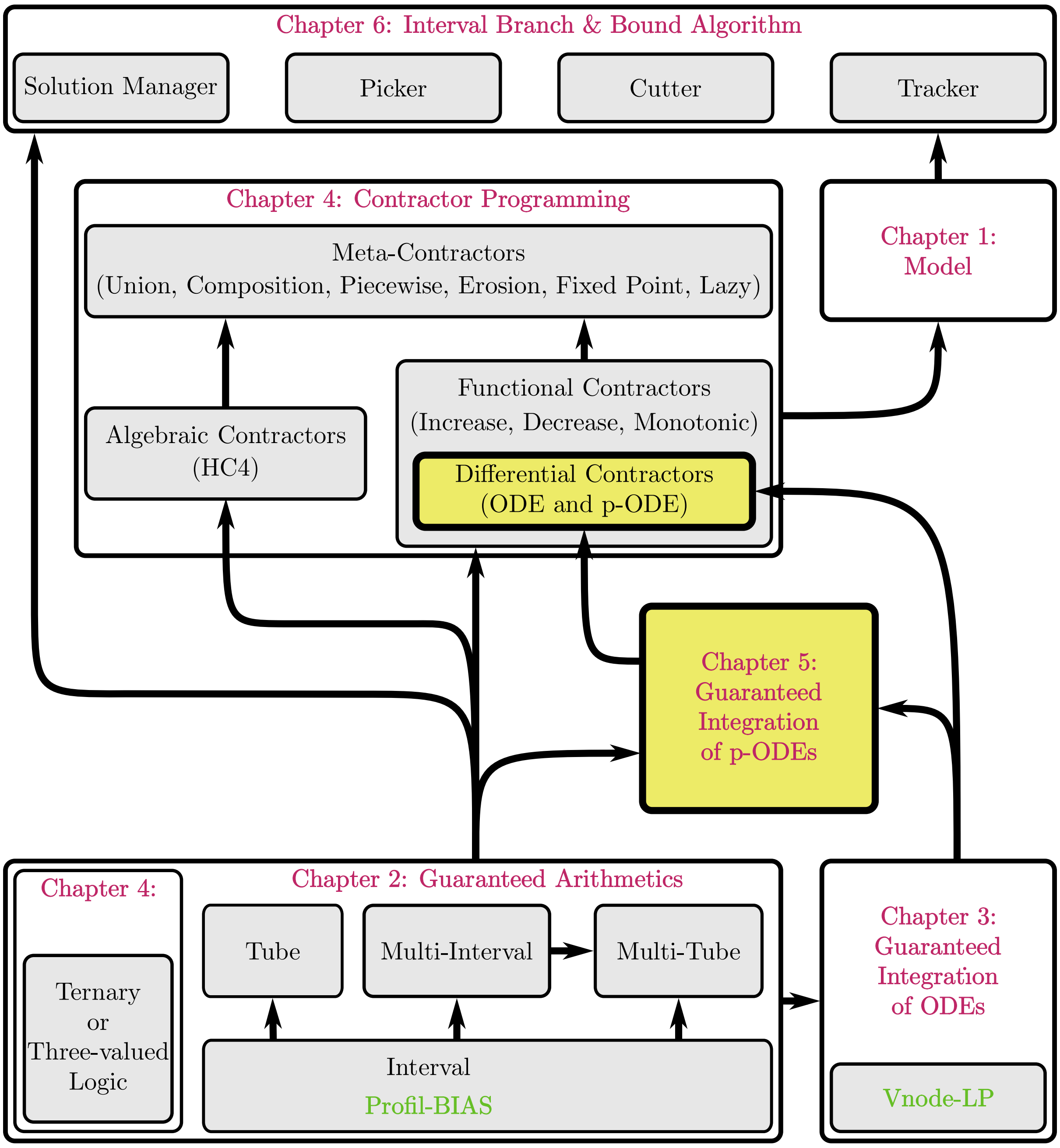

Chapter 6: Interval Branch & Bound Algorithm
Solution Manager
Picker
Cutter
Tracker
Chapter 4: Contractor Programming
Meta-Contractors (Union, Composition, Piecewise, Erosion, Fixed Point, Lazy)
Chapter 1: Model
Functional Contractors (Increase, Decrease, Monotonic)
Algebraic Contractors (HC4)
Differential Contractors (ODE and p-ODE)
Chapter 5: Guaranteed Integration of p-ODEs
Chapter 4:
Ternary or Three-valued Logic
Chapter 2: Guaranteed Arithmetics
Tube
Multi-Interval
Multi-Tube
Interval
Profil-BIAS
Chapter 3: Guaranteed Integration of ODEs
Vnode-LP

Figure 8: Architecture of the optimisation tool developed: GDODynS

The chapters followed a plan that led to a method to compute the guaranteed integration of piecewise ODE constraints (Chapter 5) and a global optimization algorithm (Chapter 6), starting from the analysis of the problems that motivated this research (Chapter 1).

All along this document, the reoccurring limitations of the methods were the computation time and the amount of memory required to compute solutions of the problems considered. Their complexity comes from their size, that is to say the number of decision variables, the type of variables (continuous, unary-functions) and the type of constraints (algebraic, functional, differential and piecewise differential).

Given that the computation time increases or decreases with the size of the model, choosing which method must be applied is of paramount importance. Consequently, several methods have been introduced in this thesis and implemented in GDODynS in order to solve these problems. All these methods are gathered into several modules that are represented on Figure 8.

1. First, the guaranteed arithmetics (with the interval and the tube arithmetic) has been detailed in Chapter 2. These arithmetic aspects are essential to guarantee operations and deal with continuous real variables and functional variables. In the optimization tool GDODynS, this module is based on the library *Profil-BIAS* [Knüppel, 1994] that implements the interval arithmetic. All the other guaranteed arithmetics, Multi-Intervals, Tubes and Multi-Tubes, are derived from the interval arithmetic.

2. Second, the guaranteed integration process has been introduced in Chapter 3 and is used to compute the enclosure of ODEs and to solve initial value problems. In GDODynS this module is based on Vnode-LP [Nedialkov et al., 2001] because it provides both relevant enclosures and fast computations.

3. Third, the problematic of solving piecewise ODEs have been detailed in Chapter 5 with the required assumptions is dealt with. A specific guaranteed integration scheme has been developed that is based on a new Picard operator and a theorem. In GDODynS this module is based on the module that computes guaranteed integration of ODEs.

4. Fourth, the contraction methods have been studied in Chapter 4 using Contractor Programming in order to reduce the domains of the variables and exploit algebraic and functional constraints inside the models. This module requires the guaranteed arithmetics module to provide contractors from algebraic constraints and the guaranteed integration modules that are used inside the differential contractors.

5. Fifth, all of the previous modules are combined within a global and deterministic optimization algorithm in Chapter 6 to optimize models presented in the first chapter.

At each step of this thesis, a new decision is taken that improves the quality of the solution or the speed of the optimization process. Usually these two aspects are divergent, for example it is the case for the piecewise evaluation method; however it has to be noted that, at some points, when the quality of the computation is improved, the global computation time that is required is decreased as well. This is what happens during the guaranteed integration of the ODEs in which the quality of the enclosure allows larger stepsizes, or in the optimization algorithm in which the quality of the computation improves the upper bound.

It has also been stated that the computation with rigorous bound can harm the guarantee of the result granted by the optimization process.

The work presented in this thesis opens a lot of perspectives that can be studied and developed in the future.

- Interval arithmetic is used to compute rigorous bounds of operations, but it is limited. Therefore, other guaranteed arithmetics could be used such as the Affine Arithmetic [De Figueiredo and Stolfi, 2004] and applied on guaranteed integration of ODEs [Alexandre dit Sandretto and Chapoutot, 2016b] and global optimization algorithms [Messine, 2002].

- The tubes may be represented differently in order to limit the overapproximation of the solution that happens on transition phases during the guaranteed integration of piecewise-ODEs. For example, the global enclosures of tubes could be represented with polynomials.

- The IBBA requires a lot of computations and limits the size of the problems that can be handled. That is why other methods such as the evolutionary methods can be coupled with the IBBA so as to compute and update the upper bound [Vanaret, 2015].

- The loss of guarantee is an important issue and some works should be dedicated to how one should deal with this problem. One might also deal with the computation of an indicator that would show how trustworthy the feasibility of a solution is, it could be computed with the *Hausdorff* metric. At last, one could use a complement of the interval arithmetic that rounds the result of operation inward instead of outward.

- The theorems and the algorithms that are used to perform the guaranteed integration of piecewise-ODEs should be extended to generic hybrid systems and implemented into GDODynS.

- The optimization tool GDODynS has been thought to be modular, and the contractor programming module allow the creation of new contractors. A specific contractor could be implemented and paired with external libraries such as Choco [Cambazard et al., 2006] in order to extend the model with integer variables and new filtering algorithms.

Some of the contributions presented in this thesis were made public in national [Joudrier and Hadj-Hamou, 2014] [Joudrier and Hadj-Hamou, 2016] and international conferences [Joudrier and Hadj-Hamou, 2015]. Alongside the work presented in this thesis, an other research project has been studied in collaboration with Florence Thiard on a rolling stock unit management problem. This problem [Ramond and Nicolas, 2014] has been formulated by the SNCF within the scope of the ROADEF/EURO international challenge 2014. The results of the developed methods are presented in the appendix A and has been published [Joudrier and Thiard, 2017].

Bibliography

[Ábrahám and Schupp, 2012] Ábrahám, E. and Schupp, S. (2012). Modeling and analysis of hybrid systems.

[Al-Abedeen and Arora, 1978] Al-Abedeen, A. Z. and Arora, H. (1978). A global existence and uniqueness theorem for ordinary differential equations of generalized order. *Canad. Math. Bull*, 21(3):271–276.

[Alexandre dit Sandretto and Chapoutot, 2016a] Alexandre dit Sandretto, J. and Chapoutot, A. (2016a). DynIBEX: une boîte à outils pour la vérification des systèmes cyber-physiques. In *Approches Formelles dans l'Assistance au Développement de Logiciels*, Besançon, France.

[Alexandre dit Sandretto and Chapoutot, 2016b] Alexandre dit Sandretto, J. and Chapoutot, A. (2016b). Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing electronic edition*, 22.

[Alur et al., 1993] Alur, R., Courcoubetis, C., Henzinger, T. A., and Ho, P. H. (1993). *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*, pages 209–229. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Araya et al., 2010] Araya, I., Trombettoni, G., and Neveu, B. (2010). Making adaptive an interval constraint propagation algorithm exploiting monotonicity. *Principles and Practice of Constraint Programming–CP 2010*, pages 61–68.

[Araya et al., 2012] Araya, I., Trombettoni, G., and Neveu, B. (2012). A contractor based on convex interval taylor. In *CPAIOR*, pages 1–16. Springer.

[Araya et al., 2014] Araya, I., Trombettoni, G., Neveu, B., and Chabert, G. (2014). Upper bounding in inner regions for global optimization under inequality constraints. *Journal of Global Optimization*, 60(2):145–164.

[Aubry et al., 2013] Aubry, C., Desmare, R., and Jaulin, L. (2013). Loop detection of mobile robots using interval analysis. *Automatica*, 49(2):463–470.

[Balafrej et al., 2014] Balafrej, A., Bessiere, C., Bouyakhf, E.-H., and Trombettoni, G. (2014). Adaptive singleton-based consistencies. In *AAAI*, volume 14, pages 2601–2607.

[Benhamou et al., 1999] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F. (1999). Revising hull and box consistency. In *Logic Programming: Proceedings of the 1999 International Conference on Logic Programming*, pages 230–244. MIT press.

[Benhamou and Granvilliers, 2006] Benhamou, F. and Granvilliers, L. (2006). Continuous and interval constraints. *Foundations of Artificial Intelligence*, 2:571–603.

[Bennaceur and Affane, 2001] Bennaceur, H. and Affane, M.-S. (2001). Partition-k-ac: an efficient filtering technique combining domain partition and arc consistency. In *International Conference on Principles and Practice of Constraint Programming*, pages 560–564. Springer.

[Berz and Makino, 1998] Berz, M. and Makino, K. (1998). Verified integration of odes and flows using differential algebraic methods on high-order taylor models. *Reliable Computing*, 4:361–369.

[Bouissou et al., 2012] Bouissou, O., Mimram, S., and Chapoutot, A. (2012). Hyson: Set-based simulation of hybrid systems. In *Rapid System Prototyping (RSP), 2012 23rd IEEE International Symposium on*, pages 79–85. IEEE.

[Cambazard et al., 2006] Cambazard, H., Jussien, N., Laburthe, F., and Rochart, G. (2006). The choco constraint solver. In *INFORMS Annual meeting*.

[Caprani and Madsen, 1980] Caprani, O. and Madsen, K. (1980). Mean value forms in interval analysis. *Computing*, 25(2):147–154.

[Carbonnel et al., 2014] Carbonnel, C., Trombettoni, G., Vismara, P., and Chabert, G. (2014). Q-intersection algorithms for constraint-based robust parameter estimation. In *AAAI*, pages 2630–2636.

[Cartwright and Piro, 1992] Cartwright, J. H. E. and Piro, O. (1992). The dynamics of runge-kutta methods. *International Journal of Bifurcation and Chaos*, 2:1–4.

[Chabert, 2007] Chabert, G. (2007). Ibex, an interval-based explorer. http://www.ibex-lib.org/.

[Chabert and Jaulin, 2009a] Chabert, G. and Jaulin, L. (2009a). Contractor programming. *Artificial Intelligence*, 173(11):1079 – 1100.

[Chabert and Jaulin, 2009b] Chabert, G. and Jaulin, L. (2009b). Hull consistency under monotonicity. In *International Conference on Principles and Practice of Constraint Programming*, pages 188–195. Springer.

[Chen et al., 2012] Chen, X., Abraham, E., and Sankaranarayanan, S. (2012). Taylor model flowpipe construction for non-linear hybrid systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 183–192. IEEE.

[Cheng et al., 1982] Cheng, K.-S., Hsu, S.-B., and Lin, S.-S. (1982). Some results on global stability of a predator-prey system. *Journal of Mathematical Biology*, 12(1):115–126.

[Coddington, 2012] Coddington, E. A. (2012). *An introduction to ordinary differential equations*. Courier Corporation.

[Coello et al., 2002] Coello, C. A. C., Van Veldhuizen, D. A., and Lamont, G. B. (2002). *Evolutionary algorithms for solving multi-objective problems*, volume 242. Springer.

[Corliss and Rihm, 1996] Corliss, G. F. and Rihm, R. (1996). Validating an a priori enclosure using high-order taylor series. *Mathematical Research*, 90:228–238.

[Cruz and Barahona, 2003] Cruz, J. and Barahona, P. (2003). Constraint satisfaction differential problems. In Rossi, F., editor, *CP*, volume 2833 of *Lecture Notes in Computer Science*, pages 259–273. Springer.

[De Figueiredo and Stolfi, 2004] De Figueiredo, L. H. and Stolfi, J. (2004). Affine arithmetic: concepts and applications. *Numerical Algorithms*, 37(1):147–158.

[Debruyne and Bessiere, 1997] Debruyne, R. and Bessiere, C. (1997). Some practicable filtering techniques for the constraint satisfaction problem. In *In Proceedings of IJCAI'97*. Citeseer.

[Deville et al., 2000] Deville, Y., Janssen, M., and Hentenryck, P. V. (2000). Consistency techniques in ordinary differential equations.

[Eijgenraam, 1981] Eijgenraam, P. (1981). The solution of initial value problems using interval arithmetic: Formulation and analysis of an algorithm. *MC Tracts*, 144:1–185.

[Floyd, 1962] Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345.

[Frehse et al., 2011] Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., and Maler, O. (2011). SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer.

[Goldsztejn et al., 2011] Goldsztejn, A., Hayes, W., and Collins, P. (2011). Tinkerbell is chaotic. *SIAM Journal on Applied Dynamical Systems*, 10(4):1480–1501.

[Goualard, 2007] Goualard, F. (2007). Interval extensions of multivalued inverse functions.

[Hansen and Walster, 2003] Hansen, E. and Walster, G. W. (2003). *Global optimization using interval analysis: revised and expanded*, volume 264. CRC Press.

[Hansen, 1968] Hansen, R. J. (1968). Interval arithmetic as a closed arithmetic system on a computer. *Jet Propulsion Laboratory Report*, 197.

[Hastings and Murray, 1975] Hastings, S. and Murray, J. (1975). The existence of oscillatory solutions in the field-noyes model for the belousov-zhabotinskii reaction. *SIAM Journal on Applied Mathematics*, 28(3):678–688.

[Henrikson, 1999] Henrikson, J. (1999). Completeness and total boundedness of the hausdorff metric. *MIT Undergraduate Journal of Mathematics*, 1:69–80.

[Herrero et al., 2012] Herrero, P., Georgiou, P., Toumazou, C., Delaunay, B., Jaulin, L., et al. (2012). An efficient implementation of sivia algorithm in a high-level numerical programming language. *Reliable Computing*, 16:224–233.

[Householder, 1958] Householder, A. S. (1958). Unitary triangularization of a nonsymmetric matrix. *Journal of the Association for Computing Machinery*, 5(4):339–342.

[Hudak et al., 2007] Hudak, P., Hughes, J., Peyton Jones, S., and Wadler, P. (2007). A history of haskell: being lazy with class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 12–1. ACM.

[Jackson and Nedialkov, 2002] Jackson, K. R. and Nedialkov, N. S. (2002). Some recent advances in validated methods for ivps for odes. *Applied Numerical Mathematics*, 42(1):269–284.

[Jaulin, 2002] Jaulin, L. (2002). Nonlinear bounded-error state estimation of continuous-time systems. *Automatica*, 38(6):1079–1082.

[Jaulin et al., 2001] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001). Applied interval analysis: with examples in parameter and state estimation, robust control and robotics, vol. 1.

[Jaulin and Walter, 1993] Jaulin, L. and Walter, E. (1993). Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064.

[Johnsson, 1984] Johnsson, T. (1984). *Efficient compilation of lazy evaluation*, volume 19. ACM.

[Joudrier, 2013] Joudrier, H. (2013). Propagation de contraintes ode par calcul d'intervalle pour l'optimisation globale. Master's thesis, Université Joseph Fourier.

[Joudrier and Hadj-Hamou, 2014] Joudrier, H. and Hadj-Hamou, K. (2014). Propagation garantie de contraintes ODE par morceaux pour l'optimisation globale. In *ROADEF - 15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision*, Bordeaux, France. Société française de recherche opérationnelle et d'aide à la décision. Prix du jeune chercheur.

[Joudrier and Hadj-Hamou, 2015] Joudrier, H. and Hadj-Hamou, K. (2015). Guaranteed Global Deterministic Optimization and Constraint Programming for Complex Dynamic Problems. In *21th International Conference on Principles and Practice of Constraint Programming*, Doctoral Program Proceedings, Cork, Ireland.

[Joudrier and Hadj-Hamou, 2016] Joudrier, H. and Hadj-Hamou, K. (2016). Optimisation globale déterministe garantie sous contraintes différentielles par morceaux. In *17ème Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2016)*, Compiègne, France.

[Joudrier and Thiard, 2017] Joudrier, H. and Thiard, F. (2017). A greedy approach for a rolling stock management problem using multi-interval constraint propagation. *Annals of Operations Research.*

[Jussien et al., 2008] Jussien, N., Rochart, G., and Lorca, X. (2008). Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10.

[Kahan, 1968] Kahan, W. (1968). A more complete interval arithmetic. *Lecture notes for a summer course at the University of Michigan.*

[Knüppel, 1994] Knüppel, O. (1994). Profil/bias—a fast interval library. *Computing*, 53(3-4):277–287.

[Kone et al., 1993] Kone, A., Nogarede, B., and Mazenc, M. L. (1993). Le dimensionnement des actionneurs électriques: un problème de programmation non linéaire. *Journal de Physique III*, 3(2):285–301.

[Kurzhanskiĭ and Vályi, 1997] Kurzhanskiĭ, A. and Vályi, I. (1997). *Ellipsoidal calculus for estimation and control.* Nelson Thornes.

[Le Bars et al., 2012] Le Bars, F., Sliwka, J., Jaulin, L., and Reynet, O. (2012). Set-membership state estimation with fleeting data. *Automatica*, 48(2):381–387.

[Lerch et al., 2006] Lerch, M., Tischler, G., Gudenberg, J. W. V., Hofschuster, W., and Krämer, W. (2006). Filib++, a fast interval library supporting containment computations. *ACM Transactions on Mathematical Software (TOMS)*, 32(2):299–324.

[Lhomme, 1993] Lhomme, O. (1993). Consistency techniques for numeric csps. In *IJCAI*, volume 93, pages 232–238.

[Lin and Stadtherr, 2006a] Lin, Y. and Stadtherr, M. A. (2006a). Deterministic global optimization for parameter estimation of dynamic systems. *Industrial and Engineering Chemistry Research*, 45(25):8438–8448.

[Lin and Stadtherr, 2006b] Lin, Y. and Stadtherr, M. A. (2006b). Validated solution of initial value problems for odes with interval parameters. In *Proceedings of 2nd NSF Workshop on Reliable Engineering Computing.*

[Lin and Stadtherr, 2007] Lin, Y. and Stadtherr, M. A. (2007). Guaranteed state and parameter estimation for nonlinear continuous-time systems with bounded-error measurements. *Industrial and Engineering Chemistry Research*, 46(22):7198–7207.

[Linnainmaa, 1970] Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7.

[Lohner, 1988] Lohner, R. (1988). *Einschließung der Lösung gewöhnlicher Anfangs-und Randwertaufgaben und Anwendungen.* PhD thesis, Universität Karlsruhe.

[Lohner, 1987] Lohner, R. J. (1987). Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic, Scientific Computation and Programming Languages*, pages 255–286.

[Lohner, 1992] Lohner, R. J. (1992). *Computation of Guaranteed Enclosures for the Solutions of Ordinary Initial and Boundary Value Problem.* Clarendon Press.

[Madsen and Jensen, 1999] Madsen, A. L. and Jensen, F. V. (1999). Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1):203–245.

[Makino and Berz, 2011] Makino, K. and Berz, M. (2011). Suppression of the wrapping effect by Taylor model-based verified integrators: Long-term stabilization by preconditioning. *International Journal of Differential Equations and Applications*, 10(4).

[Martin et al., 2002] Martin, R., Shou, H., Voiculescu, I., Bowyer, A., and Wang, G. (2002). Comparison of interval methods for plotting algebraic curves. *Computer Aided Geometric Design*, 19(7):553–587.

[Mazhoud, 2014] Mazhoud, I. (2014). *Contribution to optimization in product's preliminary design.* Theses, Université de Grenoble.

[Messine, 1997] Messine, F. (1997). *Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution de problèmes avec contraintes.* PhD thesis, Toulouse, INPT.

[Messine, 2002] Messine, F. (2002). Extensions of affine arithmetic: Application to global optimization. *Journal of Universal Computer Science*, 8(11):992–1015.

[Messine, 2004] Messine, F. (2004). Deterministic global optimization using interval constraint propagation techniques. *RAIRO-OR*, 38(4):277–294.

[Messine, 2005] Messine, F. (2005). A deterministic global optimization algorithm for design problems.

[Messine et al., 2001] Messine, F., Monturet, V., and Nogarède, B. (2001). An interval branch and bound method dedicated to the optimal design of piezoelectric actuators. *Mathematics and Computers in Sciences and Engineering*, pages 174–180.

[Messine et al., 1998] Messine, F., Nogarede, B., and Lagouanelle, J.-L. (1998). Optimal design of electromechanical actuators: a new method based on global optimization. *IEEE transactions on magnetics*, 34(1):299–308.

[Moore, 1966] Moore, R. E. (1966). *Interval Analysis.* Prentice-Hall series in automatic computation. Prentice-Hall.

[Moore, 1979] Moore, R. E. (1979). *Methods and applications of interval analysis*, volume 2. SIAM.

[Mukaidono, 1986] Mukaidono, M. (1986). Regular ternary logic functions ternary logic functions suitable for treating ambiguity. Technical Report 2, Washington, DC, USA.

[Nedialkov and Jackson, 1999] Nedialkov, N. S. and Jackson, K. R. (1999). An interval hermite-obreschkoff method for computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation. *Reliable Computing*, 5(3):289–310.

[Nedialkov and Jackson, 2001] Nedialkov, N. S. and Jackson, K. R. (2001). *Perspectives on Enclosure Methods*, chapter A New Perspective on the Wrapping Effect in Interval Methods for Initial Value Problems for Ordinary Differential Equations, pages 219–263. Springer Vienna, Vienna.

[Nedialkov et al., 1999] Nedialkov, N. S., Jackson, K. R., and Corliss, G. F. (1999). Validated solutions of Initial Value Problems for Ordinary Differential Equations. *Applied Mathematics and Computation*, 105(1):21–68.

[Nedialkov et al., 2001] Nedialkov, N. S., Jackson, K. R., and Pryce, J. D. (2001). An effective high-order interval method for validating existence and uniqueness of the solution of an IVP for an ODE. *Reliable Computing*, 7:449–465.

[Neumaier, 1990] Neumaier, A. (1990). *Interval methods for systems of equations*, volume 37. Cambridge university press.

[Neveu and Trombettoni, 2013] Neveu, B. and Trombettoni, G. (2013). Adaptive constructive interval disjunction. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 900–906. IEEE.

[Neveu et al., 2015] Neveu, B., Trombettoni, G., and Araya, I. (2015). Adaptive constructive interval disjunction: algorithms and experiments. *Constraints*, 20(4):452–467.

[Neveu et al., 2016] Neveu, B., Trombettoni, G., and Araya, I. (2016). Node selection strategies in interval branch and bound algorithms. *Journal of Global Optimization*, 64(2):289–304.

[Ninin, 2015] Ninin, J. (2015). Global optimization based on contractor programming: An overview of the ibex library. In *International Conference on Mathematical Aspects of Computer and Information Sciences*, pages 555–559. Springer.

[Papamichail and Adjiman, 2004] Papamichail, I. and Adjiman, C. S. (2004). Global optimization of dynamic systems. *Computers and Chemical Engineering*, 28(3):403–415.

[Raıssi et al., 2004] Raıssi, T., Ramdani, N., and Candau, Y. (2004). Set membership state and parameter estimation for systems described by nonlinear differential equations. *Automatica*, 40(10):1771–1777.

[Ramdani et al., 2006] Ramdani, N., Meslem, N., Raïssi, T., and Candau, Y. (2006). Set-membership identification of continuous-time systems. In *Proceedings of 14th IFAC Symposium on System Identification*.

[Ramond and Nicolas, 2014] Ramond, F. and Nicolas, M. (2014). Trains don't vanish ! ROADEF EURO 2014 Challenge Problem Description. 39 pages.

[Raskin, 2011] Raskin, J.-F. (2011). Reachability problems for hybrid automata. In *International Workshop on Reachability Problems*, pages 28–30. Springer.

[Ratschek, 1985] Ratschek, H. (1985). Inclusion functions and global optimization. *Mathematical Programming*, 33(3):300–317.

[Rogers et al., 1991] Rogers, D. F., Plante, R. D., Wong, R. T., and Evans, J. R. (1991). Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39(4):553–582.

[Sam-Haroud and Faltings, 1996] Sam-Haroud, D. and Faltings, B. (1996). Consistency techniques for continuous constraints. *Constraints*, 1(1-2):85–118.

[Sans et al., 2016] Sans, O., Coletta, R., and Trombettoni, G. (2016). An interval filtering operator for upper and lower bounding in constrained global optimization. In *Tools with Artificial Intelligence (ICTAI), 2016 IEEE 28th International Conference on*, pages 218–225. IEEE.

[Teschl, 2012] Teschl, G. (2012). *Ordinary differential equations and dynamical systems*, volume 140. American Mathematical Society Providence.

[Trombettoni et al., 2011] Trombettoni, G., Araya, I., Neveu, B., and Chabert, G. (2011). Inner regions and interval linearizations for global optimization. In *AAAI*.

[Trombettoni and Chabert, 2007] Trombettoni, G. and Chabert, G. (2007). Constructive interval disjunction. *Principles and Practice of Constraint Programming–CP 2007*, pages 635–650.

[Trombettoni et al., 2010] Trombettoni, G., Papegay, Y., Chabert, G., and Pourtallier, O. (2010). A box-consistency contractor based on extremal functions. In *CP*, pages 491–498. Springer.

[Tsang, 2014] Tsang, E. (2014). *Foundations of constraint satisfaction: the classic text*. BoD–Books on Demand.

[Vanaret, 2015] Vanaret, C. (2015). *Hybridization of evolutionary algorithms and interval-based methods for optimizing difficult problems*. Theses, INP Toulouse. Prix math/info de l'académie des sciences de Toulouse 2015.

[Walter et al., 1996] Walter, E., Norton, J., Piet-Lahanier, H., and Milanese, M. (1996). *Bounding Approaches to System Identification.* Perseus Publishing.

[Wengert, 1964] Wengert, R. E. (1964). A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464.

ROADEF/EURO Challenge 2014 A

A greedy approach for a rolling stock management problem using multi-interval constraint propagation

In this article we present our contribution to the Rolling Stock Unit Management problem proposed for the ROADEF/EURO Challenge 2014. We propose a greedy algorithm to assign trains to departures. Our approach relies on a routing procedure using multi-interval constraint propagation to compute the individual schedules of trains within the railway station. This algorithm allows to build an initial solution, satisfying a significant subset of departures.

A.1 Introduction

In this paper, we present our approach of the Rolling Stock Unit Management problem presented in the ROADEF/EURO Challenge 2014 [Ramond and Nicolas, 2014]. This problem combines in a single formulation various sub-problems of rolling stock management, such as assignment of trains to departures, platform assignment, routing inside the station, planning of maintenance operations... We propose a greedy algorithm to build an initial solution, allowing an incomplete coverage of arrivals and departures. This solution could serve as a basis for an optimization algorithm. Compared to the proposed problem, we make some simplifications by forbidding maintenance, junction and disjunction operations (note that train convoys might be used as such), which limits the number of coverable departures. Our solution is based on a greedy progressive assignment algorithm to assign arrivals (and corresponding trains) to departures, and a routing algorithm using multi-interval constraint propagation to prevent conflicts with already scheduled trains while keeping as much flexibility as possible.

The rest of this paper is organized as follows: Section A.2 presents a simplified model of the rolling stock management problem presented in the ROADEF/EURO Challenge

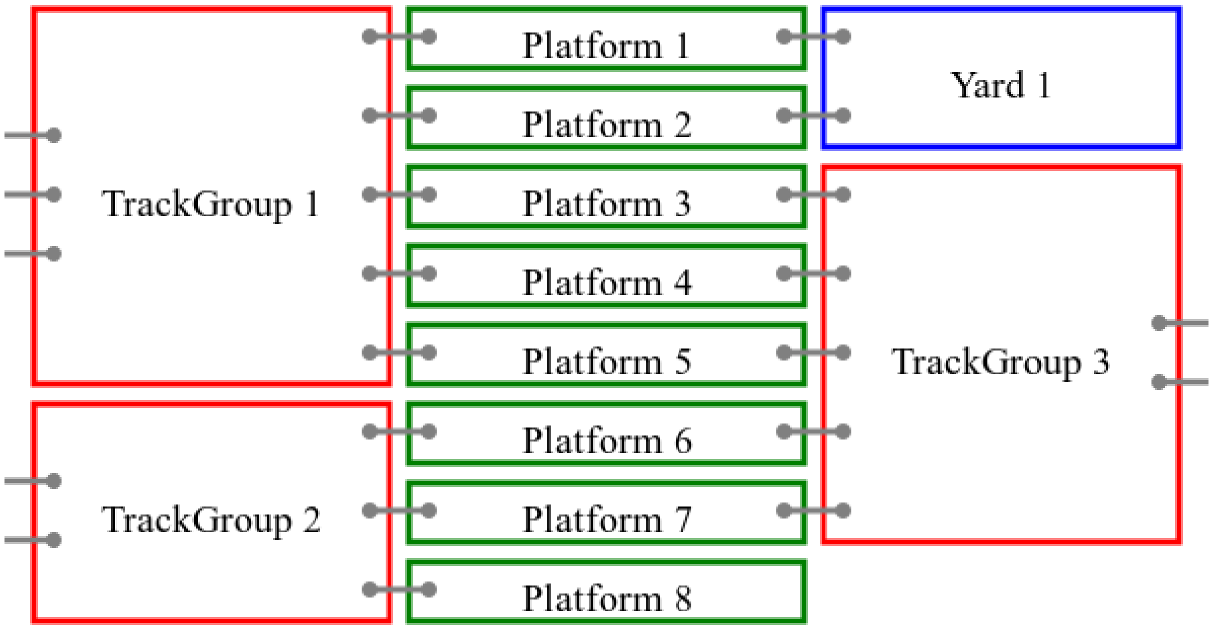


Figure A.1: Example instance: railway station

2014. Section A.3 describes the pre-processing phase and the model used to represent the structure of the station. Section A.4 details the multi-interval routing algorithm, while Section A.5 describes the general assignment procedure. Note that the assignment problem is not treated as a whole, separately from the routing problem, but progressively. Finally Section A.6 presents an overview of the results and some perspectives. In the following, we will use the concepts and notations introduced by [Ramond and Nicolas, 2014].

A.2 Simplified model

The original formulation, as presented by [Ramond and Nicolas, 2014], aimed at treating the rolling stock management problem as a whole, integrating constraints and costs of various nature. Focusing mainly on the routing constraints, we chose to overlook some aspects of the problem in order to work on a simplified model while still providing valid solutions.

The original industrial problem consists in handling trains within a station over a biweekly planning horizon, by deciding their route and schedule inside the station, and assigning them to compatible departures. A station is composed of resources of several types (track-groups, single tracks and yards for parking, maintenance facilities for performing maintenance, platforms for arrival and departures), linked by gates. Routing of the trains inside the station must respect the length and capacity of the resources, as well as the trains' order on the resources and train/resource compatibility constraints. To be allowed to circulate, trains must be submitted to regular maintenance. Therefore, a train might be assigned to a departure only if it has enough time and mileage left before a maintenance is required for the journey. Maintenance operations can be performed in the stations on appropriate resources, at a cost. Some arrivals (joint arrivals) are composed of several trains joined together; likewise, some departures (joint departures) must be satisfied by several joint trains. Convoys of several joint trains might be formed (respectively separated) by performing junction and disjunction operations.

The two main simplifications that were made are described below. Their main consequence is to avoid modifying characteristics of the trains in the solution.

- Not performing maintenance operations. This choice simplifies the assignment decision process: the compatibility of a given train with a given departure depends only on the instance data (including the time / distance remaining before maintenance), and not on the choice to perform maintenance on it.

- Not performing junction nor disjunction operations. That way, trains can be treated as immutable *convoys* from their arrivals to their departure. A *convoy* might be formed of a single train, or of several trains coming from a joint-arrival, which might then satisfy a joint-departure.

In this section we first extend the notations defined by [Ramond and Nicolas, 2014]. Then, we use these new notations to formalize our simplified model.

A.2.1 Definitions

A.2.1.1 Definition : Convoys

A convoy v is defined by a set of trains t in $\mathcal{T}$. We note $\mathcal{V}$ the set of all the convoys.

$$\mathcal{V} := P(\mathcal{T}) \tag{A.1}$$

where P is the power set operator. The length and size of a convoys v in $\mathcal{V}$ can be accessed with:

$$size(v) = \#v \tag{A.2}$$
$$length(v) = \sum_{t \in v} length(t) \tag{A.3}$$

We also define $\mathcal{V}^+$, the set of convoys actually used in the solution.

A.2.1.2 Definitions : About arrivals and departures

As our model treats convoys and not individual trains, joint arrivals (respectively joint departures) can be treated as one single arrival (respectively departure). Thus, we work with a set of *extended arrivals* (respectively departures), defined in this section. An element of this set is either a unitary arrival (an arrival which is not a member of a joint arrival), or a joint arrival (a set of arrivals joint together, treated as a single element).

Let $\mathcal{U}_{arr}$ be the set of unitary arrival in opposition with $\mathcal{J}_{arr}$ the set of joint-arrivals (and similarly the set of unitary departures).

$$\mathcal{U}_{arr} := \{a \in \mathcal{A} \mid \forall \mathcal{B} \in \mathcal{J}_{arr}, a \notin \mathcal{B}\} \tag{A.4}$$

$$\mathcal{U}_{dep} := \{d \in \mathcal{D} \mid \forall \mathcal{B} \in \mathcal{J}_{dep}, d \notin \mathcal{B}\} \tag{A.5}$$

Now, we use the sets $\mathcal{J}_{arr}$ and $\mathcal{U}_{arr}$ to build a new set $Ext\mathcal{A}rr$ called set of all extended arrivals (and the set $Ext\mathcal{D}ep$ with $\mathcal{J}_{dep}$ and $\mathcal{U}_{dep}$).

$$Ext\mathcal{A}rr := J_{arr} \cup \{\{a\} \mid a \in \mathcal{U}_{arr}\} \tag{A.6}$$

$$Ext\mathcal{D}ep := J_{dep} \cup \{\{d\} \mid d \in \mathcal{U}_{dep}\} \tag{A.7}$$

A.2.1.3 Definitions : compatibility

For each convoy $v \in \mathcal{V}$ and departure in $d \in Ext\mathcal{D}ep$, a value $Comp(v, d) \in \{0, 1\}$ is defined. If $Comp(v, d) = 1$, v and d are said to be compatible. This value takes into account several parameters of the initial formulation such as size, categories, time and distance remaining before maintenance, and so on.

Similarly, a value $Comp(v, a) \in \{0, 1\}$ is defined to express compatibility between a convoy $v \in \mathcal{V}$ and an arrival $a \in Ext\mathcal{A}rr$.

Similarly, for each convoy $v \in \mathcal{V}$ and resource $r \in \mathcal{R}$, $Comp(v, a) \in \{0, 1\}$ expresses the compatibility of all trains of v with the resource r.

A.2.1.4 Definition : Connectors

Connectors are couples of gates, representing transition between resources. Using this concept, we can model the station as a directed graph.

The set of connectors noted $\mathcal{CO}$ is defined by

$$\mathcal{CO} = \left\{ (g, g') \in \mathcal{G}_r \times \mathcal{G}_{r'} \;\middle|\; \begin{array}{c} neigh_g = g' \\ neigh_{g'} = g \end{array} \right\} \tag{A.8}$$

Then a connector $co \in \mathcal{CO}$ is a couple of gates (g, g') such that a convoy can leave the resource r ($g \in \mathcal{G}_r$) by the gate g to enter in the resource r' by g' ($g' \in \mathcal{G}_{r'}$).

It is important to note that the connector (g, g') is not equivalent to the connector (g', g). The first one means an exit by g and an entry by g' whereas the second one means an exit by g' and an entry by g.

The train station can then be modeled by a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where the vertices are connectors and the arcs are resources (Fig. A.2). This allows to represent reverse and non-reverse resources.

Some values are associated with each edge of the graph $\mathcal{G}$ such as the $minTrTime$ and the $maxTrTime$ which are the minimal and the maximal travelling time to go from a connector to an other one through a resource. These parameter values aggregate several parameters from the original problem formulation, depending on the resources type, such as $trTime$ for trackgroups, $maxDwellTime$ for platforms, $revTime$ for the reverse operations on single tracks, platform, yards...

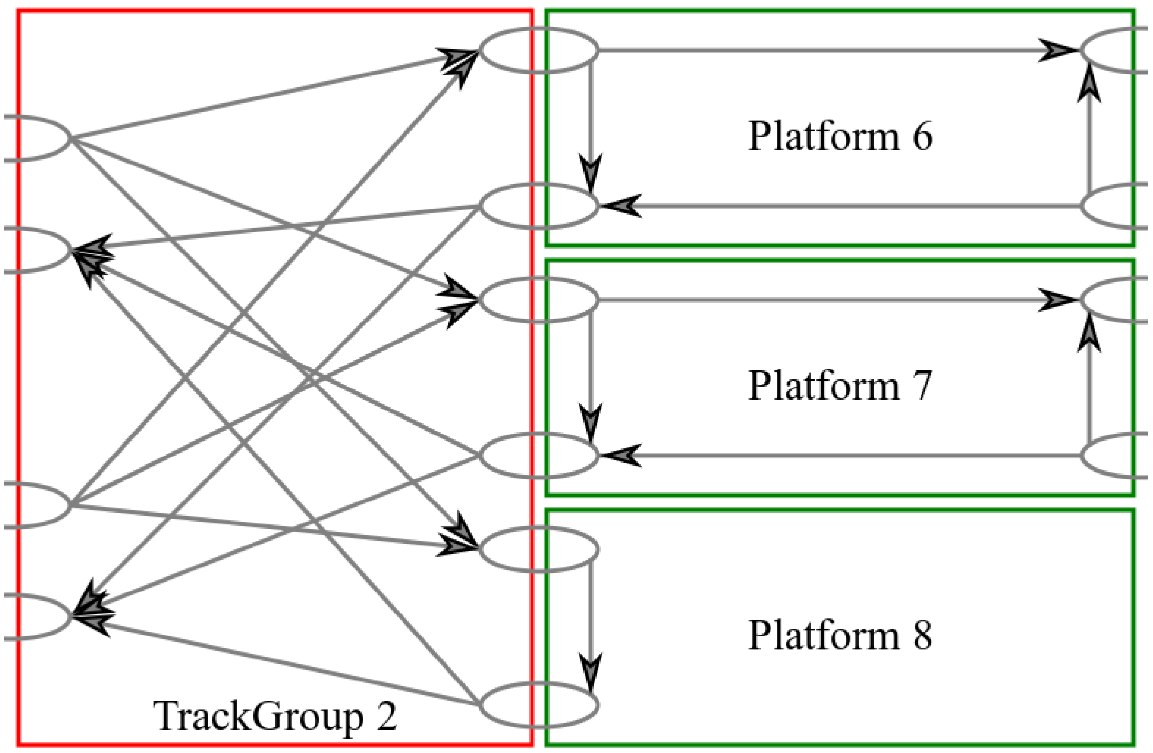


Figure A.2: Directed Graph Modelisation

A.2.1.5 Definition : Path through the station

A path through the station is defined by an alternate sequence of connectors in $\mathcal{CO}$ and resources in $\mathcal{R}$.

$$p := (co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_n) \tag{A.9}$$

The size of the path p is accessible via $size(p)$ and returns the number of resources crossed along the path.

$$size((co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_n)) = n \tag{A.10}$$

An n-sized path going through resources $r_0 \dots r_{n-1}$ requires $n+1$ connectors $co_0 \dots co_n$.

A.2.1.6 Schedule of a path

The schedule of a path p, noted $sched_p$, is a tuple of instants t in the horizon $\mathcal{H}$ which are in bijection with the connectors of the path. These values describe the time of transition from a resource to the other. Let p be a path composed by n resources (Eq. A.9). Then a schedule for p is defined by

$$sched_p = (t_0, t_1, t_2 \dots t_{n-2}, t_{n-1}, t_n) \tag{A.11}$$

where t_i in $sched_p$ is paired with co_i in p, meaning that at t_i the connector co_i is used to leave resource r_{i-1} and resource r_i.

A.2.2 Decision Variables

A.2.2.1 Arrival and Departure assignments

A convoy may come from an arrival (all trains composing the convoy enter the system as elements of this arrival), and be assigned to a departure (all trains composing the convoy leave the system satisfying elements of this departure).

$$\forall v \in \mathcal{V}, \quad \begin{cases} \forall a \in Ext\mathcal{A}rr,\ \mathit{from}(v,a) \in \{0,1\} \\ \forall d \in Ext\mathcal{D}ep,\ \mathit{assigned}(v,d) \in \{0,1\} \end{cases} \tag{A.12}$$

A.2.2.2 Paths and schedules of convoys

A convoy v in $\mathcal{V}$ may be routed, thus associated with a path $\mathit{path}(v)$ which is paired with a schedule $\mathit{sched}(v)$, of size n_v.

$$\mathit{path}(v) = (co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_{n_v}) \tag{A.13}$$

$$\mathit{sched}(v) = (t_0, t_1, t_2 \dots t_{n-2}, t_{n-1}, t_{n_v}) \tag{A.14}$$

A.2.3 Constraints

A.2.3.1 Convoys paired with Arrivals and Departures

Recall that $\mathcal{V}^+$ is the set of all convoys scheduled in the solution. Let v be a convoy in $\mathcal{V}$. If v is scheduled in the solution then v is paired with an arrival a in $Ext\mathcal{A}rr$ and a departure d in $Ext\mathcal{D}ep$.

$$v \in \mathcal{V}^+ \Rightarrow \begin{cases} \sum_{a \in Ext\mathcal{A}rr} from(v,a) = 1 \\ \sum_{d \in Ext\mathcal{D}ep} assigned(v,d) = 1 \end{cases} \tag{A.15}$$

Otherwise, v is not an element of $\mathcal{V}^+$ and there is no arrival nor departure assigned to it.

$$v \notin \mathcal{V}^+ \Rightarrow \begin{cases} \forall a \in Ext\mathcal{A}rr, from(v,a) = 0 \\ \forall d \in Ext\mathcal{D}ep, assigned(v,d) = 0 \end{cases} \tag{A.16}$$

A convoy v cannot be assigned to an incompatible arrival a or departure d. In other words :

$$Comp(v,a) = 0 \Rightarrow from(v,a) = 0 \tag{A.17}$$

$$Comp(v,d) = 0 \Rightarrow assigned(v,d) = 0 \tag{A.18}$$

A.2.3.2 Usage of resources

The usage in term of capacity and length has to be respected. Let v be a convoy, with the schedule $sched(v)$ associated to the path $path(v)$. For each convoy v in $\mathcal{V}^+$ and each instant t in $\mathcal{H}$, we define $use(v,t)$ the resource r in $\mathcal{R}$ used by the convoy v at t.

$$use(v,t) = \begin{cases} r_i \in path(v) & \textbf{if } \exists t_i, t_{i+1} \in sched_v, t_i \leq t < t_{i+1} \\ \emptyset & \textbf{otherwise} \end{cases} \tag{A.19}$$

Then we define the value $use(v,t,r)$, which is 1 if and only if the resource r is used at time t in the schedule $sched(v)$ of the path $path(v)$ associated to the convoy v.

$$use(v,t,r) = \begin{cases} 1 & \textbf{if } use(v,t) = r \\ 0 & \textbf{otherwise} \end{cases} \tag{A.20}$$

A.2.3.3 Length and capacity constraints

$$\forall r \in \mathcal{R}, \forall t \in \mathcal{H}, \begin{cases} \sum_{v \in \mathcal{V}^+} (use(v,t,r) * size(v)) \leq capa(r) \\ \sum_{v \in \mathcal{V}^+} (use(v,t,r) * length(v)) \leq length(r) \end{cases} \tag{A.21}$$

A.2.3.4 Valid path

Let v be a convoy in $\mathcal{V}^+$, and $path(v)$ the associated path, of size n_v. To be valid, the path, defined as in (Eq. A.13), followed by v must satisfy the following constraints.

$$co_0 = (\emptyset, g_{initial}) \tag{A.22}$$

$$co_{n_v} = (g_{final}, \emptyset) \tag{A.23}$$

This means the convoy enters the station via the first connector of the path, and leaves it via the last connector of the path.

Let us now consider the i-th connector co_i, composed of two gates g_{out} and g_{in}. By definition of connectors, $neighg_{out} = g_{in}$ and $neigh(g_{in}) = g_{out}$. Additionally, for the path to be valid, each resource must be consistent with the adjacent connectors:

$$r_{g_{out}} = r_{i-1} \tag{A.24}$$

$$r_{g_{in}} = r_i \tag{A.25}$$

A convoy v cannot use an incompatible resource r:

$$\forall r \in \mathcal{R}, Comp(v, r_i) = 0 \Rightarrow r_i \notin path(v) \tag{A.26}$$

Since v is a convoy of the solution, it is assigned to an arrival a in $Ext\mathcal{A}rr$ as well as a departure d in $Ext\mathcal{D}ep$. Then the path followed by the convoy v must begin with the arrival sequence $arrSeq_a$ (Eq. A.27) and end with the departure sequence $depSeq_d$ (Eq. A.27).

$$arrSeq_a = (tg_0^a, \dots, tg_{n_1}^a) \tag{A.27}$$

$$depSeq_d = (tg_0^d, \dots, tg_{n_2}^d) \tag{A.28}$$

The arrival (resp. departure) sequence is composed by n_1+1 (resp. n_2+1) track-groups.

$$\forall r_i \in path(v), \begin{cases} i \leq n_1 \Rightarrow r_i = tg_i^a \\ n_v - n_2 - 1 \leq i \Rightarrow r_i = tg_j^d \textbf{ with } j = n_v - n_2 - 1 + i \end{cases} \tag{A.29}$$

Since the arrival and the departure must take place on a platform, we have:

$$r_{n_1+1} \in \mathcal{P} \quad \textbf{and} \quad r_{n_v-n_2-2} \in \mathcal{P} \tag{A.30}$$

A.2.3.5 Valid schedule

Let $v \in \mathcal{V}^+$ be a convoy of the solution. The schedule $sched(v)$ must respect the delay imposed by the graph $\mathcal{G}$ to go from one connector to another through a resource.

$$\forall t_i, t_{i+1} \in sched(v), \begin{cases} minTrTime(co_i, co_{i+1}) \leq t_{i+1} - t_i \\ t_{i+1} - t_i \leq maxTrTime(co_i, co_{i+1}) \end{cases} \tag{A.31}$$

The arrival time $arrTime_a$ and departure time $depTime_d$ must be respected. We assume the arrival sequence is composed by n_1+1 track-groups and the departure sequence by n_2+1 track-groups.

$$\forall t_i \in sched(v), \begin{cases} i = n_1 + 1 \Rightarrow t_i = arrTime_a \\ i = n_v - n_2 - 1 \Rightarrow t_i = depTime_d \end{cases} \tag{A.32}$$

A.2.3.6 Conflicts between convoys

A.2.3.7 On single resources

Let v_1, v_2 in $\mathcal{V}^+$ be two convoys of the solution, $path(v_1) = (co_0^1, r_0^1, \dots, co_{n_{v_2}}^1)$, $path(v_2) = (co_0^2, r_0^2, \dots, co_{n_{v_2}}^2)$ their respective paths, and $sched(v_1) = (t_0^1, \dots t_{n_{v_1}}^1)$, $sched(v_2) = (t_0^1, \dots t_{n_{v_2}}^1)$ the respectively associated schedules.

If a same single resource (platform, single track or maintenance facility) belongs to both path, the associated entry times must be different :

$$\forall r_i^1 \in sched(v_1), r_j^2 \in sched(v_2),\ r_i^1 = r_j^2 \Rightarrow t_i^1 \neq t_j^2 \tag{A.33}$$

Moreover, depending on the entry and exit connectors of both convoys, the following constraints on their entry and exit times must be enforced to preserve the order of the convoys on the resource (without loss of generality, we assume $t_1^i < t_j^2$) :

$$\forall r_i^1 \in sched(v_1), r_j^2 \in sched(v_2) \mid r_i^1 = r_j^2, t_1^i < t_j^2,$$
$$\begin{cases} (co_j^2 = co_{i+1}^1 \wedge co_j^2 \neq co_{j+1}^2) \Rightarrow t_j^2 > t_{i+1}^1 \\ (co_{j+1}^2 = co_{i+1}^1 \wedge co_j^2 \neq co_{j+1}^2) \Rightarrow t_{j+1}^2 > t_{i+1}^1 \\ (co_j^2 = co_{i+1}^1 \wedge co_j^2 = co_{j+1}^2) \Rightarrow (t_{j+1}^2 > t_{i+1}^1 \vee t_j^2 > t_{i+1}^1) \end{cases} \tag{A.34}$$

A.2.3.8 On track groups

On track groups, a specific conflict constraint applies, depending on the respective order of the entry and exit gates of both convoys. Though this constraint was implemented in the final solution, we do not reproduce it here for the sake of simplicity. One can refer to [Ramond and Nicolas, 2014].

A.2.3.9 At most one convoy assigned per extended arrival and departure

$$\forall a \in Ext\mathcal{A}rr, \sum_{v \in \mathcal{V}^+} from(v, a) \leq 1 \tag{A.35}$$

$$\forall d \in Ext\mathcal{D}ep, \sum_{v \in \mathcal{V}^+} assigned(v, d) \leq 1 \tag{A.36}$$

A.2.3.10 Example (Fig. A.3)

Let us consider the following instance: the sets of arrivals $\mathcal{A} = \{a_1 \dots a_5\}$ and departures $\mathcal{D} = \{d_1 \dots d_6\}$; The sets $\mathcal{J}_{arr} = \{\{a_2, a_3, a_4\}\}$ and $\mathcal{J}_{dep} = \{\{d_4, d_5, d_6\}\}$. Then

$$Ext\mathcal{A}rr = \{ea_1 = \{a_1\}, ea_2 = \{a_2, a_3, a_4\}, ea_3 = \{a_5\}\}$$
$$Ext\mathcal{D}ep = \{ed_1 = \{d_1\}, ed_2 = \{d_2\}, ed_3 = \{d_3\}, ed_4 = \{d_4, d_5, d_6\}\}$$

The convoys v_1 and v_2 where

- $v_1 = \{t_1\}$, with $from(v_1, ea_1) = 1$ and $assigned(v_1, ed_2) = 1$
- $v_2 = \{t_2, t_3\}$, with $from(v_2, ea2) = 1$ and $assigned(v_2, ed_4) = 1$

are represented on the Fig. A.3.

A.2.4 Objective function

The original problem featured two optimization criteria assembled in a weighted sum: maximization of the covered arrival and departures, and minimization of performance

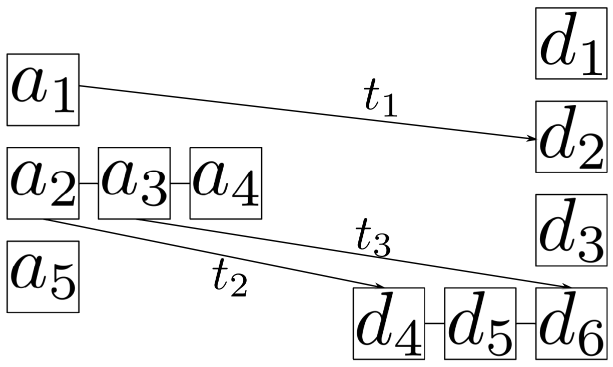


Figure A.3: Convoys with provenance and assignment to departures.

costs (platform usage costs, preferred reuse of trains, etc.). We focused on the maximization of covered arrival and departures (while of course limited by our choices to disregard maintenance, junction and disjunction operations). With the formalism defined in this section, maximizing arrivals and departures coverage comes down to maximizing the number of trains used in the solution:

$$\max \sum_{v \in \mathcal{V}^+} size(v) \tag{A.37}$$

A.3 Data Structures and Pre-processing

In the remainder of the paper, we will use the instance represented on Fig. A.1 as an example.

A maximum amount of convoy needs to be scheduled using the graph $\mathcal{G}$ previously defined. In order to reduce the cost of the routing phase by avoiding redundant treatment of very similar paths, we need a simplified structure with a higher granularity level.

The simplification consists of aggregating resources in groups (Fig. A.4) of resources sharing the same characteristics and neighborhood [Rogers et al., 1991]. We define the set $\mathcal{RG}$ of resources groups such that each resource $r \in \mathcal{R}$ is contained in one group $rg \in \mathcal{RG}$ with the following properties:

$$\forall rg \in \mathcal{RG}, \forall r1, r2 \in rg, neighSet_{r1} = neighSet_{r2} \wedge type(r1) = type(r2) \tag{A.38}$$

where *type* is defined as follows:

$$type : r \in \mathcal{R} \rightarrow \begin{cases} 0 \ \textit{if}\ r \in \mathcal{K} \\ 1 \ \textit{if}\ r \in \mathcal{P} \\ 2 \ \textit{if}\ r \in \mathcal{S} \\ 3 \ \textit{if}\ r \in \mathcal{Y} \\ 4 \ \textit{if}\ r \in \mathcal{F} \end{cases} \tag{A.39}$$

The goal of this process is to break the systematic and redundant treatment of similar resources while allowing for a more global reasoning.

Once the simplification is done, we can define a new graph $\mathcal{G}'((\mathcal{V}', \mathcal{E}'))$, more global, enclosing the graph $\mathcal{G}$ where vertices are groups connectors $\mathcal{GCO}$ and arcs are resources groups $\mathcal{RG}$. These two graphs represent the station with different levels of granularity: the level which needs to be used depends on the step of the algorithm.

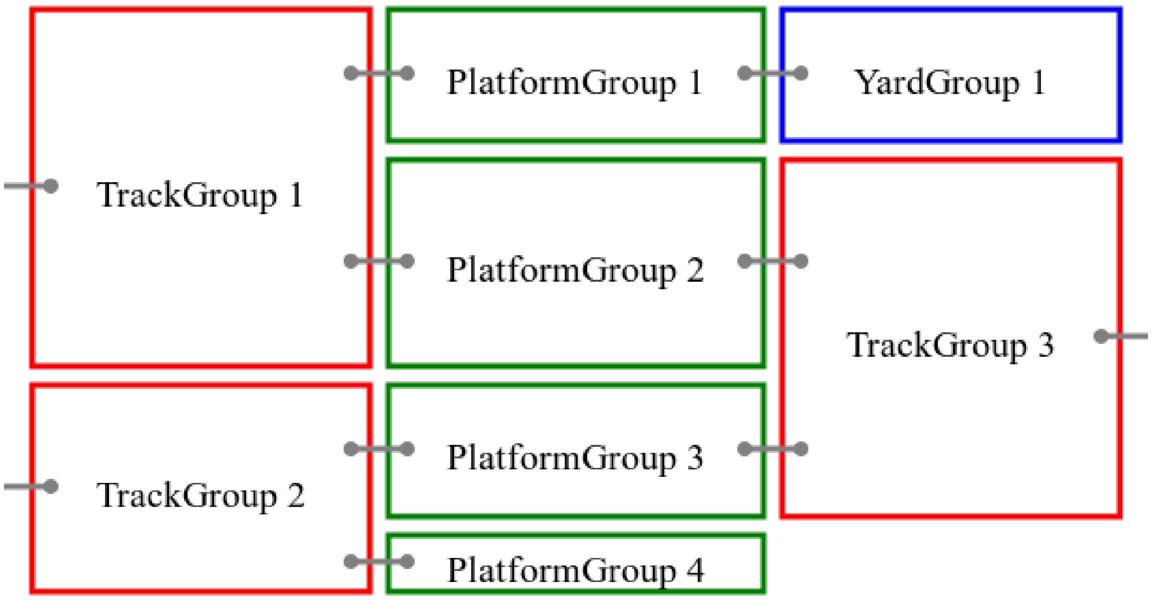


Figure A.4: Resource Aggregation

To simplify further the routing phase, we choose to consider the shortest path as the preferred path between two resources depending on the category. Thus, shortest paths between any two groups of resources, or rather group connectors, are computed during the pre-processing phase using *Floyd-Warshall* algorithm [Floyd, 1962]. During the process, a matrix of minimal and maximal travel time along this path is computed. Each resource $r \in R$ can be considered to have a minimum and a maximum usage time. For any track-group k, the minimum bound is equal to the maximum one: $trTime_k$. The other resources have a minimum bound equal to $minResTime$ or $\max(minResTime, revTime)$ (in case of similar input/output side) and the platforms have an upper bound $maxDwellTime$.

Let $gc1$, $gc2$ be two group connectors and c a category. First, we compute the shortest path to reach $gc2$ from $gc1$. Then we note $minTrTime_c(gc1, gc2)$ the lower time bound and $maxTrTime_c(gc1, gc2)$ the upper time bound to travel from the input connector $gc1$ to the output connector $gc2$ on the path found compatible with category c.

A.4 Routing Procedure

This section deals with the effective scheduling and routing of a train along a given group-level path. To schedule a train, the routing procedure considers the resource group-level path computed during the pre-processing phase, then tries to schedule the train at the resource-level, taking into account resource capacities, maximum lengths, minimal and maximal transition time, and conflicts with already scheduled trains. This is done by constraint propagation on multi-interval variables. Section A.4.1 introduce the notion of multi-interval variables in this context, and Section A.4.2 details the routing procedure.

A.4.1 Multi-interval variables

We use constraint propagation [Benhamou and Granvilliers, 2006] on hull and boxes [Benhamou et al., 1999] using multi-intervals variables to filter solutions of the *Path Scheduling problem.* First we introduce the notion of time interval $\mathcal{IH}$ (Eq. A.40) in the planning horizons $\mathcal{H}$, as defined by [Moore, 1966]:

$$\mathcal{IH} := \left\{ [\underline{h}, \overline{h}] \middle| \begin{array}{l} (\underline{h}, \overline{h}) \in \mathcal{H}^2 \\ \underline{h} \leq \overline{h} \end{array} \right\} \tag{A.40}$$

We note $[h]$ the interval $[\underline{h}, \overline{h}]$. Let $[h_1]$, $[h_2]$ be two intervals, we define the set operations $\cap$, $\cup$:

$$\begin{aligned} [h_1] \cap [h_2] &= [max(\underline{h_1}, \underline{h_2}), min(\overline{h_1}, \overline{h_2})] \\ [h_1] \cup [h_2] &= [min(\underline{h_1}, \underline{h_2}), max(\overline{h_1}, \overline{h_2})] \end{aligned} \tag{A.41}$$

A large number of interval arithmetic libraries exist (see [Knüppel, 1994], [Lerch et al., 2006]).

A generalization of $\mathcal{IH}$ is the power set $\mathcal{P}(\mathcal{IH})$. We call $\mathcal{MIH} = \mathcal{P}(\mathcal{IH})$ the set of all the time multi-intervals in the planning horizon. Let mi_1 and mi_2 be two elements of $\mathcal{MIH}$, the multi-interval intersection is defined as below:

$$mi_1 \cap mi_2 = \bigcup_{\substack{i_1 \in mi_1 \\ i_2 \in mi_2}} i_1 \cap i_2 \tag{A.42}$$

We introduce the operators *lo* and *up* on intervals and multi-intervals, which represent the lower and upper bounds of these quantities (see Fig A.5 for an example).

$$\forall x \in \mathcal{IH} \begin{cases} lo(x) = \underline{x} \\ up(x) = \overline{x} \end{cases} \quad \textbf{and then} \quad \forall y \in \mathcal{MIH} \begin{cases} lo(y) = \min\limits_{x \in y} lo(x) \\ up(y) = \max\limits_{x \in y} up(x) \end{cases} \tag{A.43}$$

Mathematical comparison operators can be defined to deal with intervals and multi-intervals. Let x be an interval or a multi-interval quantity. Then the comparison of x with a real value y implies conditions over the bounds of x.

$$x \leq y \quad \Leftrightarrow \quad up(x) \leq y \tag{A.44}$$

$$y \leq x \quad \Leftrightarrow \quad y \leq lo(x) \tag{A.45}$$

The inequality constraints are used to reduce the domain of the values through a set of methods called contractors [Chabert and Jaulin, 2009a]. Atomic and meta-contractors can be considered as basic elements of a new paradigm called contractor programming in order to perform efficient global optimization algorithm [Ninin, 2015] exploiting the properties of groups of constraints. In the next section of the paper we present a contraction method for the *Path Scheduling Problem* previously introduced.

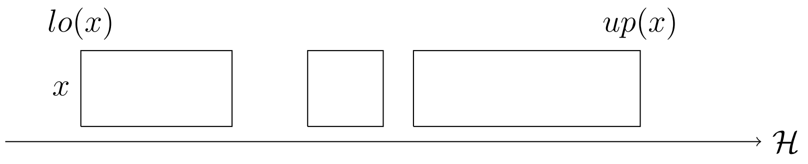


Figure A.5: Example of a multi-interval, with its lower and upper bounds.

The idea behind the routing procedure, described in the next section, is to consider scheduling variables as two types of multi-interval variables, *transition* variables t_i and *usage* variables u_i. That way, for a given convoy's schedule, the multi-interval variable u_i represents the time intervals during which the corresponding resource of the convoy's path r_i is available for the convoy (with respect, amongst other, to length and capacity constraints). Similarly, the multi-interval variable t_i represents the time intervals during which the convoy may exit the previous resource and enter resource r_i through connector co_i (with respect to other convoy's transitions).

The path and schedule definition presented in the simplified model are extended accordingly, by adding to the schedule usage time variables u_i in bijection with the resources r_i in the path:

$$path(v) = (co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_n) \tag{A.46}$$

$$sched(v) = (t_0, u_0, t_1, u_1, t_2 \dots t_{n-2}, u_{n-2}, t_{n-1}, u_{n-1}, t_n) \tag{A.47}$$

The procedure consists in refining those multi-intervals by constraint propagation.

However, propagating constraints on multi-intervals variables is very expensive. Computing speed may be increased using lazy evaluation [Madsen and Jensen, 1999]. Laziness is especially used and developed in functional programming languages [Johnsson, 1984] such as *Haskell* [Hudak et al., 2007]. It consists into a set of methods in order to limit the number of computations by non-evaluating quantities which are not required. This approach is not yet fully implemented in our solution, so there is still room for progress and potential speed-up.

A.4.1.1 Definition : $\mathcal{H}_{r,l,s}$

Let r be in $\mathcal{R}$, and l, s in $\mathbb{N}$. We define the set $\mathcal{H}_{r,l,s}$ as all the instants t in the planning horizons $\mathcal{H}$ such that the resource can be used by a convoy of size s with a total length l.

$$\mathcal{H}_{r,l,s} = \left\{ t \in \mathcal{H} \;\middle|\; \begin{array}{c} \sum_{v \in \mathcal{V}^+} (use(v,t,r) \times size(v)) \leq capa(r) - s \\ \sum_{v \in \mathcal{V}^+} (use(v,t,r) \times length(v)) \leq length(r) - l \end{array} \right\} \tag{A.48}$$

A.4.2 Algorithm

In this section, we detail the procedure used to route and schedule a convoy along a group-level path (Algorithm 3).

Let us consider a m-sized *convoy*, of total length l, and a group-level path *GlobalPath*. *GlobalPath* is formalized as an alternating list of connector-groups and resource-groups (gco_0, gr_0, ..., gco_1, gr_{n-1}, gco_n).

The procedure returns, if possible, a resource-level *path* (formalized as an alternating list of connectors and resources (co_0, r_0, ..., co_1, r_{n-1}, co_n) and a matching *schedule*. A *Schedule* is an alternate list of transition multi-interval variables, and usage multi-interval variables $(t_0, u_0, \dots, t_{n-1}, u_{n-1}, t_n)$. t_i represents the available intervals for transition between resources r_{i-1} and r_i on connector co_i, and u_i the available intervals for usage of resource r_i.

Schedule is initialized using *InitialSchedule*. In practice, we initialize all multi-interval variables to the full planning horizon, except for the transitions corresponding to the arrival and departure of the convoy, which are reduced to the arrival (respectively departure) time (see Fig. A.6a).

Algorithm (Alg. 3) consists of an iteration over every path $path := (co_0,\ r_1,\ \dots,\ r_{n-1},\ co_n)$ enclosed in the *GlobalPath*. Each iteration is divided in 2 steps (*FilterResourceUsagePath*, and *FilterConnectorPath*), described below.

Algorithm 3 : *Routing*

```
Require: $convoy$
Require: $globalPath := (gco_0,\ gr_0,\ \dots,\ gco_1,\ gr_{n-1},\ gco_n)$
Require: $initSchedule := (t_0, u_0, \dots, t_{n-1}, u_{n-1}, t_n)$
 1: for all $path = (co_0,\ r_0,\ \dots,\ co_{n-1}\ ,r_{n-1},\ co_n) \in globalPath$ do
 2:    $schedule \leftarrow FilterResourceUsagePath(convoy, path, initSchedule)$
 3:    if $(schedule) \neq \emptyset$ then
 4:       $schedule \leftarrow FilterConnectorPath(convoy, path, schedule)$
 5:       if $(schedule) \neq \emptyset$ then
 6:          return $(path, schedule)$
 7:       end if
 8:    end if
 9: end for
10: return $(\emptyset, \emptyset)$
```

1. Refinement of the multi-interval scheduling variables (t_0, u_0, ..., t_{n-1}, u_{n-1}, t_n) associated with the path p by filtering on each resource r_k, taking into account capacity, length and usage time constraints (Method *FilterResourceUsagePath*) using the following contractors. An example of this step is unfolded (Fig. A.6) on the simplified instance represented on Fig. A.1, using the parameters given in Tables A.1 and A.2, with the parameters $minResTime = 00:02:00$ and $revTime = 00:05:00$.

$$u_k \subseteq u_{max_k} \tag{A.49}$$

where u_{max_k} is the minimal multi-interval containing $\mathcal{H}_{r_k,l,m}$. This constraint (Eq. A.49) represents the limitation in term of length and/or capacity of the resource (Fig. A.6a, A.6b).

$$lo(t_k) \leq u_k \leq up(t_{k+1}) \tag{A.50}$$

This constraint (Eq. A.50) translates the fact that the convoy cannot occupy the resource before entering it and after exiting it (Fig. A.6b, A.6c).

$$\begin{aligned} minTrTime_{r_k}(co_k, co_{k+1}) &\leq lo(t_{k+1}) - lo(t_k) \\ up(t_{k+1}) - up(t_k) &\leq maxTrTime_{r_k}(co_k, co_{k+1}) \end{aligned} \tag{A.51}$$

These two constraints (Eq. A.51) express the relation between the enter time t_k and the exit time t_{k+1} depending on the minimal and maximal usage time of the resource r_k (Fig. A.6c, A.6d).

$$\forall u \in u_k, \{u\} \cap t_k \neq \emptyset \wedge \{u\} \cap t_{k+1} \neq \emptyset \tag{A.52}$$

This constraint (Eq. A.52) means that the resource r_k is able to welcome the convoy from its entrance to its exit (Fig. A.6d, A.6e).

$$\begin{aligned} t_k &\subseteq u_k \\ t_{k+1} &\subseteq u_k \end{aligned} \tag{A.53}$$

This constraint (Eq. A.53) means that the convoy can enter and exit the resource r_k only when r_k is able to receive it (Fig. A.6e, A.6f).

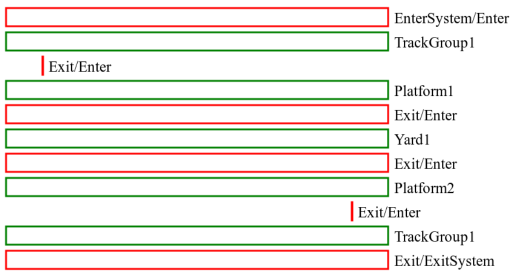
2. Similarly, procedure *FilterConnectorPath* contracts the multi-interval variables (t_0, $u_1, \ldots, u_{n-1}, t_n$) associated with *path* by computing for each resource r_k the conflicts of enter time t_{k-1} and exit times t_{k+1} with already scheduled convoys occupying r_k and propagating the associated constraints. This step also propagates again, when necessary, some of the previous constraints (Eq. A.51, A.52).

After both filtering procedures, if *schedule* is non-empty (which means it is possible to schedule the train along *path*), the algorithm returns, if not, it moves on to the next path.

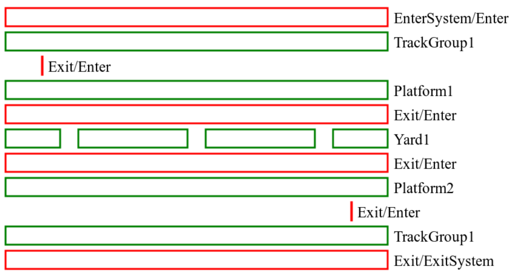
Id	arrTrain	Time	Sequence	Platform	idealDwell	maxDwell
Arr1	Train1	08:00:00	TrackGroup1	Platform1	00:03:00	00:12:00
Dep1		08:17:00	TrackGroup1	Platform2	00:04:00	00:10:00

Table A.1: Example instance: arrivals/departures listing

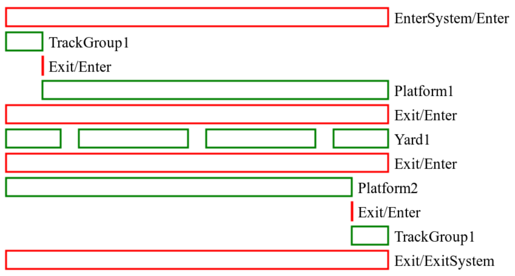
Finally, the train is scheduled according to an “earliest possible time” strategy within the allowed time intervals : each transition multi-interval is contracted to its lower bound, and usage multi-intervals are contracted accordingly. Note that the malleability of the structure leaves the possibility of implementing other strategies to avoid conflict with subsequent train schedules or optimize the enter and exit time on each resource to reduce penalty.



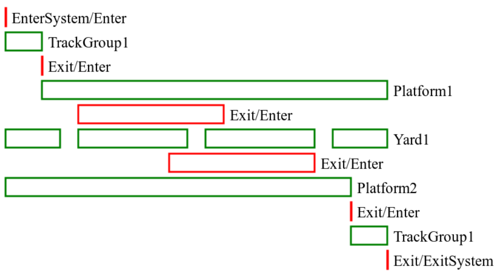
(a) Initialization



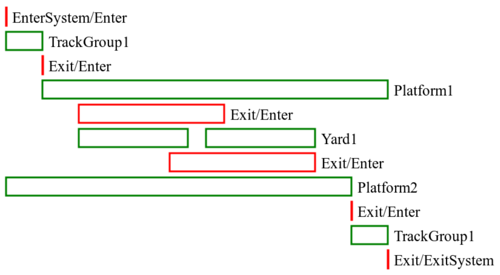
(b) Propagation Eq. A.49



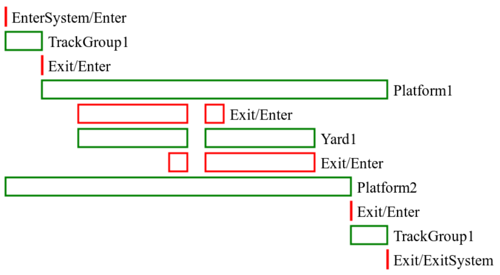
(c) Propagation Eq. A.50



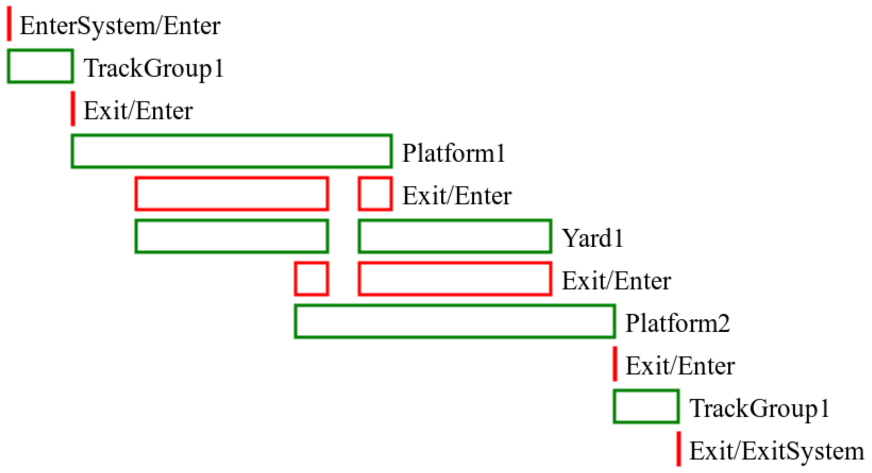
(d) Propagation Eq. A.51



(e) Propagation Eq. A.52



(f) Propagation Eq. A.53



(g) Constraint Propagation Result

Figure A.6: Multi-Interval Filtering on the example instance, from *Arr*1 to *Dep*1 (see Table A.1) with the consumptions indicated in Table A.2.

Resource	Begin	End
Yard1	08:01:00	08:02:00
Yard1	08:08:00	08:09:00
Yard1	08:15:00	08:16:00

Table A.2: Example instance: imposed consumptions listing

A.4.2.1 Example of propagation (Fig A.6)

Fig. A.6 illustrates an example of propagation. Each line represents a multi-interval variable; lines labeled "Exit/Enter" are transition variables, and lines labeled with a resource name are usage variables.

Initially (Fig. A.6a), none of the variables are restricted, except for the transitions corresponding to the arrival and departure (first and last "Exit/Enter"), which are restricted to the exact arrival and departure times. Then, in Fig. A.6b the usage of resource $Yard1$ is restricted according to the imposed consumptions listed in Table A.2. In Fig. A.6c, usage of resources $Platform1$ and $TrackGroup1$ are restricted according to the preceding "Exit/enter" transition variables: the convoy cannot use a resource before it enters it. Then, the transition variables are restricted to allow sufficient time between the entry and the exit on each resource (Fig. A.6d).

Let us now focus on resource $Yard1$: the last interval of its usage variable doesn't intersect with the following transition variable, so the yard cannot be used during this interval. It is thus removed in Fig. A.6e. The enter and exit transitions are adjusted accordingly in Fig. A.6f, as the convoy may not enter or exist the yard at a given instant if it is not available for use.

Fig. A.6g shows the result once a similar propagation has been performed $Platform1$ and $Platform2$.

A.5 Greedy Assignment Algorithm

The problem is complex due to the routing problem in the station. Resource usage is crucial and we should avoid to engorge one resource of the station. This idea leads to the strategy of the greedy algorithm we present in this section.

We call *immediate arrivals-departure* arrivals immediately followed by a departure on the same platform. In this case, the convoy only has to be routed through the arrival sequence and departure sequence, so the path is set except for the platform, and the convoy occupies few resources.

If a departure is not an immediate departure, the convoy has to be routed to a parking spot and parked between its arrival and its departure. We chose to park on yards only.

Our objective is the maximization of covered arrivals and departures as stated in Section 2. However, to comply with the multi-criteria objective of the original problem and limit penalties, considerations on trains reuses, preferred platform and platform usage penalties are implemented in the solution.

The algorithm is composed of three main steps.

1. First, we try to schedule immediate arrivals-departures while considering train reuses as hard constraints.

2. Then, we relax these constraints and try to schedule immediate arrivals-departures.

3. Finally, we try to schedule remaining trains from the arrival to departure with a parking phase (on yards only) in-between, first considering train reuses as hard constraints, then relaxing these constraints.

To improve coverage of joint departures without performing junctions nor disjunctions, each of these three main assignment phase follows the three steps described below.

(i) Try and assign joint arrivals to joint departures. Both departures and arrivals are taken in decreasing order of convoy size. Between a given joint arrival and a given joint departure, simple dynamic programming allows to compute an assignment maximizing the number of arrivals and departures satisfied, while respecting compatibility and trains order. When at least some members of a joint arrival/departure have been assigned, all the other members are locked and cannot be used in the subsequent assignment steps.

(ii) Try and assign unitary arrivals to unitary departures.

(iii) Try and assign members from remaining joint arrivals to remaining unitary departures (joint or not) and symmetrically remaining unitary arrivals to members of remaining joint departures. When one member is assigned, the other members are locked.

Formally, these sub-steps are described by Alg. 4, using the following definitions:

- $\mathcal{J}^+_{arr}$ (respectively $\mathcal{J}^+_{dep}$) the set of joint arrivals (respectively departures) used in the solution,

$$\mathcal{J}^+_{arr} = \{a \in \mathcal{J}_{arr} | \exists j = jointArr_a, \exists a' \in jaList_j, a' \in \mathcal{A}^+\}$$
$$\mathcal{J}^+_{dep} = \{d \in \mathcal{J}_{dep} | \exists j = jointDep_d, \exists d' \in jdList_j, d' \in \mathcal{D}^+\}$$

- $\mathcal{A}^+$ (respectively $\mathcal{D}^+$) the set of unitary arrivals (arrivals which are not members of any joint arrival) used in the solution,

$$\mathcal{A}^+ = \{a \in \mathcal{A} | arrTrain_a \in \mathcal{T}^+\} \cup \mathcal{J}^+_{arr}$$
$$\mathcal{D}^+ = \{d \in \mathcal{D} | depTrain_d \in \mathcal{T}^+\} \cup \mathcal{J}^+_{dep}$$

- by complement, the sets of joint / unitary arrivals (respectively departures) not used in the solution: $\mathcal{A}^- = \mathcal{A} - \mathcal{A}^+$, $\mathcal{D}^- = \mathcal{D} - \mathcal{D}^+$, $\mathcal{J}^-_{arr} = \mathcal{J}_{arr} - \mathcal{J}^+_{arr}$, $\mathcal{J}^-_{dep} = \mathcal{J}_{dep} - \mathcal{J}^+_{dep}$.

Algorithm 4 Assignment Greedy Algorithm Step

```
1: $Assign(\mathcal{J}^-_{arr}, \mathcal{J}^-_{dep})$
2: $Assign(\mathcal{A}^- - \mathcal{J}^-_{arr}, \mathcal{D}^- - \mathcal{J}^-_{dep})$
3: $Assign(\mathcal{A}^-, \mathcal{D}^-)$
```

Non-assigned departures and compatible arrivals are taken in increasing order. Each time, the *Routing* procedure is used to schedule the corresponding convoy. This continues until an arrival with a feasible routing has been found (then the arrival is assigned to the departure), or until there is no more fit arrivals (then the departure remains unsatisfied).

To limit the number of infeasible attempts, arrivals and departures are filtered, taking into account compatibility characteristics such as category and remaining time/distance before maintenance, but also existence of a path between two arrival/departure sequences and time between arrival and departure.

$$arrTime_a + minTrTime_{cat_{T_a}}(P_a, P_d) \leq depTime_d \tag{A.54}$$

The routing algorithm (Alg. 3) is used inside the greedy subroutines *Assign* (Alg. 4). The latter consists in a serie of attempts, iterating on the arrivals and the departures sets. For each couple arrival/departure, a compatible convoy is selected and a resource group-level path (computed during the pre-processing phase) is selected to link the arrival sequence with the departure sequence. Thus, the routing algorithm is applied this convoy and global path. If the process fails to provide a valid resource-level path (Alg. 3, line 10), we move on to the next compatible couple. Otherwise, the convoy is scheduled along the path, thus the arrival and the departure are respectively removed from the available arrivals and the available departures sets.

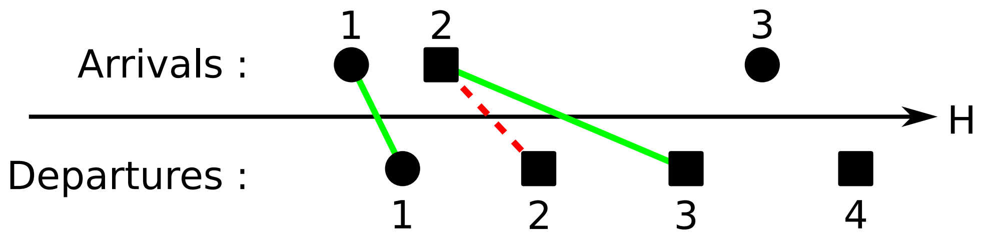


Figure A.7: Greedy Algorithm : example

Fig. A.7 shows a simplified example on an instance with three arrivals and four departures. A departure is compatible with an arrival if they are represented using the same shape (disc, square) and if the arrival takes place before the departure on the time line. On the figure, a line from an arrival to a departure is an attempt to route and schedule a convoy through the station from the arrival to the departure. A red dotted line is a failure while a green plain line is a success. Here, the couple $arr1/dep1$ is compatible and is successfully routed and scheduled. The couple $arr2/dep2$ is compatible but the routing procedure fails, so the next compatible couple $arr2/dep3$ is considered. The routing procedure successfully returns a valid schedule, so $arr2$ and $dep2$ are assigned to each other, and the convoy is routed and scheduled. The last arrival, $arr3$, does not have any compatible departure, so no routing attempt is made.

In order to limit platform assignment penalties, when scheduling a convoy from an arrival to a departure, the preferred platform is tried in priority. To limit the platform usage penalties, we limit the time that a train can spend on a platform. For immediate departures, this amount of time is computed using the ideal dwell times of the departure and the arrival, and the costs defined in parameters, in order to ensure that the platform usage cost for this train does not exceed the $Uncov$ penalty for not covering the arrival. We filter the arrivals verifying this condition. For arrivals and departures separated by a parking phase, we ensure during the routing phase that the ideal dwell times are exactly respected.

A.6 Results and Conclusions

The experiments were performed on an Optiplex computer featuring an Intel Core 2 Duo E8400 3.00GHz CPU with 4Go RAM running Xubuntu 64 bits. Table A.3 presents the results obtained on set instance B, provided by the challenge organizers. The execution was stopped after 10 minutes of running time. The table reports the results obtained when stopping after the first two steps of the algorithm (which means allowing only immediate departures), and the results obtained after full execution of the algorithm (thus allowing a parking phase). In each case, we give the number and proportion of covered departures. We also give, for information purposes, the value of the objective function used in the original formulation of the ROADEF Challenge problem [Ramond and Nicolas, 2014], which aggregates various performance costs as well as penalties for uncovered departures and arrivals. However, within our simplified model, the number of covered departures alone is a more pertinent indicator.

The solutions provided within the allowed time cover 20% to 40% of departures (and as much arrivals). Of course, this is very few and not suited for a practical application. Although, considering the complexity of the problem and the size of the search space, it is actually quite reasonable for a simple greedy assignment algorithm. Moreover, given that our approach does not allow maintenance, junction and disjunction operations, the number of departures which can be covered is limited. This suggests that the routing algorithm itself is quite powerful, and that paired with a more advanced assignment optimization algorithm, it could lead to interesting results.

From Table A.3, one can notice that most covered departures are "immediate departures", which are departures following directly the arrival on the same platform. The second phase, allowing trains to change platforms between arrival and departure by transiting through yards, gains only a few percent and is time consuming. Thus, considering only immediate departures seems a suitable approach to compute rather quickly an initial solution.

When considering immediate departures, the path and timing is already precisely defined by the arrival and departure sequences. In this case, the *Routing* procedure only serve to select determine the feasibility of this path and timing for a given couple ar-

	immediate departures only			with parking phase		
	t (s)	objective	covered dep	t (s)	objective	covered dep
B1	24	1812600	35% (443/1235)	t/o	1749000	38% (474/1235)
B2	23	1682080	39% (490/1235)	t/o	1656180	40% (503/1235)
B3	21	1711330	38% (472/1235)	t/o	1665330	40% (495/1235)
B4	71	2578160	36% (649/1780)	t/o	2561360	36% (657/1780)
B5	136	3190890	34% (747/2153)	t/o	3155790	35% (764/2153)
B6	72	2584130	36% (645/1780)	t/o	2559330	36% (657/1780)
B7	2	456112	34% (105/304)	68	420112	40% (123/304)
B8	2	473484	31% (96/304)	83	439884	37% (113/304)
B9	72	2978598	29% (590/1967)	t/o	2858098	32% (649/1967)
B10	< 1	364594	18% (36/196)	226	321794	29% (57/196)
B11	3	1984144	16% (190/1122)	t/o	1858544	22% (250/1122)
B12	1	1031864	16% (94/570)	t/o	982164	20% (118/570)

Table A.3: Summary of the results on set instance B. The first 3 columns present the results for the algorithm limited to immediate departures only, the last 3 when allowing an intermediate parking phase in yards between arrival and departure. for each case, we give the execution time in seconds, the value of the ROADEF Challenge objective function and the proportion of covered departures. “t/o” means that the execution has reached the timeout, set to 10 minutes.

rival/departure and select a platform, given the current state of the solution and avoiding conflict with other trains. When allowing parking, additional decisions must be made, namely the selection of the parking yard and the duration of stay. Thus, in this case the exploration performed by the *Routing* procedure is heavier and more time consuming.

Total execution times appear shorter for instances with few departures and arrivals (B7,B8,B10), as less routing attempts need to be made. The comparatively high execution time for instance B10 might result of the high proportion of joint departures and arrivals in this instance, as members of a joint arrival or departure may be tentatively routed several times, as convoys and as unitary trains.

The most interesting point in the solution exposed in this paper is without any doubt the association of the three-level routing algorithm with the greedy algorithm. While the structure of the instance can be used by the greedy method, results are quickly returned. When the situation becomes more intricated the routing algorithm is used to create relevant routes. Indeed, the multi-interval constraint propagation is flexible, efficient and returns a set of possible range-value to enter and exit from each resource.

Note that as a canonical path between any pair of resources is computed during the preprocessing phase, trains travelling with a given origin and destination are always routed along the same path. Choosing among a set of preferred paths, or computing on the fly alternative paths forbidding certain resources could be useful to avoid engorgement.

However the solution we propose here is robust, and the optimizer always return a feasible solution. An important aspect of the greedy algorithm is the ability to build a correct solution really fast on all the instances and to build a better solution if more time is available.

A simple optimization process could merely consist of choosing a random train, then removing it from the solution and finally try to build another schedule. Rather than just choosing randomly, one could select the train which generates the most conflicts: each time the algorithm tries and fails to schedule a train, the conflict value of the trains which prevented the scheduling is incremented ; at the end, the train with the highest conflict value is removed from the solution. This could be used as the basis for a local search optimization procedure.

B Résumé en version Française

Ce mémoire présente un ensemble d'outils pour résoudre de manière garantie des problèmes d'optimisation de systèmes dynamiques multi-physiques. Ces systèmes sont largement représentés dans la recherche fondamentale (physique, chimie, biologie, environnement, etc) et trouvent des applications directes dans des domaines variés tels que la conception en ingénierie, la modélisation de réactions chimiques et de composants électroniques, la simulation de systèmes biologiques et économiques ou même dans la prédiction de la performance sportive.

La résolution de ces problèmes d'optimisation s'effectue en deux phases, appelées phase de modélisation (A) et phase de résolution (B) qui sont dépendantes l'une de l'autre. Les méthodes de résolution dépendent du modèle utilisé, ce dernier étant également influencé par les méthodes de résolution envisagées.

La phase de modélisation (A) consiste à mettre en équation le problème sous forme d'un modèle mathématique constitué d'un ensemble de variables (1), d'un ensemble de contraintes (2) ainsi que d'une ou plusieurs fonctions de coût (3) que l'on veut minimiser.

(1) L'ensemble des variables du modèle que nous considérons se divise en trois catégories.

   (a) Les variables de décision $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^p$ sont des variables entières ou réelles qui permettent de décrire le système et d'agir sur son comportement. Ces variables peuvent prendre différentes valeurs et l'espace dans lequel elles évoluent est appelé espace de recherche.

   (b) Les variables de contrôle $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^s$ sont des variables réelles de sortie sur lesquelles il n'est pas possible ou souhaitable d'agir directement. Cependant, elles permettent de contrôler des grandeurs qui dépendent des variables de décision (a), des variables fonctionnelles (c) et des contraintes (2).

   (c) Les variables fonctionnelles $\mathbf{u} \in \mathcal{U} \subseteq (\mathbb{R} \to \mathbb{R})^n$ sont des variables qui décrivent des fonctions. Elles permettent de représenter et de contrôler la dynamique du système induit par la valeur des variables de décision (a), des variables de contrôle (b) et des contraintes (2).

(2) Dans le modèle que nous considérons, nous distinguons différents types de contraintes.

(d) Les contraintes algébriques permettent de manipuler des grandeurs statiques (quantité, taille, poids, volume, densité, etc). Elles sont la plupart du temps non linéaires, non convexes et parfois discontinues. Ces contraintes permettent de lier les variables de décisions aux variables réelles de contrôle (b) et aux fonctions de contrôle (c).

$$\mathbf{g}(\mathbf{x}, \mathbf{y}, \mathbf{u}) \leq 0$$

(e) Les contraintes fonctionnelles permettent de manipuler des grandeurs dynamiques. Ces contraintes peuvent être relativement simples comme la monotonie, la périodicité, la croissance, mais aussi bien plus complexe par la prise en compte de contraintes différentielles simples ou définies par morceaux. Les équations différentielles sont généralement utilisées pour modéliser des comportements physico-chimiques (électriques, mécaniques, magnétiques, thermiques, cinétiques, etc), des propriétés d'usure et d'autres caractéristiques qui varient lors de l'évolution du système.

∗ Les contraintes différentielles sont décrites à l'aide d'équations différentielles ordinaires (ODEs). Elles permettent de modéliser simplement le comportement d'un système tout au long de son évolution.

$$\mathbf{u}'(t) = \mathbf{d}(\mathbf{x}, \mathbf{y}, \mathbf{u}(t))$$

∗ Les contraintes différentielles par morceaux permettent de modéliser des systèmes dynamiques bien plus complexes, dont le comportement lui même n'est pas constant mais varie au cours de l'évolution du système. Ces contraintes particulières sont utiles pour modéliser des changements d'environnement.

$$\mathbf{u}'(t) = \begin{cases} \mathbf{d_1}(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) & \text{if} \quad c_1(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) \\ & \vdots \\ \mathbf{d_q}(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) & \text{if} \quad c_q(\mathbf{x}, \mathbf{y}, \mathbf{u}(t)) \end{cases}$$

Par exemple, un objet massif se déplaçant en orbite terrestre haute n'est soumis qu'à l'unique force de gravitation. Une fois celui-ci rentré dans l'atmosphère des forces de frottements viennent s'ajouter. Quand l'objet tombe dans l'océan, les forces de frottement sont encore différentes en raison du changement de milieu entre l'atmosphère et l'hydrosphère. De plus la poussée d'Archimède ainsi que les courants marins modifient la trajectoire de l'objet.

L'utilisation conjointe des contraintes algébriques ainsi que des contraintes fonctionnelles rend possible la modélisation de systèmes dynamiques et permet de décrire l'ensemble des solutions réalisables $\mathcal{S}_{\mathcal{X}}$. Une solution réalisable est un ensemble de valeurs, qui, une fois assignées à l'ensemble des variables, respectent chacune des contraintes du modèle.

$$\mathcal{S}_{\mathcal{X}} = \left\{ (\mathbf{x}, \mathbf{y_x}, \mathbf{u_x}) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{U} \;\middle|\; \begin{array}{c} \mathbf{g}(\mathbf{x}, \mathbf{y_x}, \mathbf{u_x}) \leq 0 \\ \mathbf{d}(\mathbf{x}, \mathbf{y_x}, \mathbf{u_x}(t)) = \mathbf{u}'(t) \end{array} \right\}$$

(3) Les fonctions de coût $\mathbf{f_{cost}}$ sont utilisées pour mesurer la qualité d'une solution selon un ou plusieurs critères (volume, poids, impact environnemental, rapidité de la réaction, énergie mécanique produite. . .).

$$\textit{Minimize}\ \mathbf{f_{cost}}(\mathbf{x}, \mathbf{y}, \mathbf{u})$$

Ces critères sont importants mais ne requièrent pas un contrôle strict à l'aide des contraintes, ils sont considérés comme des préférences. L'optimisation de problèmes multi-objectifs est motivée par la gestion de coûts parfois conflictuels les uns avec les autres. Par exemple on peut chercher à:

– maximiser une population en minimisant son impact sur l'environnement,
– maximiser une performance sportive en minimisant la dépense énergétique qu'elle requiert,
– maximiser la qualité d'un produit en minimisant son coût de production,
– maximiser la température d'une pièce en minimisant la dépense calorifique.

Les fonctions de coût permettent de munir l'ensemble des solutions réalisables d'un ordre partiel et donc de comparer différentes solutions réalisables entre elles. Ainsi, une solution optimale est une solution qui est réalisable et minimale au sens de l'ordre partiel donné par la fonction de coût. Ainsi, il est possible de définir un ensemble de solutions dominantes $\mathcal{D}_{\mathcal{X}}$ qui regroupe toutes les solutions optimales.

$$\mathcal{D}_{\mathcal{X}} = \{(\mathbf{x}^*, \mathbf{y}^*, \mathbf{u}^*) = \mathbf{s}^* \in \mathcal{S}_{\mathcal{X}} \mid \nexists \tilde{\mathbf{s}}_{\mathbf{x}} \in \mathcal{S}_{\mathcal{X}},\ \mathbf{f_{cost}}(\tilde{\mathbf{s}}_{\mathbf{x}}) < \mathbf{f_{cost}}(\mathbf{s}^*)\}$$

L'optimisation de ce type de modèle est un problème difficile. Cette difficulté est induite par le nombre, la nature et l'hétérogénéité des variables (entières, réelles et fonctionnelles), les différents ordres de grandeur mis en jeu, mais également par la nature des contraintes (algébriques, fonctionnelles) qui ne sont pas nécessairement convexes, ni même continues. De plus les fonctions de coûts ne sont pas nécessairement régulières et peuvent comporter un grand nombre de minima locaux, ce qui rend la recherche de la solution optimale d'autant plus difficile.

La seconde phase (B) consiste à utiliser un ensemble de méthodes afin d'extraire du modèle une ou plusieurs solutions. Cependant la complexité de ces problèmes ainsi que la limite des outils existants sont deux freins à l'utilisation de méthodes d'optimisation globale garantie dans le but de résoudre de tels problèmes. Pour cette raison, on trouve dans la littérature scientifique plusieurs niveaux d'approximation concernant chacune de ces deux phases.

Lors de la modélisation d'un problème, il est courant d'assister à une simplification de sa partie dynamique, cette partie pouvant même être ignorée dans certains cas. Une partie du modèle statique ou son intégralité peut aussi être rendue convexe ou linéarisée.

Lors de la résolution, des méthodes approchées peuvent être utilisées. La recherche d'une solution optimale par des algorithmes évolutionnaires ne permet pas de garantir l'optimalité de la solution obtenue. De la même manière, l'utilisation de méthodes numériques ne permet pas de garantir la résolution des contraintes différentielles et la réalisabilité ou non de la solution.

Ces approximations donnent en pratique des résultats suffisants pour l'industrie, cependant elles ne permettent pas de garantir l'optimalité ni même la réalisabilité de la solution obtenue. De plus, la modélisation et la résolution des parties dynamiques sont importantes et ces approximations ne permettent pas de garantir le comportement du système au cours de son évolution.

C'est la raison pour laquelle on s'intéresse, dans cette thèse, à la création d'un outil global permettant de résoudre de manière garantie les problèmes pouvant être modélisés de cette manière. Après une courte introduction, nous introduisons plus en détail le modèle des problèmes que nous souhaitons résoudre dans le premier chapitre (I) avec quelques exemples d'applications industrielles sur des problèmes de conception.

(II) Dans le deuxième chapitre, nous abordons le cas des arithmétiques garanties avec l'arithmétique d'intervalles et une extension appelée arithmétique de tubes.

  - L'arithmétique d'intervalles (IA) est une arithmétique particulière permettant de garantir les résultats numériques des opérations et de représenter des ensembles continus de valeurs. Dans un premier temps, cette garantie s'obtient en représentant les valeurs réelles par les plus petits intervalles numériquement représentables les contenant. Dans un second temps, la garantie se conserve en redéfinissant les opérateurs arithmétiques et les fonctions usuelles avec un contrôle strict des arrondis de l'unité de calcul. Dans ce chapitre, nous présentons les aspects positifs de l'arithmétique d'intervalles, tels que la garantie des résultats et la possibilité de raisonner sur des ensembles de cardinal infini. De manière similaire, les limites portant sur la précision des intervalles résultant d'opérations sont abordées et quelques astuces permettant d'augmenter leur précision sont développées.
  - L'arithmétique de tubes (TA) est une extension de l'arithmétique d'intervalles qui permet de représenter des ensembles de fonctions unaires et dans le même temps d'étendre la garantie des opérations sur ces fonctions. Dans ce chapitre les opérations sont redéfinies et plusieurs représentations sont discutées, tout comme leurs atouts et leurs limites.

(III) Le troisième chapitre est dédié à l'intégration garantie d'équations différentielles ordinaires (ODEs) afin de prendre en compte les contraintes différentielles de manière efficace. Dans ce chapitre nous étudions un algorithme itératif qui repose sur deux étapes permettant de calculer d'abord une inclusion globale de la solution puis une inclusion locale par un processus de contraction. Différentes méthodes permettant de gagner en précision et en efficacité y sont exposées. En particulier, pour le calcul des inclusions globales deux méthodes FOE puis HOE basées sur l'opérateur de Picard-Lindelöf, puis l'utilisation de polynômes de Taylor, de formes centrées, de changements de repères par la QR-Factorization et de méthodes de coupes par consistances pour la contraction.

(IV) Dans le quatrième chapitre nous posons les bases de la Programmation Par Contraintes (CP) permettant de résoudre des Problèmes de Satisfaction de Contraintes (CSP) avec des méthodes de filtrages et de contractions. Ces problèmes consistent à rechercher une ou plusieurs solutions réalisables. Ces algorithmes sont utilisés pour converger plus rapidement vers l'ensemble des solutions réalisables ce qui permet de restreindre l'espace de recherche. Dans ce chapitre nous étudions divers contracteurs atomiques comme HC4, Cmonotonic et Cdiff permettant d'exploiter les contraintes algébriques et fonctionnelles ainsi que quelques méta-contracteurs permettant d'utiliser des schémas particuliers de contractions sur une partie ou l'ensemble des contraintes.

(V) Dans le cinquième chapitre, nous exposons les problématiques liées à l'intégration garantie d'équations différentielles ordinaires par morceaux (p-ODEs) ainsi qu'une méthode pour y parvenir. Pour cela nous réutilisons les méthodes étudiées pour l'intégration garantie d'ODEs. Cependant les limites posées par certaines zones nécessitent un traitement spécifique. Nous développons donc un théorème avec sa preuve permettant de construire un algorithme pour l'intégration garantie de ces zones particulières.

(VI) Dans le sixième chapitre nous intégrons l'ensemble des méthodes précédemment décrites au sein d'un algorithme d'optimisation globale déterministe. Celui-ci repose sur l'utilisation conjointe de méthodes par intervalles et de l'algorithme Branch and Bound bien connue dans le monde de la recherche opérationnelle. Aussi, nous décrivons les limites paradoxales de l'optimisation garantie que sont la perte de garantie pour des systèmes contenant des contraintes liées.

(VII) Dans le septième chapitre, nous testons les méthodes décrites sur différents systèmes. Dans un premier temps nous appliquons l'algorithme d'optimisation garantie sur le problème industriel, détaillé dans le deuxième chapitre, qui ne contient pas de contrainte fonctionnelle. Puis nous testons la méthode d'intégration garantie d'équations différentielles ordinaires par morceaux sur un exemple académique développé pour l'occasion. Finalement nous étudions le résultat de l'algorithme d'optimisation globale sur un problème contenant des contraintes différentielles par morceaux.

Au cours de ce travail de recherche, j'ai développé un outil GDODynS en langage C++, inspiré d'IBEX, mettant en oeuvre l'ensemble des méthodes présentées pour la prise en compte des incertitudes, pour garantir l'optimalité et la réalisabilité des solutions à ces problèmes. Pour cela j'ai utilisé un ensemble de bibliothèques permettant la gestion de l'arithmétique d'intervalles avec Profil-BIAS et la résolution garantie d'équations différentielles ordinaires avec Vnode-LP. J'ai également étendu les connaissances théoriques pour permettre la résolution garantie d'équations différentielles ordinaires par morceaux.

Cependant, la taille des instances ainsi que la complexité que représente l'intégration garantie d'équations différentielles constituent une barrière pour une optimisation rapide. La mise en place d'heuristiques concernant la propagation des contraintes fonctionnelles semble être une approche intéressante de même que l'utilisation de nouvelles arithmétiques garanties telles que l'arithmétique affine.

Durant cette thèse, j'ai également eu l'opportunité de travailler en collaboration avec Florence Thiard sur un projet de recherche opérationnel proposé par la SNCF dans le cadre du challenge international ROADEF/EURO 2014 que nous avons remporté dans la catégorie Junior et où nous nous sommes classés $4^{ème}$ dans la catégorie Senior. Le résultat de cette collaboration a donné lieu à une publication dans Annals of Operations Research, laquelle est disponible en annexe de ce document.

Abstract

In this thesis a set of tools based on guaranteed methods are presented in order to solve multi-physics dynamic problems. These systems can be applied in various domains such that engineering design process, model of chemical reactions, simulation of biological systems or even to predict athletic performances.

The resolution of these optimization problems is made of two stages. The first one consists in defining a mathematical model by setting up the equations for the problem. The model is made of a set of variables, a set of algebraic and functional constraints and cost functions. The latter are used in the second stage in order to extract the optimal solutions from the model depending on several criteria (volume, weight, etc).

Algebraic constraints are used to describe the static properties of the system (quantity, size, density, etc). They are non-linear, non-convex and sometimes discontinuous. Functional constraints are used to manipulate dynamic quantities. These constraints can be quite simple such as monotony or periodicity or they can be more complex such as simple or piecewise differential constraints. Differential equations are used to describe physico-chemical properties (magnetic, thermal, etc) and other features evolving with the component use.

Several levels of approximation exist for each of these two stages. These approximations give some relevant results but they do not guarantee the feasibility nor the optimality of the solutions.

After presenting a set of guaranteed methods in order to perform the guaranteed integration of ordinary differential equations, a peculiar type of hybrid system that can be modeled with piecewise ordinary differential equation is considered. A new method that computes guaranteed integration of these piecewise ordinary differential equations is developed through an extension of the initial algorithm based on several proofs and theorems. In a second step these algorithms are gathered within a contractor programming module that have been implemented. It is used to solve algebraic and functional constraint satisfaction problems with guaranteed methods. Finally, the considered optimization problems are solved with a modular deterministic global optimization algorithm that uses the previous modules.

Keywords: Global Optimization, Guaranteed Arithmetic, Constraint Programming, Ordinary Differential Equation, Guaranteed Integration, Hybrid System.

Résumé

Ce mémoire présente une approche basée sur des méthodes garanties pour résoudre des problèmes d'optimisation de systèmes dynamiques multi-physiques. Ces systèmes trouvent des applications directes dans des domaines variés tels que la conception en ingénierie, la modélisation de réactions chimiques, la simulation de systèmes biologiques ou la prédiction de la performance sportive.

La résolution de ces problèmes d'optimisation s'effectue en deux phases. La première consiste à mettre le problème en équations sous forme d'un modèle mathématique constitué d'un ensemble de variables, d'un ensemble de contraintes algébriques et fonctionnelles ainsi que de fonctions de coût. Celles-ci sont utilisées lors de la seconde phase qui consiste à extraire du modèle les solutions optimales selon plusieurs critères (volume, poids, etc).

Les contraintes algébriques permettent de manipuler des grandeurs statiques (quantité, taille, densité, etc). Elles sont non linéaires, non convexes et parfois discontinues. Les contraintes fonctionnelles permettent de manipuler des grandeurs dynamiques. Ces contraintes peuvent être relativement simples comme la monotonie ou la périodicité, mais aussi bien plus complexe par la prise en compte de contraintes différentielles simples ou définies par morceaux. Les équations différentielles sont utilisées pour modéliser des comportements physico-chimiques (magnétiques, thermiques, etc) et d'autres caractéristiques qui varient lors de l'évolution du système.

Il existe plusieurs niveaux d'approximation pour chacune de ces deux phases. Ces approximations donnent des résultats pertinents, mais elles ne permettent pas de garantir l'optimalité ni la réalisabilité des solutions.

Après avoir présenté un ensemble de méthodes garanties permettant de résoudre de manière garantie des équations différentielles ordinaires, nous formalisons un modèle particulier de systèmes hybrides sous la forme d'équations différentielles ordinaires par morceaux. A l'aide de plusieurs preuves et théorèmes nous étendons la première méthode de résolution pour résoudre de manière garantie ces équations différentielles par morceaux. Dans un second temps, nous intégrons ces deux méthodes au sein d'un module de programmation par contracteurs, que nous avons implémenté. Ce module basé sur des méthodes garantie permet de résoudre des problèmes de satisfaction de contraintes algébriques et fonctionnelles. Ce module est finalement utilisé dans un algorithme d'optimisation globale déterministe modulaire permettant de résoudre les problèmes considérés.

Mots clefs: Optimisation Globale, Arithmétique Garantie, Programmation Par Contraintes, Équation Différentielle Ordinaire, Intégration Garantie, Système Hybride.