



Simulation abstraite : une analyse statique de modèles Simulink

Alexandre Chapoutot

► To cite this version:

Alexandre Chapoutot. Simulation abstraite : une analyse statique de modèles Simulink. Génie logiciel [cs.SE]. Ecole Polytechnique X, 2008. Français. NNT : . tel-00366685

HAL Id: tel-00366685

<https://pastel.hal.science/tel-00366685>

Submitted on 9 Mar 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Thèse présentée pour obtenir le grade de

DOCTEUR DE L'ÉCOLE POLYTECHNIQUE

Spécialité : Informatique

par

Alexandre CHAPOUTOT

Simulation abstraite : une analyse statique de modèles Simulink

Soutenue le 8 décembre 2008 devant le jury composé de :

Marc POUZET	Université Paris-Sud, Orsay	<i>Président</i>
Jean-Marie CHESNEAUX	Université Pierre et Marie Curie, Paris	<i>Rapporteur</i>
Nicolas HALBWACHS	Vérimag, Gières	<i>Rapporteur</i>
Daniel KROB	Ecole Polytechnique, Palaiseau	<i>Examineur</i>
Bruno PAGANO	Esterel Technologies, Elancourt	<i>Examineur</i>
Matthieu MARTEL	Université de Perpignan Via Domitia, Perpignan	<i>Directeur de thèse</i>

Passage obligatoire dans la rédaction d'un manuscrit de thèse, je tiens, dans les quelques lignes qui suivent, à remercier toutes les personnes qui ont rendu ce travail possible.

Je remercie sincèrement Jean-Marie Chesneaux et Nicolas Halbwachs pour avoir rapporté cette thèse. La lecture attentive de ce manuscrit n'a pas dû être de tout repos. Pour ce travail et en particulier, pour leurs commentaires avisés et pertinents, je ne peux être qu'extrêmement reconnaissant.

Je suis très honoré de compter comme membres de mon jury Daniel Krob, Marc Pouzet et Bruno Pagano. Une mention particulière à Marc Pouzet pour avoir pris le temps d'échanger de manière très enrichissante sur les langages synchrones.

Je suis très reconnaissant envers mon directeur Matthieu Martel. J'ai pris énormément de plaisir à être son second doctorant. Il a toujours pris le temps de me guider et de m'écouter tout au long cette formation. De plus, son amour inconditionnel envers sa ville *Perpinyà* (le centre cosmique du monde d'après un certain Dalí) a su rendre plus agréable les réunions de travail. Pour toutes ces raisons *moltes gràcies*.

J'ai une grande gratitude envers Eric Goubault pour m'avoir accueilli au sein de son laboratoire et pour m'avoir offert des conditions idéales pour travailler en toute sérénité. Je remercie les membres de ce laboratoire : Sylvie Putot, Khalil Ghorbal, Asslé Adjé (alias Michel), Emmanuel Haucourt (pour sa patience infinie à expliquer les mathématiques), Sanjeevi Krishnan (alias *je ne comprends pas*).

Je n'oublie pas tous mes collègues du LSL qui m'ont accueilli lors de ma première année de thèse et de mon stage de master.

Des remerciements particuliers à Olivier Bouissou collègue de thèse mais avant tout ami. Ces trois années à ses côtés ont été essentielles pour accomplir ce travail. De plus, son sens aigu du savoir-vivre nous a permis de passer de très bons moments en particulier, pendant les séjours en conférence.

Mille mercis à Michel Hirschowitz et Nicolas Ayache pour leurs blagues et leur bonne humeur. Les pauses déjeuner dans le bureau 131 du bâtiment 528 resteront inoubliables ; les quatre, avec Olivier, nous formions une sacrée équipe.

Merci également à Xavier Allamigeon pour nos discussions interminables lors de ses trop rares venues au CEA. Il a également réussi à me communiquer sa passion de \LaTeX ¹.

Un grand merci à Patricia Mouy, Muriel Roger et Sarah Zennou ces drôles de dames de l'informatique. Merci pour tous les conseils que vous m'avez prodigués et pour les soirées passées en votre compagnie permettant d'oublier le travail l'espace d'un instant. Merci également à la famille Coyère.

Merci à ma maman sans qui je ne serais pas là. Merci à Ghislain pour sa bonne humeur. Merci à la famille Détré, Lili et Marion. Merci à la famille Détré-Dicarolo, Delphine, Tonio, Lucas, Lina et Fanette. Merci également à mon oncle Cuong pour son soutien.

Je clos ces remerciements par ceux adressés à Bérangère qui y tient une place importante dans ma vie. Merci d'avoir été à mes côtés pendant toutes ces années estudiantines et d'avoir pris soin de moi quand je n'avais pas le temps de m'occuper de toi.

Alexandre Chapoutot

¹La compilation de ce paragraphe produit un *Underfull \hbox*. J'espère que tu m'excuseras.

Table des matières

1	Introduction	5
1.1	La conception de systèmes embarqués critiques	6
1.1.1	Les normes de sécurité	6
1.1.2	Cycle de développement logiciel	8
1.1.3	Méthodes actuelles de validation du logiciel	9
1.2	Contributions	10
1.3	Plan	13
I	Contexte théorique et scientifique	15
2	Analyse statique de programmes	17
2.1	Ensembles ordonnés	18
2.2	Représentation et sémantique des programmes	19
2.3	Sémantique abstraite des programmes	22
2.4	Exemple : l'analyse d'intervalles	23
2.5	Validation des programmes à flots de données synchrones	25
2.5.1	Sémantique synchrone	25
2.5.2	Analyse statique de programmes synchrones	26
3	Outils d'étude des programmes numériques	29
3.1	Problématique de la précision numérique	29
3.2	Méthodes pour l'étude de la précision numérique	31
3.2.1	Arithmétique d'intervalles	31
3.2.2	Arithmétique stochastique	32
3.2.3	Différentiation automatique	34
3.2.4	Domaine des flottants avec erreurs	36
3.3	Intégration numérique	38
3.3.1	Problématique	38
3.3.2	Principales méthodes d'intégration	38
3.3.3	Intégration numérique garantie	40

II Analyse statique de Simulink 43

4	Présentation de Simulink	45
4.1	Langage Simulink	45
4.2	Equations sémantiques	49
4.3	Processus de simulation	53
4.4	Aperçu de l'analyse statique	54
5	Domaines abstraits	57
5.1	Domaine des formes de Taylor	57
5.1.1	Génération et arithmétique des formes de Taylor	58
5.1.2	Domaine concret	59
5.1.3	Domaine abstrait	59
5.2	Domaine des nombres flottants avec erreurs différentiées	63
5.2.1	Motivation	64
5.2.2	Domaine concret	66
5.2.3	Domaine abstrait	67
5.3	Domaine des séquences	68
5.3.1	Abstractions des séquences	68
5.3.2	Sémantique des équations	71
6	Analyse statique des modèles Simulink	73
6.1	Présentation de l'analyse statique	73
6.2	Sémantique des modèles pour une itération	76
6.2.1	Sémantique des modèles à temps discret	78
6.2.2	Sémantique des modèles à temps continu	80
6.2.3	Capteurs et actionneurs	85
6.2.4	Sémantique des modèles hybrides	89
6.3	Analyse statique	91
7	Expérimentation	97
7.1	Aspects logiciels de l'analyseur	97
7.1.1	Parser de fichiers mdl	97
7.1.2	Analyseur statique	100
7.2	Résultats expérimentaux	102
7.2.1	Détection de freinages	102
7.2.2	Contrôleur du papillon des gaz	103

III Réflexions et conclusion 107

8	Eléments de réflexion	109
8.1	Extensions du langage Simulink	109
8.2	Etude fréquentielle des systèmes	111
8.3	Propriétés temporelles	114
8.4	Validation multi-niveaux	114
9	Conclusion	117

Bibliographie	121
----------------------	------------

CHAPITRE 1

Introduction

L'intrusion des systèmes informatiques dans les appareils de notre quotidien est massive. Il est maintenant possible de trouver des logiciels dans la plupart des parties d'une voiture, d'un avion, d'un téléphone portable, ou les systèmes de contrôle d'imagerie médicale. Deux qualificatifs sont généralement associés à ces systèmes : l'adjectif *embarqué* qui caractérise la relation étroite entre le matériel et le logiciel ; et l'adjectif *critique* qui révèle l'aspect sécuritaire des systèmes, par exemple un régulateur de vitesse d'une voiture ou un dispositif anti-collision d'un avion. La sécurité est définie par : "l'absence de risque inacceptable, de blessure ou d'atteinte à la santé des personnes, directement ou indirectement, résultant d'un dommage matériel ou à l'environnement". Plus précisément, dans le cas des systèmes embarqués critiques, nous parlons de sécurité fonctionnelle c'est-à-dire d'une sécurité dépendante du bon fonctionnement d'un système en réponse à ses entrées. Il faut remarquer que la notion de sécurité, évoquée dans cette thèse, est associée à la notion de sûreté de fonctionnement (*safety* en anglais) et non à la sécurité des informations (*security*). Nous prenons comme référence dans ce document le vocabulaire utilisé dans les normes.

La problématique de la sécurité fonctionnelle dans les systèmes embarqués critiques est un enjeu majeur tant du point de vue industriel que du point de vue de la recherche académique. De nombreux exemples de défaillances, comme le cas du missile Patriot [GAO92] ou de la fusée Ariane 5 [Boa96] montrent d'une part que les conséquences sont importantes (une vingtaine de morts dans le cas du Patriot et des millions de dollars perdus pour la fusée Ariane) et, d'autre part, que les causes ne sont pas évidentes et nécessitent des outils d'étude appropriés. En outre, le remplacement des systèmes matériels par des systèmes électromécaniques a un impact non négligeable comme l'a montré la conception de l'Airbus A340¹, le premier avion à commande électrique (*fly-by-wire flight control*), qui a demandé la définition de nouveaux procédés de conception. De plus, la prévention des défaillances nécessite l'utilisation de méthodes de validation garantissant autant que faire ce peut le bon fonctionnement du système vis-à-vis de sa spécification.

Un autre point important à prendre en compte est la complexité croissante des systèmes embarqués (critiques ou non), qui rend la conception et la validation de plus en plus difficiles. Prenons comme exemple les nouvelles évolutions du système de régulation de la vitesse dans l'automobile. Ce système permet de maintenir automatiquement la vitesse d'une voiture en fonction du profil de la route. Il est maintenant possible de le combiner avec un radar anti-collision permettant ainsi de vérifier qu'une distance de sécurité est bien respectée². Ces systèmes découlant de la combinaison de systèmes plus simples ne peuvent pas être validés par composition de systèmes validés. En effet, la composition introduit de nouveaux comportements, qui n'existent pas dans les sous-systèmes

¹<http://www.aerospace-technology.com/projects/a340-200/>

²<http://fr.delphi.com/enfr/manufacturers/auto/safety/active/stopgo/>

séparés, ce qui nécessite une nouvelle validation globale. En général, la conception et la validation des systèmes embarqués critiques sont contraintes ou définies par des normes telle que la norme DO178B pour le domaine avionique ou la norme générique IEC 61508 (voir section 1.1).

Nous nous intéressons dans cette thèse à la validation de la partie logicielle des systèmes embarqués critiques (nommée *logiciel embarqué critique* dans la suite) par des méthodes d'analyse statique de programmes. La validation de ces logiciels requiert la prise en compte de l'environnement d'exécution (composant matériel et/ou grandeur physique) dans le but d'obtenir des résultats respectant au plus près les conditions d'utilisation. Cependant, ce paramètre supplémentaire, l'environnement d'exécution, élève sensiblement la difficulté du processus de validation des logiciels embarqués critiques. Néanmoins, il comble un manque important dans les méthodes de validation actuelles qui se concentrent sur le logiciel avec des hypothèses simplificatrices sur l'environnement d'exécution.

Cette première partie est organisée comme suit. Nous présentons, à la section 1.1, les contraintes normatives s'appliquant lors de la conception des systèmes embarqués critiques ainsi que la problématique liée à la validation. Dans la section 1.2, nous introduisons les travaux élaborés durant la thèse. Le plan de la thèse est donné à la section 1.3.

1.1 La conception de systèmes embarqués critiques

L'application des méthodes de validation dans l'industrie est essentiellement induite par les exigences de conception des systèmes embarqués critiques. Cependant, la difficulté, d'un point de vue industriel, est la possibilité de faire cohabiter simultanément les contraintes du marché et les contraintes sécuritaires. Le développement d'outils automatiques de validation est une réponse à ce problème. La compréhension de l'impact des normes sur la conception des systèmes embarqués critiques permet de mieux appréhender la nécessité de définir de tels outils.

1.1.1 Les normes de sécurité

Les normes de sécurité régissant la conception de systèmes embarqués critiques se basent sur deux concepts : le cycle de vie de sécurité et le niveau de sécurité. Le cycle de vie de sécurité représente les étapes de vie du logiciel, en incluant les exigences de la sécurité fonctionnelle (qualité de développement, documentation, etc.) de la conception au possible démantèlement. Le niveau de sécurité permet de classer les systèmes suivant leur niveau de criticité, c'est-à-dire suivant l'importance des dommages qu'une défaillance peut engendrer.

Nous présentons brièvement dans cette section la norme IEC 61508 [IEC01] publiée de 2001. Cette norme définit une approche générale de la sécurité fonctionnelle de tous les systèmes comportant des dispositifs électriques, électroniques ou électroniques programmables (E/E/EP). Son objectif principal est de définir des normes spécifiques à des domaines d'application plus restreints comme la norme IEC 61513 pour le nucléaire ou la norme EN 5012 x ($x=6$: système, $x=8$: logiciel ou $x=9$: matériel) pour le ferroviaire. La future norme automobile ISO 26262 sera elle aussi issue de la norme IEC 61508. Une conséquence induite par cette norme est qu'elle est applicable dans des secteurs d'activité qui n'ont pas encore de norme de sécurité. Il faut remarquer que le domaine de l'avionique civile s'est doté d'une norme de sécurité pour les logiciels, la DO178B, de façon indépendante. Cependant, ces deux normes ont beaucoup de caractéristiques communes.

La norme détermine un ensemble de règles, techniques et méthodes à appliquer pour un niveau de sécurité visé. Les prescriptions de la sécurité fonctionnelle sont décrites dans les sept parties qui la composent. Ces parties énumèrent les prescriptions de sécurité affectant les différents éléments du système qu'ils soient matériels (partie 2) ou logiciels (partie 3). Cette norme est orientée moyen et non pas but, c'est-à-dire qu'elle ne vise pas la certification ou la qualification des systèmes conçus suivant ces recommandations, à l'opposé de la DO178B. Néanmoins, la démarche rigoureuse décrite dans les documents qui la composent permet d'accroître la confiance dans les systèmes et donc elle facilite une possible qualification ou certification.

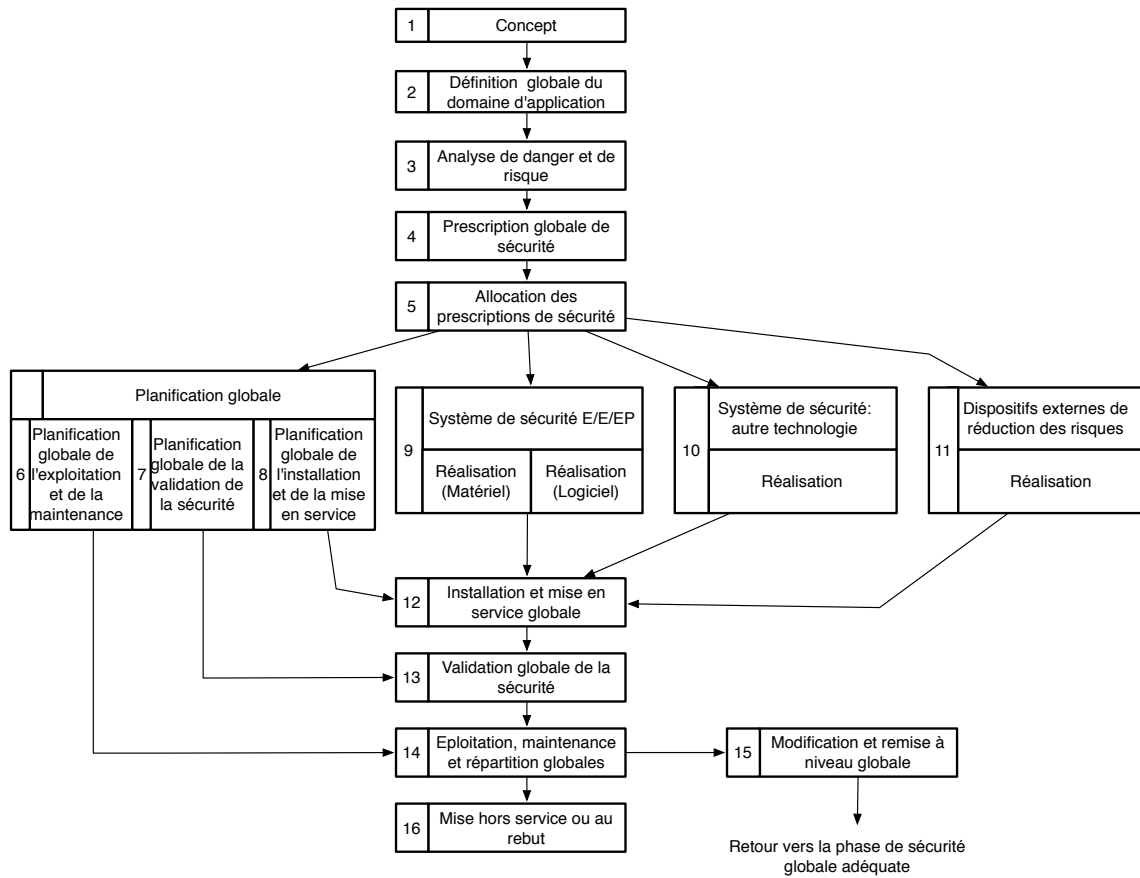


FIG. 1.1 – Cycle de vie de la sécurité.

La figure 1.1 représente le cycle de vie de sécurité tel qu'il est défini dans la partie 1 de la norme IEC 61508. Nous pouvons remarquer que les aspects sécuritaires sont présents dès le début du cycle et qu'ils gouvernent par ailleurs tout le reste. Les phases 3, 4 et 5 de ce cycle de vie de sécurité permettent d'attribuer à chaque partie du système un niveau d'intégrité de sécurité (SIL : *Safety Integrity Level*) et déterminent les moyens à mettre en œuvre pour l'atteindre. De plus, nous remarquons qu'il est possible d'associer dans le cycle de vie des éléments extérieurs au système, phases 10 et 11, dont le rôle est de limiter les risques, par exemple la redondance des éléments critiques. La phase d'exploitation est nécessairement précédée par une phase de validation des exigences de sécurité sur l'ensemble des composants du système. En général, c'est un organisme externe, lors de l'étape de *certification*, qui s'assure que les exigences de sécurité sont satisfaites, et qui autorise la mise en exploitation.

Il existe quatre niveaux d'intégrité, les systèmes de niveau 4 sont les plus critiques et ceux de niveau 1 sont les moins dangereux. Implicitement le niveau 0 représente les parties non sécuritaires du système et elles ne sont pas prises en compte dans cette norme. C'est la phase d'analyse de risque qui va déterminer les niveaux des composants d'un système. Le niveau d'intégrité d'un système est, en général, celui de sa composante du niveau le plus faible mais il existe des règles de composition qui ne respectent pas cette règle. Les exigences de sécurité dépendent du mode de fonctionnement des systèmes.

La norme discrimine les systèmes suivant deux modes de fonctionnements : le *mode continu* représentant les systèmes de commande assurant un contrôle, et le *mode à la demande* désignant ceux qui sont destinés à réagir à certaines conditions. Le régulateur de vitesse est un exemple de système en mode continu tandis que le système d'arrêt d'urgence d'une machine outil est un

exemple de système en mode à la demande. Dans le cas du mode à la demande, la norme définit la notion de probabilité de défaillances à la demande, PFD (*Probability of Failure on Demand*), exprimée en taux de défaillances par an. Dans le cas du mode continu, la notion de taux de défaillances dangereuses, notée λ , est exprimée en nombre d'occurrences par heure. Les différents niveaux d'intégrité de sécurité définis dans la norme, ainsi que les taux de défaillances associés, par exemple le niveau SIL 4 des systèmes en mode continu exigent un taux entre 10^{-8} et 10^{-9} . L'élément important est la fréquence très basse d'occurrence d'une défaillance pour les modes continu ou à la demande.

Le niveau d'intégrité de sécurité décroît avec le temps (en raison de l'usure du matériel par exemple). Il faut alors définir un intervalle de test et de maintenance périodique afin que le niveau d'intégrité soit conforme avec les exigences de sécurité tout au long du cycle de vie.

1.1.2 Cycle de développement logiciel

Nous considérons maintenant plus particulièrement la troisième partie de la norme IEC 61508 qui décrit la phase de réalisation des logiciels relatifs à la sécurité. Cette présentation permet de mieux appréhender les spécificités et les contraintes liées au développement de logiciels embarqués critiques.

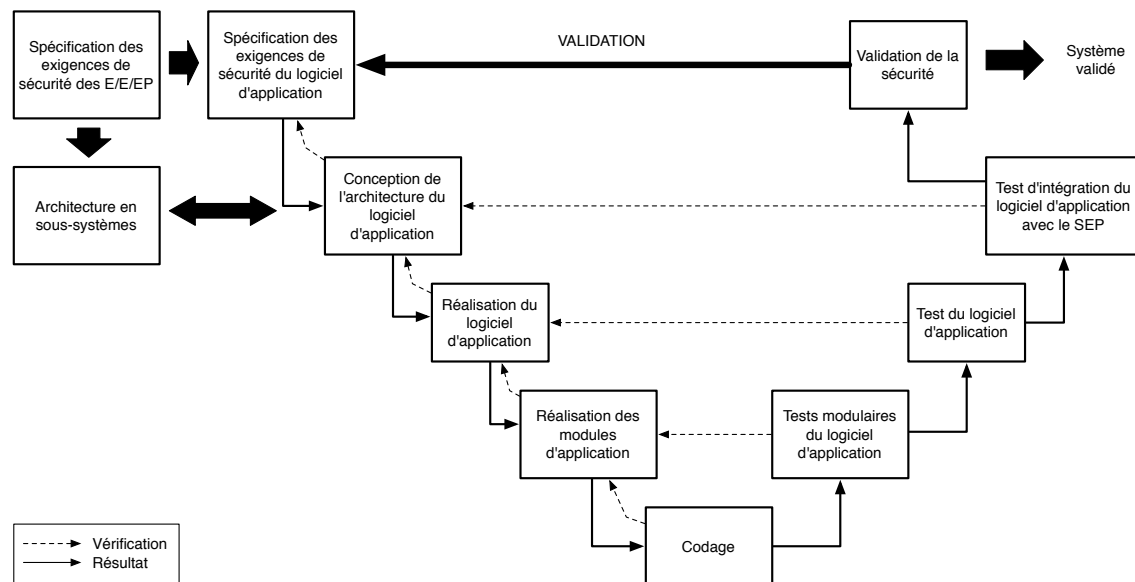


FIG. 1.2 – Cycle de développement de logiciels : modèle en V.

Il existe plusieurs cycles de développement de logiciels [Som06] mais le plus rencontré dans l'industrie et celui qui est préconisé par la norme, est basé sur le modèle en V. La figure 1.2 décrit les différentes étapes de ce cycle de développement telles qu'elles sont définies dans la partie 3 de la norme IEC 61508. La phase descendante représente les étapes de spécifications et la phase montante correspond aux étapes de validation. Nous remarquons en particulier que cette méthodologie met en relation une phase de test à chaque étape de la conception ainsi qu'une phase de vérification par d'autres méthodes comme par exemple la relecture de code. Par ailleurs, il faut souligner le lien étroit qu'il existe entre le développement de la partie matérielle et celui de la partie logicielle. L'intérêt est de s'assurer au plus tôt dans le cycle de développement que les choix de conception du logiciel sont cohérents vis-à-vis du matériel. Il semble alors important de prendre en compte, lors de la validation du logiciel, les spécificités matérielles et donc l'environnement d'exécution du logiciel.

De plus, la norme, en plus de définir les étapes de conception, suggère quelques techniques et méthodes à utiliser qui sont en accord avec le niveau de sécurité visé. Par exemple, il est très fortement recommandé d'utiliser lors du développement, soit un compilateur certifié (par exemple le compilateur Ada³), soit un compilateur dont la confiance résulte de l'utilisation (par exemple gcc⁴) ; la conséquence est le nombre limité d'outils respectant ces critères. Un autre exemple est l'utilisation de langages fortement typés ou encore l'utilisation de règles de codage. Les contraintes liées au développement du logiciel embarqué critique engendrent une classe de programmes très spécifique. Elle est, en général, réduite car elle ne contient pas des cas réputés comme difficiles à étudier ou à comprendre, par exemple les programmes utilisant l'instruction *goto* [Dij68].

1.1.3 Méthodes actuelles de validation du logiciel

L'activité de test est la méthode de validation préconisée dans les normes de sécurité. Cependant, le test ne peut pas être exhaustif et il ne garantit pas l'absence d'erreurs [Mye79, Chap. 4]. Les méthodes formelles, basées sur des concepts mathématiques, permettent de pallier le problème de l'activité de test en offrant la possibilité de raisonner mathématiquement sur les propriétés d'un programme. Pour mettre en évidence la différence entre ces deux méthodes : l'activité de test vérifie si un système, soumis à une situation particulière, a un comportement correct par rapport à sa spécification, tandis que les méthodes formelles vérifient l'adéquation du système à sa spécification pour n'importe quelle situation. Les méthodes formelles permettent de *prouver* des propriétés mais elles sont plus difficiles à mettre en œuvre. Nous constatons que les méthodes formelles sont alors un choix naturel de méthode de validation au vue des exigences de sécurité induites par les normes.

Les récents résultats sur l'application des techniques d'analyse statique par interprétation abstraite sur des programmes embarqués critiques de grande taille [BCC⁺03, CCF⁺05] ou la vérification de programmes par techniques de vérification de modèles (*model checking*) [HJMS03] jouent en faveur de l'utilisation de telles méthodes dans le cycle de développement. Ces méthodes automatiques fournissent des preuves de propriétés telles que l'absence d'erreurs à l'exécution [Cou07] ou de satisfiabilité de propriétés temporelles [HJM⁺02]. Ces applications des méthodes formelles montrent, bien que la mise en œuvre soit complexe, la faisabilité de l'utilisation de telles techniques dans le processus de développement de systèmes embarqués critiques.

Cependant, la validation des programmes par des méthodes d'analyse statique ou par des méthodes de vérification de modèles font en général des hypothèses simplificatrices sur l'environnement d'exécution. Typiquement, les entrées provenant de capteurs sont approchées par le domaine de valeurs de ceux-ci ; la conséquence est l'oubli de la dynamique des valeurs mesurées (par exemple l'entrée est la valeur d'un courant sinusoïdal). Ces simplifications ont pour effet de rendre, dans certaines circonstances, des résultats peu précis et peu en adéquation avec les conditions d'utilisation. Le constat [Cou05] est qu'il semble alors important de prendre en compte l'environnement d'exécution dans la phase de validation du logiciel.

L'utilisation de plus en plus fréquente d'outils de conception tels que Lustre/SCADE⁵ ou Matlab/Simulink⁶ permet de contenir la complexité croissante des systèmes en offrant un niveau d'abstraction supérieure. Elle apporte également de nouvelles perspectives sur l'application de méthodes formelles dans le cycle de développement des logiciels critiques embarqués. En effet, ces outils, en particulier Matlab/Simulink, rendent possible la modélisation de l'environnement d'exécution et de la partie logicielle dans un même formalisme. Cette nouvelle façon de concevoir les systèmes embarqués critiques se situe dans la nouvelle mouvance de la conception basée sur les modèles (*model-based design*) [SM04]. L'idée principale est de dériver le logiciel embarqué critique directement à partir des spécifications afin de limiter les erreurs et de gagner en rapidité de développement. La majeure partie des méthodes est orientée conception de logiciels et elle est souvent basée sur le langage UML [MH06].

³<http://www.adacore.com/home/>

⁴<http://gcc.gnu.org>

⁵Marque de Esterel Technologies

⁶Marque de Mathworks

La particularité des langages comme Lustre/SCADE ou Matlab/Simulink est qu'ils se placent dans les langages métiers (*Domain Specific Languages*) [Con04]. Ce sont des outils spécialisés dans des domaines particuliers comme l'automatique ou la conception de systèmes de contrôle. L'avantage est de s'adresser directement aux concepteurs des logiciels embarqués critiques en leur fournissant des outils adaptés. C'est une des principales forces de ces outils par rapport à ceux basés sur UML. De plus, les méthodes basées sur UML nécessitent une adaptation pour prendre en considération les spécificités métiers et elles permettent, en général, des descriptions de haut niveau qui ne contiennent pas encore les choix d'implémentation.

Des travaux récents sur la combinaison des formalismes UML avec le langage Lustre/SCADE [GD06] présagent des futures méthodes de conception de logiciels embarqués critiques. Mais, à ce jour, les spécifications en Lustre/SCADE ou Matlab/Simulink semblent être le niveau adéquat pour appliquer les méthodes formelles. Elles permettent d'avoir une vue d'ensemble du système (partie logicielle et environnement d'exécution) avec suffisamment de détails pour valider les comportements vis-à-vis de la spécification.

1.2 Contributions

Les travaux développés dans cette thèse sont essentiellement centrés sur l'application des méthodes d'analyse statique par interprétation abstraite aux programmes Matlab/Simulink, appelés modèles dans la suite en accord avec la terminologie Simulink. Simulink est le standard *de facto* dans les outils d'aide à la conception. L'application de l'analyse statique permet d'une part de définir une méthode automatique non intrusive des spécifications, c'est-à-dire s'insérant naturellement dans le processus de développement, et, d'autre part, de bénéficier des informations de haut niveau offrant la possibilité de prendre en compte l'environnement d'exécution des logiciels critiques embarqués.

La principale contribution est la définition d'une méthode automatique d'évaluation de la qualité numérique des exécutions des modèles Simulink, c'est-à-dire des simulations numériques. À partir des outils issus de la théorie de l'interprétation abstraite et de l'analyse numérique, nous avons défini et mis en œuvre une méthode permettant de mesurer la distance entre un modèle mathématique des systèmes embarqués critiques, décrit dans le langage Simulink, et l'exécution de ce modèle telle qu'elle est réalisée par Simulink. L'objectif de cette méthode est de fournir une information essentielle lors de la conception des logiciels embarqués critiques. Cette information, la qualité numérique des simulations, permet d'étudier les propriétés du modèle mathématique vis-à-vis de son exécution et ainsi donne un critère de validité du modèle. Par ce biais, nous sommes capables de séparer les problèmes liés à une mauvaise modélisation de ceux introduit par les approximations numériques.

Autrement dit, cette contribution est la définition d'une analyse statique de modèles Matlab/Simulink. Elle est présentée plus en détail au chapitre 4. Matlab/Simulink permet la modélisation de systèmes dynamiques c'est-à-dire évoluant au cours du temps. Plus précisément, il est possible de décrire des systèmes à temps continu, à temps discret ou des combinaisons de ces deux types. Il est donc très bien adapté pour décrire un logiciel (partie discrète) et son environnement d'exécution (partie continue) des systèmes embarqués critiques.

En effet, un logiciel embarqué critique de contrôle a pour fonction de maintenir un système dans un état suivant les comportements d'un élément observé. La figure 1.3 donne un exemple de système de contrôle, un régulateur du papillon des gaz dans une automobile, qui permet de comprendre le schéma général de ces systèmes. Un programme (le régulateur électronique) mesure périodiquement au travers de capteurs des valeurs issues de l'environnement d'exécution (le papillon des gaz), traite celles-ci, puis agit en fonction des résultats du traitement, sur cet environnement au travers d'actionneurs (ouvre plus ou moins le papillon grâce à un moteur électrique).

Matlab/Simulink est un langage graphique composé de boîtes (*blocks*) et de fils reliant ces boîtes (*signals*). Les boîtes représentent les opérations (par exemple opérations arithmétiques) et les fils représentent les valeurs, c'est-à-dire les fonctions du temps, qui sont transformées, par ces opérations. Le formalisme utilisé par Matlab/Simulink est très proche de celui utilisé par les

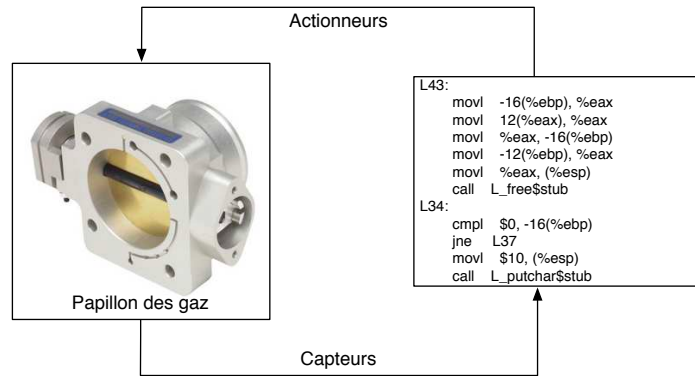


FIG. 1.3 – Vue schématique d'un système de contrôle.

automaticiens : les diagrammes en blocs (*block-diagrams*) qui décrivent les fonctions d'un système au travers d'équations de flots de données. Par exemple, il est très utilisé en traitement du signal [Jac89] ou encore en description d'architectures matérielles. L'avantage de ce formalisme est de permettre aussi bien un développement ascendant (*bottom-up*), du moins détaillé au plus détaillé, que l'approche inverse c'est-à-dire descendante (*top-down*).

Un des avantages des descriptions Matlab/Simulink est qu'elles sont exécutable, en d'autres termes ce sont des programmes informatiques. Matlab/Simulink utilise des techniques de l'analyse numérique pour simuler le comportement des systèmes décrits dans son formalisme graphique. Ces simulations permettent de tester, au sens de l'activité de test logiciel, si leurs comportements sont conformes aux attentes des concepteurs.

L'utilisation de techniques issues de l'analyse numérique pose le problème de l'influence des approximations numériques sur les exécutions de ces programmes. Il est connu que les calculs sur ordinateur avec une précision finie sont forcément entachés d'erreurs [Gol91]. Les erreurs proviennent soit des arrondis des résultats des calculs, soit des données partiellement représentées (par exemple π) ou encore de la méthode de calcul utilisée (par exemple algorithme d'intégration). Cet ensemble d'approximations peut avoir des effets non désirés mais difficiles à mettre en évidence. L'étude de ces approximations permet de valider que le modèle, c'est-à-dire la spécification du système, est suffisamment précise ou robuste pour être utilisée dans les prochaines étapes de conception. Nous donnons à la figure 1.4 un exemple de problème numérique. Le programme calcule $\exp(x)$ par une approximation polynomiale fondée sur un développement de Taylor de la fonction exponentielle, qui est $\exp(x) = 1 + x + x^2/2 + x^3/3! + \dots$. Le résultat pour $x = -25$ est $-7.129780403672078e^{-7}$ ce qui est, de toute évidence, incorrect. Le problème est induit par une élimination catastrophique (voir section 3.1) qui se produit au niveau des ordres 24 et 25 et qui affecte le résultat final. L'exemple précédent met en évidence la facilité d'obtenir sur des

```
double exp (double x) {
    int i = 0;
    double somme = 0.0;
    double terme = 1.0;
    for (i = 0; somme != somme + terme; i++) {
        somme = somme + terme;
        terme = terme * x / i;
    }
}
```

FIG. 1.4 – Approximation polynomiale de la fonction exponentielle.

ordinateurs des résultats de calculs faux. La défaillance du missile Patriot (erreur d'arrondi) et

de la fusée Ariane 5 (dépassement de capacité) sont directement issus de problème de précision numérique. Nous pouvons citer également la défaillance du porte avion USS Yorktown⁷ qui s'est arrêté de fonctionner en pleine mer en 1998, pendant trois heures, à cause d'une division par zéro. Tous ces exemples mettent en évidence l'intérêt d'étudier ce genre de propriété lors de la conception des logiciels embarqués critiques.

Les travaux développés dans cette thèse sont d'une part, une amélioration de l'analyse statique de propriétés numériques liées à l'arithmétique flottante ; d'autre part, la définition d'une analyse statique de modèles Matlab/Simulink permettant de valider les simulations numériques de ceux-ci.

Etude de la précision numérique : Les résultats présentés dans [CM07a, CM08a] sont une amélioration d'une analyse statique de programmes numériques basée sur les travaux de [Mar06] et sur la comparaison des méthodes d'étude de la précision numérique [Mar05]. En effet, il existe plusieurs méthodes issues de l'analyse numérique permettant d'étudier la précision numérique comme l'arithmétique d'intervalles [Moo79], l'arithmétique stochastique [Che95] ou la différentiation automatique [Gri00]. Chacune de ces méthodes a des avantages comme la garantie des résultats en arithmétique d'intervalles mais aussi des inconvénients comme une sur-approximation des résultats pour cette même arithmétique.

Le point de départ a été de vouloir bénéficier des avantages des méthodes tout en éliminant les inconvénients ou du moins en réduisant leurs effets. L'amélioration de l'étude de la précision numérique, décrite dans cette thèse, dans le cadre de l'analyse statique de programmes, est issue d'une combinaison des techniques de la différentiation automatique et de l'arithmétique d'intervalles dans le cadre de la théorie de l'interprétation abstraite.

Cette combinaison permet de définir un domaine numérique abstrait relationnel. Plus précisément, elle permet de définir des formes de Taylor qui captent des relations de dépendance entre les erreurs d'arrondi apparaissant au fil des exécutions d'un programme. Ces relations permettent de limiter les sur-approximations induites par l'arithmétique d'intervalles et donc réduisent ainsi le nombre de faux positifs lors d'une analyse. Un faux positif est la détection d'une erreur pendant l'analyse qui n'existe pas dans le système réel et elle est issue des sur-approximations.

Les travaux liés à l'étude de la précision numérique, développés initialement dans le contexte de programmes numériques, seront présentés de manière unifiée dans le cadre de l'analyse statique de modèles Matlab/Simulink.

Analyse statique de programmes Matlab/Simulink : Les systèmes mêlant des parties à temps continu et des parties à temps discret s'inscrivent dans le domaine des *systèmes hybrides*. La majeure partie des résultats dans ce domaine est liée à la théorie des automates hybrides [ACHH93]. L'analyse statique de modèles Matlab/Simulink [CM08b] permet une nouvelle approche dans l'étude des systèmes hybrides. L'approche de la vérification de systèmes hybrides (en particulier les automates hybrides) par des méthodes de vérification de modèles (*model checking*) se concentre sur le modèle mathématique et ses propriétés. L'application de ces méthodes pour la vérification de modèles Simulink demande un changement de formalisme [CCM⁺03a, SSC⁺04, HKSP03, Tiw02, ASK04].

La conversion des modèles Matlab/Simulink en programmes Lustre [CCM⁺03a, SSC⁺04] s'intéresse uniquement à la partie discrète des programmes Matlab/Simulink. Cette méthodologie souffre, dans le contexte de la validation, des mêmes limitations que l'application des méthodes formelles au niveau du code : elle ne prend pas en compte l'environnement d'exécution. La transformation en automates hybrides [Tiw02, ASK04] permet de prendre en compte tous les aspects des modèles Matlab/Simulink. Cependant, les méthodes de vérifications imposent, en général, des contraintes sur les parties à temps continu, par exemple qu'elles soient linéaires ou polynomiales.

L'analyse statique de modèles hybrides Simulink permet de valider les comportements numériques vis-à-vis des comportements mathématiques de ces programmes. Nous obtenons d'une part les comportements mathématiques ainsi que les comportements "machine" et d'autre part,

⁷Pour plus de détails, voire http://www.gcn.com/print/17_17/33727-1.html.

nous fournissons un critère de correction c'est-à-dire une comparaison entre ces deux comportements. Dans le cadre des logiciels embarqués critiques, nous offrons une nouvelle méthode de validation des comportements des logiciels par rapport à leurs spécifications en étant plus proches de leurs conditions d'utilisation.

Une partie importante de ce travail a été de définir une analyse statique des modèles Simulink mimant au mieux les techniques numériques de simulation. Les avantages d'une telle méthode sont de travailler avec le même formalisme que les ingénieurs concepteurs du système et d'utiliser les mêmes outils issus de l'analyse numérique que ceux de Matlab/Simulink et donc de ne pas limiter ou de contraindre les parties discrètes et continues des systèmes.

1.3 Plan

Ce document est composé de trois parties présentant le contexte scientifique, les résultats théoriques, et les perspectives. Nous décrivons, dans cette section, l'organisation de cette thèse chapitre par chapitre.

Première partie : Cette thèse se place dans un contexte théorique au centre de plusieurs domaines scientifiques qui sont l'analyse statique par interprétation abstraite [CC77], l'analyse numérique [Neu01, Jed06] et les langages synchrones [Hal93]. Cette première partie est une présentation des techniques utilisées dans chacun de ces domaines. L'analyse statique de programmes par interprétation abstraite est présentée de façon générale dans le chapitre 2. En particulier, les langages synchrones à flots de données et l'application des méthodes d'analyse statique sur ces langages sont introduites à la section 2.5. Le chapitre 3 décrit les outils utiles à l'étude de la précision numérique ainsi que les outils de l'analyse numérique, en particulier ceux liés à l'intégration numérique.

Seconde partie : Les travaux et les résultats issus de la thèse sont regroupés dans cette seconde partie. Une présentation du langage Matlab/Simulink ainsi que la génération des équations sémantiques sont données au chapitre 4. Nous définissons au chapitre 5 les différents domaines abstraits qui seront utilisés pour mettre au point l'analyse statique de modèles Simulink. A la section 5.1 est décrite une formalisation des formes de Taylor dans le contexte de la théorie de l'interprétation abstraite. Elle est par la suite utilisée dans la définition d'une amélioration du domaine des flottants avec erreurs à la section 5.2. A la section 5.3, nous définissons un domaine des séquences, inspiré du paradigme des langages synchrones à flots de données. La sémantique des modèles à temps continu, à temps discret et hybrides est définie au chapitre 6 en utilisant les précédents domaines. Des résultats expérimentaux sont commentés au chapitre 7.

Troisième partie : Cette dernière partie décrit quelques pistes de recherche, en particulier au chapitre 8, sur l'analyse statique de modèles Matlab/Simulink. D'une part, les extensions du langage, en prenant en compte des machines à états décrites en Stateflow, sont envisagées à la section 8.1. L'étude fréquentielle des systèmes est une technique classique en automatique. La section 8.2 est un début de formalisation, dans la théorie de l'interprétation abstraite d'analyse fréquentielle, dans le cas particulier de modèles Matlab/Simulink linéaires. L'étude de propriétés fonctionnelles telle que l'influence des approximations sur la structure de contrôle des modèles Matlab/Simulink est présentée à la section 8.3. La section 8.4 esquisse les premières idées sur l'utilisation de Matlab/Simulink dans le processus de validation des spécifications et du code.

Première partie

Contexte théorique et scientifique

Commissaire Adamsberg : « Opération calamiteuse, Mordent. Je ne sais pas au juste ce que vous essayez de faire mais la prochaine fois, faites-le mieux. »

Fred Vargas, *Un lieu incertain*.

L'analyse statique de programmes Matlab/Simulink s'appuie sur des travaux issus de plusieurs domaines scientifiques. Cette première partie présente les concepts importants des différentes théories ou méthodes qui ont été utilisées pour mener à bien les travaux de thèse.

Analyse statique de programmes

L'étude des langages de programmation est une des plus anciennes disciplines de l'informatique théorique et plus précisément en théorie de la compilation [ALSU06, App98]. Les comportements des programmes sont décrits à l'aide de sémantiques telles que la sémantique opérationnelle, dénotationnelle ou encore axiomatique [Win93]. En particulier, la sémantique d'un programme P est une description formelle (c'est-à-dire mathématique) de tous les comportements possibles de P dans l'ensemble des environnements d'exécution possibles. L'automatisation de cette étude, notamment dans le cas de l'optimisation à la compilation ou la vérification de propriétés, a conduit à définir des techniques particulières pour répondre à ce besoin ; l'analyse statique est l'une d'entre elles.

L'analyse statique de programmes permet de prédire des comportements ou des ensembles de valeurs survenant à l'exécution, à partir du code source d'un programme. Autrement dit, l'étude sémantique de la représentation syntaxique d'un programme permet d'inférer des propriétés survenant à l'exécution mais sans l'exécuter. Les propriétés des programmes sont définies comme des ensembles de comportements qui respectent un ou plusieurs critères.

Les propriétés sur les programmes sont soit des propriétés de vivacité (*liveness property*), traduit par "quelque chose de bien va arriver", soit des propriétés de sûreté (*safety property*), traduit par "quelque chose de mauvais ne va pas arriver". Nous nous intéressons dans cette thèse à la validation de propriétés de sûreté et plus particulièrement par calcul d'invariants. Un invariant est une propriété vraie pour tous les comportements possibles du programme. Le domaine des valeurs des variables d'un programme est un exemple d'invariant. Nous nous plaçons plus généralement dans le contexte de l'analyse du flot de données (*dataflow analysis*) [App98].

L'étude des propriétés des programmes informatiques est soumise à deux limitations théoriques :

- La sémantique d'un programme n'est pas calculable, en général ;
- Le théorème de Rice : "Toute propriété non triviale (ni toujours vraie, ni toujours fausse) sur la sémantique d'un langage de programmation est indécidable".

La théorie de l'interprétation abstraite [CC77, CC92, Cou81] a été définie pour remédier à ces limitations, en définissant une méthode de calcul approché par sur-approximation. L'analyse statique par interprétation abstraite est ainsi sûre ce qui signifie que les propriétés validées (c'est-à-dire prouvées) dans la description approchée (*abstraite*) correspondent aux propriétés de la description la plus précise (*concrète*).

L'analyse statique par interprétation abstraite s'appuie beaucoup sur les propriétés des ensembles partiellement ordonnés [Bir67, Sch02]. Nous rappelons les résultats importants de cette théorie, et qui sont utilisés dans ce chapitre, à la section 2.1. Puis à la section 2.2, nous définissons

la sémantique d'un fragment de langage impératif. Nous présentons les concepts importants de la théorie de l'interprétation abstraite à la section 2.3 en définissant une sémantique abstraite et nous illustrons ces idées par l'exemple de l'analyse d'intervalles à la section 2.4. Les langages à flots de données synchrones possèdent beaucoup de caractéristiques communes avec le sujet d'étude de cette thèse : le langage Simulink. Nous décrivons brièvement, à la section 2.5, la sémantique de ces langages ainsi que les techniques de validation appliquées à ceux-ci.

2.1 Ensembles ordonnés

Nous présentons dans cette section les principaux résultats liés aux ensembles partiellement ordonnés qui sont à la base de l'analyse de programmes. Cette section est largement inspirée de [Sch02, Win93].

Définition 2.1 (Poset) Un ensemble E muni d'une relation \sqsubseteq réflexive, antisymétrique et transitive est un ensemble partiellement ordonné (*poset*), noté $\langle E, \sqsubseteq \rangle$.

Exemple 2.1 L'ensemble des entiers naturels \mathbb{N} muni de la relation d'ordre "plus petit ou égale à", notée \leq , est un poset.

Exemple 2.2 L'ensemble des parties d'un ensemble X , noté $\wp(X)$, muni de la relation d'inclusion \subseteq est un poset.

Définition 2.2 (Borne supérieure et borne inférieure) Soit $D \subseteq E$, $b \in E$ est une borne supérieure (respectivement inférieure) de D si :

$$\forall a \in D, a \sqsubseteq b \text{ (resp. } b \sqsubseteq a \text{)}$$

b est la plus petite borne supérieure (resp. inférieure) de D , notée $\sqcup D$ (resp. $\sqcap D$), si b est une borne supérieure (resp. inférieure) de D et si pour toute les bornes supérieure (resp. inférieure) c de D , $b \sqsubseteq c$ (resp. $c \sqsubseteq b$).

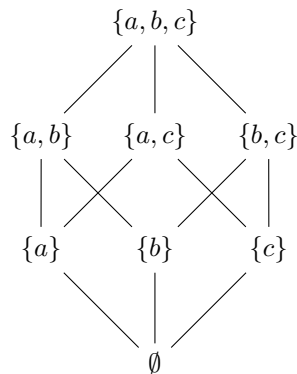
Exemple 2.3 Le poset $\langle \mathbb{N}, \leq \rangle$ possède une borne inférieure 0 mais pas de borne supérieure.

Exemple 2.4 Le poset $\langle \wp(X), \subseteq \rangle$ possède une borne inférieure \emptyset et une borne supérieure X .

Définition 2.3 (Treillis) Un poset est un treillis, noté $\langle E, \sqsubseteq, \sqcup, \sqcap \rangle$ si et seulement si :

$\forall a, b \in E$ il existe un plus petite borne supérieure et il existe une plus grande borne inférieure.

Exemple 2.5 Le poset $\langle \wp(X), \subseteq \rangle$ est un treillis complet. Soit $X = \{a, b, c\}$, le diagramme de Hasse (représentation graphique des ensembles ordonnés) associé à $\langle \wp(X), \subseteq \rangle$ est :



Et par exemple, la borne supérieure des éléments $\{a\}$ et $\{b, c\}$ de $\wp(X)$ est l'élément $\{a, b, c\}$.

Définition 2.4 (Treillis complet) Un *poset* est un treillis complet, noté $\langle E, \sqsubseteq, \perp, \top, \sqcup, \sqcap \rangle$, si et seulement si :

$\forall S \subseteq E$ a un plus petite borne supérieure $\sqcup S$ et une plus grande borne inférieure $\sqcap S$ avec $\perp = \sqcup \emptyset$ et $\top = \sqcap E$.

Exemple 2.6 Le *poset* $\langle \wp(X), \subseteq \rangle$ est un treillis complet.

Définition 2.5 (Chaîne) Une chaîne d'un *poset* $\langle E, \sqsubseteq \rangle$ est une partie de E totalement ordonnée.

Exemple 2.7 Le *poset* $\langle \mathbb{N}, \leq \rangle$ est une chaîne.

Définition 2.6 (c.p.o.) Un ordre partiel complet (*complete partial order* ou *c.p.o.*) est un *poset*, noté $\langle E, \sqsubseteq, \perp, \sqcup \rangle$, avec une borne inférieure \perp tel que pour toute chaîne C de E , C possède une plus petite borne supérieure.

Nous rappelons dans la suite les propriétés et les résultats associés aux fonctions entre ensembles partiellement ordonnés.

Définition 2.7 (Croissance) Une fonction f entre deux *poset* $\langle E, \sqsubseteq_E \rangle$ et $\langle L, \sqsubseteq_L \rangle$ est croissante (*monotone* en anglais) ou préserve l'ordre si et seulement si :

$$\forall x, y \in E, x \sqsubseteq_E y \Rightarrow f(x) \sqsubseteq_L f(y).$$

Exemple 2.8 La fonction $\mathbf{f} : \langle \mathbb{N}, \leq \rangle \rightarrow \langle \mathbb{N}, \leq \rangle$ défini par $\mathbf{f}(x) = 2 \times x$ est croissante.

Définition 2.8 (Continuité) Une fonction \mathbf{f} entre deux *poset* $\langle E, \sqsubseteq_E \rangle$ et $\langle L, \sqsubseteq_L \rangle$ est continue si et seulement si \mathbf{f} préserve les possibles plus petites bornes supérieures des chaînes croissantes de E .

Définition 2.9 (Point fixe) Soit un *poset* $\langle E, \sqsubseteq \rangle$ et soit $\mathbf{f} : E \rightarrow E$ une fonction continue. Alors $p \in E$ est un point fixe de \mathbf{f} si et seulement si $\mathbf{f}(p) = p$. L'ensemble E possède la propriété de point fixe si toute fonction continue $\mathbf{f} : E \rightarrow E$ a un point fixe. L'ensemble des points fixes de E est :

$$\text{Fix}(\mathbf{f}) = \{p \in E : \mathbf{f}(p) = p\}.$$

Théorème 2.1 (Théorème de Tarski-Davis) Soit E un treillis, E a la propriété de point fixe si et seulement si E est complet. Dans ce cas, pour chaque fonction continue $\mathbf{f} : E \rightarrow E$ l'ensemble $\text{Fix}(\mathbf{f})$ est un treillis complet.

Preuve La preuve de ce théorème est donnée dans [Sch02, section 5.2, p. 114].

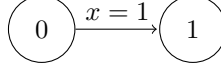
2.2 Représentation et sémantique des programmes

Nous considérons qu'un programme P est représenté par un graphe de flots de contrôle $G[P]$ dont les arcs sont étiquetés par les instructions i de P et nous désignons par $I[P]$ l'ensemble fini des instructions de P . Les instructions représentent une affectation ou une comparaison entre une variable et une constante. Plus précisément, nous considérons le langage défini par la grammaire décrite à la figure 2.1, la règle *inst* décrit ces instructions. La règle *a* représente la construction des expressions arithmétiques composées de constantes entières r , de variables x appartenant à l'ensemble fini $V[P]$ des variables du programme, et des opérations arithmétiques $+$, $-$, \times , \div .

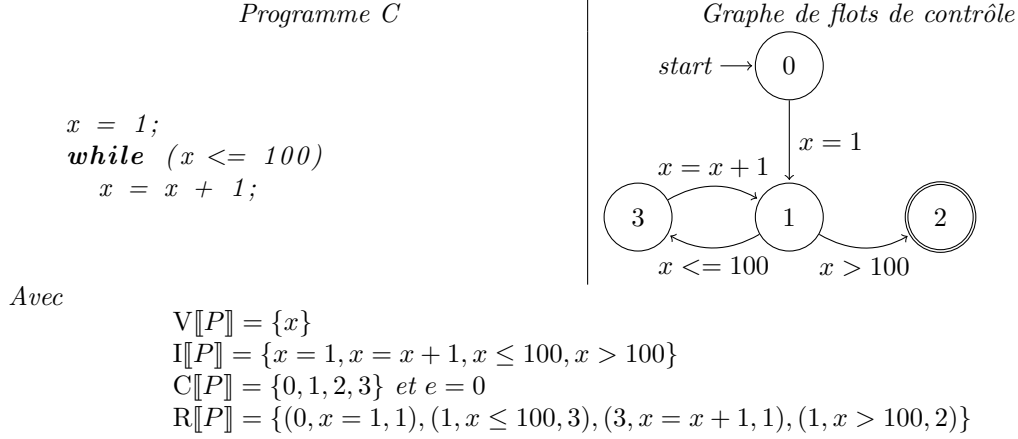
Les nœuds c de $G[P]$ représentent les points de contrôle du programme, c'est-à-dire les parties du programme importantes pour son étude. Il n'y a qu'un nombre fini de points de contrôle et nous notons $C[P]$ l'ensemble des points associés à P . Le choix des points de contrôle est effectué lors de la phase d'analyse grammaticale et il ne change pas pendant l'analyse. Un point d'entrée e est associé à $G[P]$ représentant le point d'entrée du programme ainsi qu'une relation $R[P]$ qui lie les points de contrôle et les instructions, $R[P] \in C[P] \times I[P] \times C[P]$. Cette relation décrit l'ensemble des arcs du graphe de flots de contrôle. Par exemple, l'élément $(0, x = 1, 1)$ de cet ensemble est représenté graphiquement par le graphe suivant :

$$\begin{aligned}
a &::= r \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 \div a_2 \\
inst &::= x = a \mid x \leq r \mid x < r \mid x \geq r \mid x > r
\end{aligned}$$

FIG. 2.1 – Grammaire des opérations sur les arcs du graphe de flots de contrôle.



Exemple 2.9 Le programme suivant calcule la valeur $x = 101$ à l'aide d'une boucle de 100 itérations. Nous donnons le graphe de flots de contrôle associé. Les points de contrôle représentent les états mémoire du programme avant et après chaque instruction.



Sémantique La sémantique d'un programme P est l'ensemble de tous les états de ce programme, noté $S[P]$, ainsi que les fonctions sémantiques qui, pour chaque instruction i de $I[P]$, définissent l'effet de i sur l'état courant. Nous notons σ l'environnement qui associe une valeur (dans notre cas un nombre entier dans \mathbb{Z}) à chaque variable, et nous appelons Σ l'ensemble des environnements. Un état du programme est alors décrit par un point de contrôle et un environnement, noté $\langle c, \sigma \rangle$.

La sémantique concrète (c'est-à-dire la plus précise) des expressions définit l'évaluation d'une expression a en une valeur entière dans un environnement σ . Nous notons $\llbracket \cdot \rrbracket_{\mathbb{A}}$ cette sémantique et elle est définie par :

$$\begin{aligned}
\llbracket r \rrbracket_{\mathbb{A}}(\sigma) &= r \\
\llbracket x \rrbracket_{\mathbb{A}}(\sigma) &= \sigma(x) \\
\llbracket a_1 \diamond a_2 \rrbracket_{\mathbb{A}}(\sigma) &= v_1 \diamond v_2 \text{ avec } v_1 = \llbracket a_1 \rrbracket_{\mathbb{A}}(\sigma), v_2 = \llbracket a_2 \rrbracket_{\mathbb{A}}(\sigma) \text{ et } \diamond \in \{+, -, \times, \div\}.
\end{aligned}$$

L'interprétation d'une constante est la valeur de cette constante indépendamment de l'environnement. La valeur associée à une variable est celle donnée par l'environnement. Le résultat d'une opération arithmétique est donné en effectuant l'opération avec les résultats des évaluations des sous-expressions.

La sémantique des instructions décrit les transformations des environnements. Nous notons $\llbracket \cdot \rrbracket_{\mathbb{C}}$ cette sémantique définie par :

$$\begin{aligned}
\llbracket x = a \rrbracket_{\mathbb{C}}(\sigma) &= \sigma[x \leftarrow v] \text{ avec } v = \llbracket a \rrbracket_{\mathbb{A}}(\sigma); \\
\llbracket x * r \rrbracket_{\mathbb{C}}(\sigma) &= \begin{cases} \sigma & \text{si } v * r \text{ vrai} \\ \emptyset & \text{si } v * r \text{ faux} \end{cases} \text{ avec } v = \llbracket x \rrbracket_{\mathbb{A}}(\sigma) \text{ et } * \in \{<, >, \leq, \geq\}.
\end{aligned}$$

Une affectation associe une nouvelle valeur, résultat de l'évaluation de l'expression, à la variable concernée. Un test filtre les environnements qui vérifient ou pas les conditions du test. Si le test n'est pas vérifié alors l'environnement vide est le résultat.

La sémantique d'un programme P est donnée par les règles suivantes, en partant de l'état $\langle e, \sigma_0 \rangle$ avec $e \in G[P]$ et σ_0 l'environnement initial :

- soit $\langle c_1, \sigma \rangle$ un état du programme avec $c_1 \in G[P]$ de degré sortant 0 alors le programme a fini son exécution lorsqu'il atteint cet état ;
- soit $\langle c_1, \sigma \rangle$ un état du programme avec $c_1 \in G[P]$ de degré sortant 1, c'est-à-dire $(c_1, i, c_2) \in R[P]$ avec i une affectation, alors $\langle c_2, \langle i \rangle_{\mathbb{C}}(\sigma) \rangle$;
- soit $\langle c_1, \sigma \rangle$ un état du programme avec $c_1 \in G[P]$ de degré sortant 2, c'est-à-dire $(c_1, i_1, c_2), (c_1, i_2, c'_2) \in R[P]$ avec i_1, i_2 un test tel que $i_1 = \neg i_2$, alors si $\langle i_1 \rangle_{\mathbb{C}}(\sigma) \neq \emptyset$ alors $\langle c_2, \sigma \rangle$ sinon $\langle c'_2, \sigma \rangle$.

Sémantique collectrice La sémantique collectrice d'un programme associe à chaque point de contrôle une accumulation de l'ensemble des environnements rencontrés lors de toutes les exécutions à partir d'un ensemble d'environnements initiaux, c'est-à-dire qu'à chaque point de contrôle est associé un invariant. Elle est définie comme la plus petite solution d'un système récursif d'équations sémantiques.

Nous notons $\mathcal{X}_c \subseteq \Sigma$ l'ensemble des environnements au point de contrôle c . La définition de la sémantique collectrice est donnée par :

$$\forall c \in C[P], \mathcal{X}_c = \begin{cases} \mathcal{X}_0 & \text{si } c = e \\ \bigcup_{(c', i, c) \in R[P]} \{ \langle i \rangle_{\mathbb{C}}(\sigma) \mid \sigma \in \mathcal{X}_{c'} \} & \text{sinon} \end{cases},$$

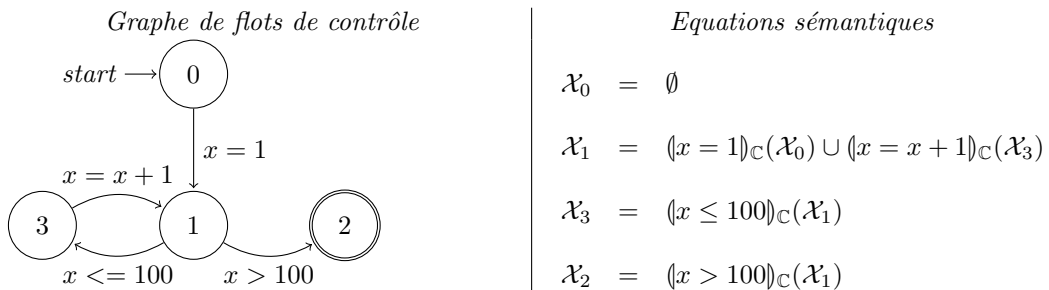
où e est le point d'entrée du graphe de flots de contrôle et $\mathcal{X}_0 \subseteq \Sigma$ est l'ensemble des environnements initiaux.

Nous notons $\wp(E)$ l'ensemble des parties de l'ensemble E . Par le théorème de Tarski (voir théorème 2.1), nous savons que la solution du système d'équations sémantiques existe puisque :

- $D = \langle \wp(\Sigma), \subseteq, \emptyset, \Sigma, \cup, \cap, \rangle$ est un treillis complet avec \emptyset et Σ la borne inférieure et la borne supérieure de D , respectivement ;
- chaque fonction $c \mapsto \bigcup_{(c', i, c) \in R[P]} \{ \langle i \rangle_{\mathbb{C}}(\sigma) \mid \sigma \in \mathcal{X}_{c'} \}$ est croissante au sens de \subseteq .

La résolution du système d'équations sémantiques se fait par itération suivant le théorème de Kleene. Ce théorème stipule que le plus petit point fixe d'une fonction continue \mathbf{f} dans un treillis complet est égale à $\bigcup \{ \mathbf{f}^i(\perp) : i \geq 0 \}$ où \mathbf{f}^i est défini par récurrence par $\mathbf{f}^0 = \lambda x. x$ et $\mathbf{f}^{i+1} = \lambda x. \mathbf{f}(\mathbf{f}^i(x))$. Nous rappelons que toute fonction croissante dans un treillis complet est continue (voir [Win93, section 5.4]).

Exemple 2.10 Nous donnons le système d'équations sémantiques associé au programme de l'exemple 2.9.



Dans la majeure partie des cas, la sémantique collectrice n'est pas calculable. Les valeurs ne sont pas représentables en mémoire (par exemple $\pi, \sqrt{2}, 0.1$) et le nombre d'itérations est potentiellement infini. La théorie de l'interprétation abstraite permet alors de calculer une sur-approximation de la sémantique concrète à partir d'une abstraction de cette dernière.

2.3 Sémantique abstraite des programmes

La théorie de l'interprétation abstraite permet de pallier au problème de la non calculabilité de la sémantique collectrice (concrète). Elle définit les outils théoriques nécessaires pour en évaluer une solution approchée mais sûre, en garantissant que la sémantique abstraite calculable contient tous les comportements du programme décrits par sa sémantique concrète.

La sémantique abstraite associe à chaque point de contrôle du programme un environnement abstrait (c'est-à-dire un ensemble d'environnements concrets). En d'autres termes, cette sémantique permet de calculer les états abstraits du programme qui représentent des ensembles d'états concrets. Elle est définie à partir d'un ensemble de valeurs abstraites et de fonctions sémantiques abstraites. L'ensemble des valeurs abstraites est appelé *domaine abstrait* des valeurs. Ce domaine doit former un treillis complet afin de permettre l'application du théorème du point fixe de Tarski. Un élément de ce domaine représente un ensemble de valeurs concrètes. Un treillis abstrait est défini par :

- $D^\#$ un ensemble de valeurs symboliques ;
- un ordre partiel $\sqsubseteq^\#$ avec un plus petit élément $\perp^\#$ et un plus grand élément $\top^\#$;
- des équivalents des opérateurs d'union $\sqcup^\#$ et d'intersection $\sqcap^\#$.

En nous appuyant sur ce treillis, nous redéfinissons les sémantiques des expressions et des instructions, notée $\llbracket \cdot \rrbracket_C^\#$ dans la suite, ce qui conduit à représenter un programme par un système d'équations sémantiques abstraites. Nous notons par $\Sigma^\# : V[[P]] \rightarrow D^\#$ l'ensemble des environnements abstraits c'est-à-dire qui associent un élément de $D^\#$ aux variables du programme. Avec $\sigma_c^\# \in \Sigma^\#$, l'environnement abstrait au point de contrôle c , nous obtenons :

$$\forall c \in C[[P]], \sigma_c^\# = \begin{cases} \perp^\# & \text{si } c = e \\ \bigsqcup_{(c', i, c) \in R[[P]]}^\# \llbracket i \rrbracket_C^\#(\sigma_{c'}^\#) & \text{sinon} \end{cases},$$

où e est le point d'entrée du graphe de flots de contrôle $G[[P]]$. Comme le domaine abstrait forme un treillis complet, les fonctions sémantiques abstraites sont continues, il est donc possible d'utiliser le théorème de Kleene pour résoudre par itération ce système d'équations sémantiques abstraites [CC79].

La théorie de l'interprétation abstraite permet de s'assurer de la correction de cette abstraction en se basant sur les correspondances de Galois. Les fonctions (α, γ) forment une correspondance de Galois, entre les treillis complets (D, \sqsubseteq) et $(D^\#, \sqsubseteq^\#)$ si :

- $\gamma : D^\# \rightarrow D$ est croissante $x \sqsubseteq^\# y \Rightarrow \gamma(x) \sqsubseteq \gamma(y)$;
- $\alpha : D \rightarrow D^\#$ est croissante $x \sqsubseteq y \Rightarrow \alpha(x) \sqsubseteq^\# \alpha(y)$;
- et si elles vérifient $\alpha(X) \sqsubseteq^\# Y^\# \Leftrightarrow X \sqsubseteq \gamma(Y^\#)$.

α est appelée fonction d'abstraction et γ fonction de concrétisation. En se basant sur cette correspondance, la sûreté de l'analyse est donnée par :

$$\forall c \in C[[P]], \mathcal{X}_c \subseteq \Sigma, \sigma_c^\# \in \Sigma^\#, \alpha(\mathcal{X}_c) \sqsubseteq \sigma_c^\#,$$

$$\bigcup_{\sigma \in \mathcal{X}_c} \left(\bigcup_{(c', i, c) \in R[[P]]} \llbracket i \rrbracket_C(\sigma) \right) \subseteq \gamma \left(\bigsqcup_{(c', i, c) \in R[[P]]}^\# \llbracket i \rrbracket_C^\#(\sigma_{c'}^\#) \right). \quad (2.1)$$

En d'autres termes, pour un ensemble d'environnements \mathcal{X}_c abstrait par l'environnement $\sigma_c^\#$, l'ensemble des comportements du programme au point de contrôle c est inclus dans la concrétisation des comportements abstraits au même point de contrôle. Cette règle vaut pour tous les points de contrôle du programme analysé. La sûreté d'une analyse garantit que les propriétés validées grâce à la sémantique abstraite sont également validées dans la sémantique collectrice.

Le théorème de Kleene donne une méthode de calcul du plus petit point fixe du système d'équations sémantiques abstraites. Mais il se peut que cette solution ne soit pas atteinte en un nombre fini d'itérations, c'est-à-dire que cette méthode n'est pas un algorithme. L'interprétation

abstraite s'appuie alors sur l'opérateur d'élargissement (*widening*) [CC77] pour assurer la terminaison, et donc la calculabilité de la méthode de Kleene. Formellement, cet opérateur, noté ∇ , est défini [CC77] par :

- $v_1, v_2 \in D$ un treillis complet, $v_1 \sqcup v_2 \sqsubseteq v_1 \nabla v_2$;
- Pour toute chaîne croissante $v_0 \sqsubseteq \dots \sqsubseteq v_n \sqsubseteq \dots$ du treillis complet D la chaîne croissante définie par $s_0 = v_0$ et $s_n = s_{n-1} \nabla v_n$ est ultimement stationnaire. Ainsi, il existe n_0 et pour tout n_1 et n_2 tels $n_2 > n_1 > n_0$ alors $s_{n_1} = s_{n_2}$.

L'opérateur ∇ est une sur-approximation de l'opération d'union. Il permet d'extrapoler le résultat de chaque itération de la méthode de Kleene. La suite des itérées converge nécessairement en un nombre fini d'itérations, grâce à l'opérateur ∇ , mais pas forcément sur le plus petit point fixe. En général, la solution des itérées de Kleene utilisant l'opérateur d'élargissement est un post point fixe, c'est-à-dire un point de la forme $f(x) \sqsubseteq x$.

Pour affiner la solution donnée par la méthode de Kleene utilisant l'opérateur d'élargissement, la théorie de l'interprétation abstraite définit l'opérateur de rétrécissement (*narrowing*), noté Δ . Cet opérateur est défini formellement [CC77] par :

- $v_1, v_2 \in D$ un treillis complet, $v_1 \sqsubseteq v_1 \Delta v_2 \sqsubseteq v_2$;
- Pour toute chaîne décroissante $v_0 \supseteq \dots \supseteq v_n \supseteq \dots$ du treillis complet D , la chaîne décroissante définie par $s_0 = v_0$ et $s_n = s_{n-1} \Delta v_n$ est nécessairement stable. Ce qui signifie qu'il existe n_0 et pour tout n_1 et n_2 tels $n_2 > n_1 > n_0$ alors $s_{n_1} = s_{n_2}$.

2.4 Exemple : l'analyse d'intervalles

L'objectif de l'analyse d'intervalles est de calculer le domaine des valeurs des variables d'un programme. Un ensemble de valeurs de la sémantique concrète d'une variable est représenté par un intervalle.

L'ensemble \mathbb{I} des intervalles à bornes entières est $\{[a, b] \mid a \in \mathbb{Z} \wedge b \in \mathbb{Z} \wedge a \leq b\}$ avec $\mathbb{Z} = \mathbb{Z} \cup \{-\infty, +\infty\}$. Un intervalle $[a, b]$ est composé d'une borne inférieure a et d'une borne supérieure b . Une valeur entière r est représentée par l'intervalle $[r, r]$. La sémantique $(\cdot)_{\mathbb{A}}$ des opérations mathématiques $+$, $-$, \times et \div est étendue pour manipuler des intervalles, nous notons $(\cdot)_{\mathbb{I}}$ cette nouvelle sémantique. La figure 2.2 donne la définition de ces opérateurs [Moo79, Rev01]. La somme de deux intervalles est la somme des bornes inférieures et la somme des bornes supérieures. La soustraction est la différence des bornes opposées. La multiplication nécessite de calculer toutes les combinaisons possibles des bornes pour extraire la plus petite et la plus grande valeur comme bornes de l'intervalle résultat. La division impose que le dénominateur ne contienne pas zéro pour être effectuée.

$$\begin{aligned}
 (x_1)_{\mathbb{I}} &= [a_1, b_1] \text{ et } (x_2)_{\mathbb{I}} = [a_2, b_2] \\
 (x_1 + x_2)_{\mathbb{I}} &= [a_1 + a_2, b_1 + b_2] \\
 (x_1 - x_2)_{\mathbb{I}} &= [a_1 - b_2, b_1 - a_2] \\
 (x_1 \times x_2)_{\mathbb{I}} &= [\min(a_1 a_2, a_1 b_2, b_1 a_2, b_1 b_2), \max(a_1 a_2, a_1 b_2, b_1 a_2, b_1 b_2)] \\
 \left(\frac{1}{x_1}\right)_{\mathbb{I}} &= \left[\frac{1}{b_1}, \frac{1}{a_1}\right] \text{ avec } 0 \notin [a_1, b_1]
 \end{aligned}$$

FIG. 2.2 – Définition de l'arithmétique d'intervalles d'entiers.

L'ensemble des valeurs intervalles devient un ensemble partiellement ordonné, en ajoutant la relation d'inclusion $\sqsubseteq_{\mathbb{I}}$ définie par :

$$[a_1, b_1] \sqsubseteq_{\mathbb{I}} [a_2, b_2] \Leftrightarrow a_1 \geq a_2 \wedge b_1 \leq b_2$$

et les opérations d'union $\sqcup_{\mathbb{I}}$ et d'intersection $\sqcap_{\mathbb{I}}$ sont définies par :

$$[a_1, b_1] \sqcup_{\mathbb{I}} [a_2, b_2] = [\min(a_1, a_2), \max(b_1, b_2)]$$

$$[a_1, b_1] \sqcap_{\mathbb{I}} [a_2, b_2] = [\max(a_1, a_2), \min(b_1, b_2)]$$

[CC77] démontre que $\langle \mathbb{I}, \sqsubseteq_{\mathbb{I}}, \emptyset, [-\infty, +\infty], \sqcup_{\mathbb{I}}, \sqcap_{\mathbb{I}} \rangle$ forme un treillis complet ainsi que l'existence de la correspondance de Galois suivante :

$$\langle \wp(\mathbb{Z}), \subseteq \rangle \xrightleftharpoons[\alpha_{\mathbb{I}}]{\gamma_{\mathbb{I}}} \langle \mathbb{I}, \sqsubseteq_{\mathbb{I}} \rangle$$

avec

$$\alpha_{\mathbb{I}}(R) = [\min(R), \max(R)] \text{ et } \gamma_{\mathbb{I}}([a, b]) = \{x \mid a \leq x \leq b\}$$

De plus, [CC77] définit également l'opérateur d'élargissement, $\nabla_{\mathbb{I}}$:

$$[a_1, b_1], [a_2, b_2] \in \mathbb{I}, [a_1, b_1] \nabla_{\mathbb{I}} [a_2, b_2] = \left[\begin{cases} a_1 & \text{si } a_1 \leq a_2 \\ -\infty & \text{sinon} \end{cases}, \begin{cases} b_1 & \text{si } b_1 \geq b_2 \\ +\infty & \text{sinon} \end{cases} \right] \quad (2.2)$$

ainsi que l'opérateur de rétrécissement, $\Delta_{\mathbb{I}}$:

$$[a_1, b_1], [a_2, b_2] \in \mathbb{I}, [a_1, b_1] \Delta_{\mathbb{I}} [a_2, b_2] = \left[\begin{cases} a_2 & \text{si } a_1 = -\infty \\ \min(a_1, a_2) & \text{sinon} \end{cases}, \begin{cases} b_2 & \text{si } b_1 = +\infty \\ \max(b_1, b_2) & \text{sinon} \end{cases} \right]$$

L'opérateur d'élargissement $\nabla_{\mathbb{I}}$ a pour objectif d'accélérer la convergence de calcul du point fixe. Cette accélération est fondée sur une extrapolation des bornes des intervalles. L'opérateur de rétrécissement $\Delta_{\mathbb{I}}$ a pour objectif d'affiner uniquement le bornes infinies.

La sûreté de l'analyse statique par interprétation abstraite utilisant le treillis des intervalles a aussi été démontrée dans [CC77].

Exemple 2.11 Les résultats de l'analyse d'intervalles sur le programme de l'exemple 2.9 en appliquant les opérateurs d'élargissement $\nabla_{\mathbb{I}}$ et de rétrécissement $\Delta_{\mathbb{I}}$ sont :

Equations sémantiques abstraites	Résultats
$\sigma_0^{\#} = \perp^{\#}$	$\sigma_0^{\#} = \{x \leftarrow \perp\}$
$\sigma_1^{\#} = \langle x = 1 \rangle_{\mathbb{C}}(\sigma_0^{\#}) \cup \langle x = x + 1 \rangle_{\mathbb{C}}(\sigma_3^{\#})$	$\sigma_1^{\#} = \{x \leftarrow [1, 101]\}$
$\sigma_3^{\#} = \langle x \leq 100 \rangle_{\mathbb{C}}(\sigma_1^{\#})$	$\sigma_3^{\#} = \{x \leftarrow [1, 100]\}$
$\sigma_2^{\#} = \langle i > 100 \rangle_{\mathbb{C}}(\sigma_1^{\#})$	$\sigma_2^{\#} = \{x \leftarrow [101, 101]\}$
Remarque : $\sigma_2^{\#}$ est obtenu par une application de l'opérateur de rétrécissement $\Delta_{\mathbb{I}}$	

Il existe d'autres domaines numériques abstraits comme le domaine des octogones[Min04, Min06] ou encore le domaine des polyèdres convexes[CH78]. Chacun de ces domaines calcule une propriété numérique différente. Pour les intervalles, la propriété est :

$$\forall x \in V[P], \quad c_1 \leq x \leq c_2$$

pour les octogones, elle est :

$$\forall x_1, x_2 \in V[P], \quad \pm x_1 \pm x_2 \leq c$$

et pour les polyèdres convexes, nous avons :

$$\forall x_i \in V[P], \quad \sum_{i=1}^n c_i x_i \leq c$$

où $V[[P]]$ représente l'ensemble des variables d'un programme P . Tous ces domaines permettent de calculer automatiquement des bornes numériques c et c_i sur les variables du programme. Mais dans le cas du domaine des octogones et des polyèdres, le calcul s'appuie sur des relations entre les variables pour obtenir ces bornes. Cela apporte alors une plus grande quantité d'informations sur les programmes étudiés que le domaine des intervalles. Bien que ce dernier soit moins expressif, il s'adapte plus aisément aux algorithmes numériques, objets d'étude au centre de cette thèse, ce qui n'est pas forcément le cas des deux autres, en particulier en présence de nombres à virgule flottante (voir section 3.1).

2.5 Validation des programmes à flots de données synchrones

Les méthodes d'analyse statique par interprétation abstraite ont été appliquées sur les langages à flots de données synchrones. La particularité de ces langages est qu'ils ont des traits communs avec le langage Matlab/Simulink, en particulier la sémantique synchrone qui manipule des séquences de valeurs. Cependant, ils ont une expressivité moindre puisqu'ils permettent uniquement la modélisation des systèmes à temps discret.

Les techniques de validation des programmes, écrits dans les langages à flots de données synchrones, permettent de mieux apprécier les forces et les manques de l'application des méthodes formelles au niveau des spécifications. Nous considérons les trois principaux langages à flots de données synchrones : Lustre [CPHP87], Signal [GBGM91] et Lucid synchrone [CP96]. L'analyse statique par interprétation abstraite a été appliquée sur chacun de ces langages. Nous donnons dans cette section les exemples de ces applications en soulignant les propriétés qui sont ainsi vérifiées.

2.5.1 Sémantique synchrone

Un système est dit réactif s'il agit continûment avec son environnement à l'opposé d'un système transformationnel (par exemple un compilateur). Les langages synchrones ont été conçus spécifiquement pour réaliser des systèmes réactifs. Ces langages se fondent sur un temps logique qui correspond à la suite de réactions du système. L'hypothèse principale est de considérer les calculs et les transferts d'informations comme étant instantanés. Les compilateurs de ces langages garantissent que les traitements se font en un temps borné mais non défini. Une étude du pire temps d'exécution (*Worst-case execution time (WCET)*), des implémentations des programmes synchrones, est réalisée *a posteriori* afin de déterminer le type de matériel vérifiant les exigences du temps-réel. En d'autres termes, les langages synchrones supposent que le programme s'exécute infiniment vite afin de simplifier la conception. Et c'est le choix du matériel sur lequel s'exécutent ces programmes qui permet de garantir que le système s'exécute suffisamment rapidement pour n'oublier aucune interaction avec l'environnement.

Nous présentons brièvement dans cette section le langage Lustre [CPHP87]. Lustre est un langage synchrone à flots de données¹ qui est à l'origine de l'outil industriel SCADE². Un programme Lustre est composé d'une suite d'équations. La figure 2.3 est un exemple de programme (node). Il modélise un thermostat devant maintenir la température, par exemple d'une pièce, entre une valeur maximale T_{\max} et une valeur minimale T_{\min} . Le contrôleur, en fonction de la température, active ou désactive un mécanisme de chauffage grâce à la variable *chauffe*.

Une variable en Lustre est appelée flot, c'est-à-dire qu'elle représente une fonction du temps (logique) \mathbb{N} dans un domaine de valeur V . Le temps est logique puisqu'il représente uniquement les instants des réactions. Dans l'exemple, nous avons deux domaines de valeurs *real* représentant les valeurs réelles et *bool* représentant les valeurs booléennes. Une équation exprime la définition d'un flot par la composition instant par instant de plusieurs flots. Par exemple $x = (y + z)/2$ se traduit par $\forall k \in \mathbb{N}, x_k = (y_k + z_k)/2$ ce qui signifie que la valeur du flot x est égale à la moyenne des flots y et z pour tous les instants.

¹Historiquement, le premier langage synchrone fut Esterel[BG92] qui était un langage impératif.

²<http://www.esterel-technologies.com/products/scade-suite/>

```

node control (temperature: real) returns (chauffe : bool)
var etat : bool
let
  etat = if (temperature >= Tmax)
    then false
    else if (temperature <= Tmin)
    then true
    else pre (etat);
  chauffe = false -> etat;
tel

```

FIG. 2.3 – Régulateur de température en Lustre.

Les expressions sont composées par des opérations arithmétiques, des opérations de comparaisons et des expressions conditionnelles (if). Elles sont étendues à la manipulation des séquences [Kah74]. Plus précisément, une opération o est étendue aux séquences par application de o élément par élément. Par exemple, soient x et y deux flots alors le flot $z = x + y = \forall k \in \mathbb{N}, z_k = x_k + y_k$. A un flot est associée une horloge, c'est-à-dire un flot booléen qui signifie l'absence (false) ou la présence (true) d'une valeur du signal. Des signaux sont composables s'ils ont la même horloge. En plus de ces opérations, il existe quatre opérateurs temporels :

- **pre**, un décalage temporel associé à une mémorisation de valeurs. Si $x = x_0, x_1, \dots$ est un flot,

$$\text{pre}(x) = x_{-1}, x_0, x_1, \dots$$

est un flot dont la valeur au premier instant x_{-1} n'est pas définie ;

- \rightarrow est l'opérateur d'initialisation, si x et y sont des flots,

$$x \rightarrow y = x_0, y_1, y_2, \dots$$

est le flot qui a la valeur de x au premier instant et, pour tous les autres instants, il prend les valeur de y ;

- **when** est une opération d'échantillonnage. Par exemple, soit un flot $x = x_0, x_1, x_2, \dots$ et un flot booléen $b = \text{true}, \text{false}, \text{true}, \text{false}, \dots$,

$$y = x \text{ when } b = x_0, x_2, \dots$$

où y possède une horloge deux fois plus lente que x ;

- **current** est une opération d'interpolation, agissant sur un flot échantillonné. Par exemple, soit y le flot précédent défini avec l'opérateur **when**,

$$z = \text{current}(y) = x_0, x_0, x_1, x_1, \dots$$

A chaque instant où y est défini z a la valeur de y et, aux instants où y n'est pas défini z garde la valeur précédente.

2.5.2 Analyse statique de programmes synchrones

L'une des difficultés dans les langages synchrones à flots de données est de garantir que deux flots sont effectivement composables ou synchronisables. Cette vérification se traduit alors par un calcul d'horloge qui valide la synchronisation des flots. Une analyse statique par interprétation abstraite de ce calcul d'horloge [Jen95] a été réalisée pour un langage synchrone proche de Lustre. Le langage Signal se distingue de Lustre par une représentation beaucoup plus sophistiquée des horloges. Une analyse statique de calcul d'horloge peut être trouvée dans [GGB08]. Une telle analyse existe également pour Lucid synchrone [CP96] basée sur un système de types, le typage étant vu comme une abstraction. Lucid synchrone est un langage synchrone à flots de données fondé sur un langage fonctionnel à la ML.

Afin de garantir l'exécution en un temps borné, il est nécessaire de détecter les boucles de causalité. Il existe une boucle de causalité si une variable à un instant k est définie à partir de sa valeur au même instant. Par exemple :

$$x = x + 1$$

qui se traduit par :

$$\forall k \in \mathbb{N}, x_k = x_k + 1$$

Ce genre d'équation nécessite la résolution d'une équation algébrique qui n'est pas toujours possible et qui, de toutes les façons, ne permet pas de garantir un temps d'exécution borné. La stratégie habituelle pour résoudre ce problème est de calculer le graphe de dépendance entre les variables de programme et de vérifier qu'une opération `pre` est sur le chemin. Dans le cas du langage Lucid Synchrone, une analyse des boucles de causalité est faite à la compilation grâce à un système de types [CP01]. Nous voyons encore une fois le `typepage` comme une analyse statique par interprétation abstraite.

La validation de propriétés de sûreté a été réalisée sur le langage Lustre par [JHR99, Jea00] et sur le langage Signal par [BJT99]. Ces deux analyses s'appuient sur le domaine des polyèdres convexes pour valider les parties numériques. Cependant, les méthodes diffèrent sur l'objet de l'analyse, dans [JHR99, Jea00] l'analyse est effectuée sur la représentation en automate des programmes synchrones tandis que [BJT99] applique son analyse sur le système d'équations. Dans les deux cas, les propriétés numériques sont restreintes aux variables entières.

Dernièrement, Bertrane [Ber05, Ber06] s'est intéressé à l'étude de la synchronisation des programmes synchrones en considérant des horloges imparfaites c'est-à-dire désynchronisées. Les propriétés ainsi vérifiées permettent de confronter la spécification mathématique idéale à l'implémentation imparfaite.

Une présentation plus détaillée des méthodes de vérification des langages synchrones peut être trouvée dans [HR99] et, dans le cas particulier de l'interprétation abstraite, dans [Hal94]. De plus, pour des informations complémentaires sur les langages synchrones, le lecteur pourra se référer à [Hal93, BCE⁺03].

Outils d'étude des programmes numériques

Le remplacement des systèmes mécaniques par des systèmes électromécaniques dans les systèmes de contrôle introduit les traitements numériques dans les logiciels embarqués. L'utilisation d'une arithmétique en représentation finie engendre nécessairement des erreurs de calcul [Gol91, Mon07]. Une part importante des logiciels embarqués critiques s'appuie sur des algorithmes numériques pour prendre des décisions. Il est alors important d'étudier les comportements numériques des programmes afin de s'assurer que les erreurs de calcul n'ont pas "trop" d'influence.

Le problème de la validation de ces programmes numériques est que les méthodes de tests sont très difficilement applicables. Les instabilités numériques pouvant apparaître pour des valeurs particulières qui ne sont pas couvertes par les méthodes de tests (par exemple le test aux limites des domaines des valeurs des variables). Les méthodes d'analyse statique par interprétation abstraite offrent un cadre propice à l'étude des propriétés numériques des programmes.

Ce chapitre est organisé comme suit. Une introduction à la problématique de la précision numérique est donnée à la section 3.1. Une présentation des méthodes issues du domaine de l'analyse numérique est exposée à la section 3.2. Dans ce même chapitre, nous introduirons le domaine des nombres flottants avec erreurs qui permet de faire une analyse statique de programmes numériques. Nous présenterons les principaux algorithmes d'intégration numériques ainsi que l'intégration numérique garantie à la section 3.3.

3.1 Problématique de la précision numérique

La norme IEEE 754 [IEE85] a été définie en 1985 pour pallier les problèmes de représentation des nombres réels en machine. Nous présentons les principaux aspects de cette norme dans cette section. Nous supposons dans cette thèse que l'arithmétique utilisée en machine est conforme à la norme IEEE 754. Celle-ci définit l'arithmétique à virgule flottante en base 2 qui est mise en œuvre dans la majorité des processeurs actuels. Elle caractérise complètement la représentation mémoire des éléments de cette arithmétique, appelés nombres à virgule flottante, nombres flottants ou flottants. Un flottant est composé en mémoire (au niveau du bit) de trois parties : un signe s valant 1 ou -1 , un exposant e compris entre e_{min} et e_{max} et une mantisse m . Le réel r associé à un nombre flottant est donné par la relation $r = s.m.2^e$.

La norme propose plusieurs types de flottants dont les plus importants sont le type simple précision et le type double précision. La précision p d'un nombre flottant correspond au nombre de bits composant sa mantisse. De plus, pour une précision donnée, les valeurs de e_{min} et e_{max} sont fixées. Par exemple, en double précision la mantisse est codée sur 53 bits, $e_{max} = 1023$ et $e_{min} = -e_{max} - 1$ ce qui correspond à un codage sur 11 bits de l'exposant.

Les opérations $+$, $-$, \times , \div et $\sqrt{}$ ne sont pas définies en fonction des caractéristiques techniques de la machine mais suivant des contraintes mathématiques. La propriété partagée par ces opérations est celle de l'arrondi exact, c'est-à-dire que le résultat r d'une opération flottante o est équivalent au résultat r' obtenu par le calcul de o en précision infinie puis arrondi. La norme définit plusieurs modes d'arrondi dont les principaux sont : vers $+\infty$ qui donne un flottant plus grand que le réel, vers $-\infty$ qui donne un flottant plus petit que le réel et au plus près \sim qui prend le flottant le plus proche du réel.

Une information importante sur les flottants est l'*ulp* (*Unit in the Last Place*) dont la valeur donne l'ordre de grandeur du dernier bit de la mantisse. L'*ulp* permet d'estimer la distance séparant deux nombres flottants, c'est-à-dire de majorer l'erreur d'arrondi. Pour un flottant f , l'*ulp* est défini par :

$$ulp(f) = 2^{-p+e-1}$$

avec p la précision du flottant et e son exposant. La majoration de l'erreur dépend du mode d'arrondi : pour les arrondis vers les infinis, l'erreur d'arrondi est majorée par $ulp(f)$ tandis qu'avec le mode au plus près la majoration est de $\frac{1}{2}ulp(f)$.

Exemple 3.1 Nous présentons un exemple de calcul en arithmétique flottante qui montre la difficulté d'interpréter des résultats obtenus en machine. Le programme calcule une approximation de la dérivée de la fonction $\mathbf{f}(x) = 2x$ en $x = 1$. La valeur théorique de la dérivée d est égale à 2. Nous réalisons un calcul approché en machine avec des flottants en simple précision suivant la formule :

$$\mathbf{f}'(x) \approx \frac{\mathbf{f}(x+h) - \mathbf{f}(x)}{h}$$

avec des valeurs de h petites.

Quand $h = 1e^{-8}$, il y a un phénomène d'absorption qui se produit au niveau de l'expression $\mathbf{f}(x+h)$, c'est-à-dire que le calcul $1+1e^{-8}$ donne comme résultat 1. Nous obtenons alors $\mathbf{f}(x+h) = \mathbf{f}(x)$ ce qui conduit au résultat 0.0. Si $h = 1e^{-7}$, le résultat de $1 + 1e^{-7}$ est très proche de 1 ce qui engendre un autre problème : l'élimination catastrophique. Les valeurs $1 + 1e^{-7}$ et 1 étant très proches, quand nous les soustrayons, une erreur importante survient, qui est amplifiée par la division, d'où le résultat. Avec $h = 1e^{-6}$, les mêmes problèmes sont présents mais moins soulignés, tandis qu'avec $h = 1e^{-5}$ et $1e^{-4}$ les erreurs introduites par les calculs flottants sont encore moins importantes et donc elles influencent très peu le résultat.

```
float f (float x) {
    return 2 * x;
}
```

Résultats

```
float derivation (float h) {
    float x = 1;
    float d = 0.0;

    d = (f(x+h) - f(x)) / h;

    return d;
}
```

Théorique : $d = 2$

Machine : $d = 0.0$ ($h = 1e^{-8}$)

$d = 2.384186$ ($h = 1e^{-7}$)

$d = 1.907349$ ($h = 1e^{-6}$)

$d = 2.002716$ ($h = 1e^{-5}$)

$d = 2.000332$ ($h = 1e^{-4}$)

Nous introduisons deux fonctions qui seront utilisées dans la suite de ce chapitre : $\uparrow_{\circ} : \mathbb{R} \rightarrow \mathbb{F}$ et $\downarrow_{\circ} : \mathbb{R} \rightarrow \mathbb{R}$. \mathbb{R} représente l'ensemble des réels et \mathbb{F} est l'ensemble des nombres flottants. La fonction \uparrow_{\circ} associe à un réel r le nombre flottant f correspondant, suivant le mode d'arrondi \circ choisi. Cette fonction définit la partie représentable de r dans les flottants. La fonction \downarrow_{\circ} calcule la partie non représentable de r dans les flottants et elle est définie par :

$$\downarrow_{\circ}(r) = r - \uparrow_{\circ}(r)$$

Nous définissons, à la figure 3.1, la fonction \mathcal{F}_\circ qui décrit le calcul en arithmétique flottante suivant le mode d'arrondi \circ choisi. Elle suit l'arrondi exact c'est-à-dire chaque opération est réalisée dans les réels avant d'être arrondie. Cette fonction sera utilisée à la section 3.2 pour définir les autres arithmétiques utilisées dans l'étude de la précision numérique.

$$\begin{aligned}
 a &= f_a \text{ et } b = f_b \\
 \mathcal{F}_\circ(a + b) &= \uparrow_\circ (f_a + f_b) \\
 \mathcal{F}_\circ(a - b) &= \uparrow_\circ (f_a - f_b) \\
 \mathcal{F}_\circ(a \times b) &= \uparrow_\circ (f_a \times f_b) \\
 \mathcal{F}_\circ\left(\frac{a}{b}\right) &= \uparrow_\circ \left(\frac{f_a}{f_b}\right)
 \end{aligned}$$

FIG. 3.1 – Définition de la fonction \mathcal{F}_\circ de calculs dans les flottants.

3.2 Méthodes pour l'étude de la précision numérique

L'analyse numérique est une discipline mathématique qui a pour objectif de résoudre des problèmes d'analyse mathématique par des méthodes numériques c'est-à-dire par des programmes informatiques. Une des principales difficultés en analyse numérique est la représentation des nombres réels. De nombreuses méthodes ont été mises au point pour répondre à ce problème tel que l'arithmétique d'intervalles présentée à la section 3.2.1 ou l'arithmétique stochastique exposée à la section 3.2.2. Une autre difficulté en analyse numérique est liée à la précision numérique des calculs faisant parfois diverger le résultat machine du résultat réel. La méthode de la différentiation automatique, section 3.2.3, permet d'obtenir des approximations du résultat réel en fonction des résultats numériques ainsi de limiter cette divergence. A la section 3.2.4, nous introduisons le domaine numérique des nombres flottants avec erreurs.

Dans la suite de cette section, pour chaque arithmétique présentée, nous définirons la sémantique associée au langage des expressions arithmétiques donné à la figure 3.2. Nous considérons des expressions formées de constantes réelles r , de variables x et des opérations arithmétiques $+$, $-$, \times et \div .

$$a ::= r \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 \div a_2$$

FIG. 3.2 – Grammaire des expressions arithmétiques.

3.2.1 Arithmétique d'intervalles

Une des solutions proposées pour le problème de la représentation des nombres réels est l'arithmétique d'intervalles [Moo79, BBC⁺97, Rev01, JKDW01]. Cette arithmétique permet de représenter en machine des nombres réels en les encadrant par des nombres machines. Par exemple, le réel $\frac{1}{3}$ peut être approché par l'intervalle $[0.33333, 0.33334]$. La différence par rapport à l'arithmétique d'intervalles à bornes entières est la prise en compte des arrondis des calculs. En effet, les bornes des intervalles à bornes réelles sont représentées en machine par des nombres flottants.

Comme nous avons déjà présenté l'arithmétique d'intervalles à la section 2.4, nous mettons en évidence, dans cette section, les modifications introduites par l'utilisation de nombres flottants pour les bornes des intervalles. La figure 3.3 donne la définition des opérateurs $+$, $-$, \times , \div [Moo79,

Rev01]. Nous notons \mathcal{I} la fonction qui effectue une opération dans l'arithmétique d'intervalles. Les règles de calculs sont les mêmes que pour les intervalles à bornes entières sauf que les calculs sur les bornes doivent être arrondis vers l'extérieur (vers $-\infty$ pour la borne inférieure et $+\infty$ pour la borne supérieure).

$$\begin{aligned}
 & x_1 = [a_1, b_1] \text{ et } x_2 = [a_2, b_2] \\
 & \mathcal{I}(x_1 + x_2) = [\mathcal{F}_{-\infty}(a_1 + a_2), \mathcal{F}_{+\infty}(b_1 + b_2)] \\
 & \mathcal{I}(x_1 - x_2) = [\mathcal{F}_{-\infty}(a_1 - b_2), \mathcal{F}_{+\infty}(b_1 - a_2)] \\
 & \mathcal{I}(x_1 \times x_2) = [\min(\mathcal{F}_{-\infty}(a_1 a_2), \mathcal{F}_{-\infty}(a_1 b_2), \mathcal{F}_{-\infty}(b_1 a_2), \mathcal{F}_{-\infty}(b_1 b_2)), \\
 & \quad \max(\mathcal{F}_{+\infty}(a_1 a_2), \mathcal{F}_{+\infty}(a_1 b_2), \mathcal{F}_{+\infty}(b_1 a_2), \mathcal{F}_{+\infty}(b_1 b_2))] \\
 & \mathcal{I}\left(\frac{x_1}{x_2}\right) = \left[\mathcal{F}_{-\infty}\left(\frac{a_1}{b_2}\right), \mathcal{F}_{+\infty}\left(\frac{b_1}{a_2}\right)\right] \text{ avec } 0 \notin [a_2, b_2]
 \end{aligned}$$

FIG. 3.3 – Définition de l'arithmétique d'intervalles flottants.

La sémantique $\llbracket \cdot \rrbracket_{\mathbb{I}}$ des expressions arithmétiques utilisant l'arithmétique d'intervalles flottants est donnée à la figure 3.4. $\llbracket \cdot \rrbracket_{\mathbb{I}}$ est défini par induction sur la structure d'une expression arithmétique a . ρ représente l'environnement qui à chaque variable associe une valeur intervalle.

$$\llbracket a \rrbracket_{\mathbb{I}}(\rho) = \begin{cases} [c, c] & \text{si } a \text{ est une constante } c \\ \rho(x) & \text{si } a \text{ est une variable } x \\ \mathcal{I}(\llbracket a_1 \rrbracket_{\mathbb{I}}(\rho) \diamond \llbracket a_2 \rrbracket_{\mathbb{I}}(\rho)) & \text{sinon avec } \diamond \in \{+, -, \times, \div\} \end{cases}$$

FIG. 3.4 – Sémantique des expressions arithmétiques basée sur l'arithmétique d'intervalles

Il faut remarquer que la relation d'ordre définie dans [Moo79] n'est pas la relation par inclusion. Elle est définie par :

$$[a_1, b_1] \leq [a_2, b_2] \stackrel{\text{def}}{=} b_1 \leq a_2$$

Nous introduisons dans ce paragraphe les notations et fonctions liées à l'arithmétique d'intervalles que nous utiliserons dans la suite de cette thèse. Les valeurs entre crochets $[]$ désigneront des valeurs intervalles. Par exemple, $[x]$ représente un intervalle sans expliciter ses bornes ou $[a, b]$ est un intervalle dont a est la borne inférieure et b la borne supérieure. Le centre d'un intervalle est donné par la fonction \mathbf{m} définie par l'équation (3.1).

$$\mathbf{m}([a, b]) = \frac{a + b}{2} \quad (3.1)$$

3.2.2 Arithmétique stochastique

L'arithmétique flottante introduit des résultats imprécis à cause du nombre fini de bits pour représenter les valeurs réelles (par exemple π). Pour retrouver les résultats réels des calculs effectués en machine, J. Vignes a développé la méthode CESTAC (Contrôle et Estimation Stochastique des Arrondis de Calculs) [Vig96] qui par la suite a donné naissance à l'arithmétique stochastique. Cette méthode se base sur l'échantillonnage statistique, en exécutant plusieurs fois le même programme en propageant les erreurs d'arrondi différemment de façon à obtenir des résultats différents. L'idée intuitive est que certaines erreurs d'arrondi se compensent pendant les calculs. La partie commune aux différents résultats va représenter la partie fiable du résultat et le reste la partie non significative.

Pour permettre une propagation différente des erreurs d'arrondi pour les différentes exécutions du programme, la méthode CESTAC prévoit de modifier aléatoirement le mode d'arrondi des

calculs. Lors de chaque calcul, avec une probabilité de $1/2$, soit le mode d'arrondi vers $-\infty$, soit le mode d'arrondi vers $+\infty$ est choisi.

Le calcul du résultat d'un programme avec l'arithmétique stochastique exige que chaque exécution suive le même flot de contrôle. Avec N exécutions séquentielles, en choisissant aléatoirement les modes d'arrondi, il peut se trouver que les tests sur des valeurs flottantes conduisent à prendre des branches différentes du programme et ainsi faussent complètement les résultats. Pour pallier ce problème, les N exécutions vont se faire conjointement, c'est-à-dire qu'une instruction d'un programme va être exécutée N fois avant de passer à la suivante. Ce mode d'exécution est appelé *exécution synchrone*. Cette technique a été proposée par J.M. Chesneaux [Che95] et, elle est mise en œuvre dans le logiciel CADNA permettant l'utilisation des nombres stochastiques dans les programmes C ou Fortran.

Avec une exécution synchrone, les opérateurs de comparaison doivent être redéfinis, pour que le choix des chemins à prendre dans le graphe de contrôle soit représentatif d'une exécution simple. La méthode CESTAC définit donc les opérateurs de comparaison pour des nombres stochastiques. Ces définitions se basent sur la notion du *zéro stochastique*. Le zéro stochastique permet de prendre en compte le cas où l'ensemble des N résultats n'a aucun chiffre significatif. C'est-à-dire que, les N résultats sont représentés par N valeurs différentes. Le zéro stochastique est défini de la manière suivante :

Définition 3.1 Un nombre stochastique est un zéro informatique s'il est nul ou s'il n'a aucun chiffre significatif. Il est noté $\underline{0}$.

La représentation des nombres stochastiques peut se faire de deux façons différentes, soit par un vecteur de taille N représentant les valeurs des N exécutions (*représentation discrète*), soit par la moyenne m et l'écart-type σ de ces N valeurs (*représentation continue*). La définition des règles de calcul (figure 3.5) se base sur la représentation continue des nombres stochastiques. Nous notons \mathcal{S} la fonction qui effectue les opérations stochastiques.

$$\begin{aligned}
 & x_1 = (m_1, \sigma_1) \text{ et } x_2 = (m_2, \sigma_2) \\
 \\
 \mathcal{S}(x_1 + x_2) &= (m_1 + m_2, \sigma_1 + \sigma_2) \\
 \mathcal{S}(x_1 - x_2) &= (m_1 - m_2, \sigma_1 + \sigma_2) \\
 \mathcal{S}(x_1 \times x_2) &= (m_1 \times m_2, m_2^2 \cdot \sigma_1^2 + m_1^2 \cdot \sigma_2^2) \\
 \mathcal{S}(x_1 \div x_2) &= \left(m_1 \div m_2, \left(\frac{\sigma_1}{m_2} \right)^2 + \left(\frac{m_1 \cdot \sigma_2}{m_2^2} \right)^2 \right) \text{ avec } m_2 \neq 0
 \end{aligned}$$

FIG. 3.5 – Règles de calcul de l'arithmétique stochastique.

Il est possible d'associer une précision à chaque nombre stochastique. Si $X = (m, \sigma)$ alors il existe λ_β (dépendant uniquement de β) tel que :

$$P(X \in [m - \lambda_\beta \cdot \sigma, m + \lambda_\beta \cdot \sigma]) = \beta$$

$I_{\beta, X} = [m - \lambda_\beta \cdot \sigma, m + \lambda_\beta \cdot \sigma]$ est l'intervalle de confiance de m à β . Où β représente la probabilité pour laquelle la valeur réelle est incluse dans l'intervalle de confiance. Pour $\beta = 0.95$, $\lambda_\beta \approx 1.96$. Nous pouvons alors donner les définitions des opérateurs de comparaison.

Définition 3.2 Soient x_1 et x_2 deux nombres stochastiques, x_1 est stochastiquement égal à x_2 , noté $x_1 =_{\mathcal{S}} x_2$ si :

$$\mathcal{S}(x_1 - x_2) = \underline{0}$$

Définition 3.3 Soient $x_1 = (m_1, \sigma_1)$ et $x_2 = (m_2, \sigma_2)$ deux nombres stochastiques, x_1 est stochastiquement strictement supérieur à x_2 , noté $x_1 >_S x_2$ si :

$$m_1 - m_2 > \lambda_\beta \sqrt{\sigma_1^2 + \sigma_2^2}$$

Définition 3.4 Soient $x_1 = (m_1, \sigma_1)$ et $x_2 = (m_2, \sigma_2)$ deux nombres stochastiques, x_1 est stochastiquement supérieur ou égal à x_2 , noté $x_1 \geq_S x_2$ si :

$$m_1 \geq m_2 \text{ ou } x_1 =_S x_2$$

La sémantique $\llbracket \cdot \rrbracket_S$ des expressions arithmétiques utilisant l'arithmétique stochastique est donnée à la figure 3.6. $\llbracket \cdot \rrbracket_S$ est défini par induction sur la structure d'une expression arithmétique a . Un environnement ρ associe à chaque variable un nombre stochastique. La sémantique $\llbracket \cdot \rrbracket_S$ associe à une constante réelle c un nombre stochastique en calculant la moyenne m_c à partir de trois arrondis aléatoires de c et en calculant l'écart-type à partir de m_c . Des résultats pratiques ont conduit à estimer que trois arrondis aléatoires étaient suffisants pour satisfaire les hypothèses du calcul de l'intervalle de confiance.

$$\llbracket a \rrbracket_S(\rho) = \begin{cases} (m_c, \max(d_1, d_2, d_3)) & \text{si } a \text{ est une constante } c \\ & \text{avec } m_c = \frac{c_1 + c_2 + c_3}{3}, c_i = \uparrow? (c), \text{ et } d_i = |m_c - c_i| \\ \rho(x) & \text{si } a \text{ est une variable } x \\ \mathcal{S}(\llbracket a_1 \rrbracket_S(\rho) \diamond \llbracket a_2 \rrbracket_S(\rho)) & \text{sinon avec } \diamond \in \{+, -, \times, \div\} \end{cases}$$

FIG. 3.6 – Sémantique des expressions arithmétiques basée sur l'arithmétique stochastique

3.2.3 Différentiation automatique

L'idée de la différentiation automatique [Ral81, Gri00, BHN02] est de considérer un programme comme une fonction mathématique définie par composition d'opérations élémentaires. En nous restreignant aux expressions arithmétiques, les opérations élémentaires sont $+$, $-$, \times et \div , et les fonctions étudiées sont toutes les compositions possibles de ces opérations. Cette méthode s'appuie sur la règle de dérivation en chaîne : si \mathbf{f} et \mathbf{g} sont deux fonctions différentiables alors $\mathbf{f} \circ \mathbf{g}$ est différentiable telle que :

$$(\mathbf{f} \circ \mathbf{g})' = \mathbf{f}'(\mathbf{f} \circ \mathbf{g}')$$

Cette règle donne la façon d'évaluer la dérivée de fonctions composées, par application des règles mathématiques usuelles de dérivation. La différentiation est réalisée suivant les variables du programme et nous distinguons deux types de variables : les *variables indépendantes* et les *variables dépendantes*. Les variables indépendantes sont, la plupart du temps, les variables d'entrée du programme et elles sont utilisées pour la différentiation. Les variables dépendantes sont celles pour lesquelles nous voulons calculer les dérivées et elles sont, en général, les sorties du programme. De plus, les variables sont dites *actives* si elles sont accompagnées d'une information de dérivée.

Nous notons ρ l'environnement qui associe à chaque variable un flottant avec sa dérivée. Nous définissons la sémantique $\llbracket \cdot \rrbracket_D$ associée à cette méthode suivant la structure d'une expression arithmétique a telle que :

$$\llbracket a \rrbracket_D(\rho) = \begin{cases} (c, 0) & \text{si } a \text{ est une constante } c \\ \rho(x) & \text{si } a \text{ est une variable } x \\ (\mathcal{F}_\sim(a_1 \diamond a_2), \mathcal{D}(a_1 \diamond a_2)) & \text{sinon avec } \diamond \in \{+, -, \times, \div\} \end{cases}$$

\mathcal{F}_\sim est la fonction définie à la figure 3.1 appliquant la norme IEEE 754 utilisant le mode d'arrondi au plus près et \mathcal{D} , définie à la figure 3.7, correspond au calcul des dérivées en un point. Une valeur réelle est décomposée en un couple (f, d) avec f la valeur flottante et d la valeur de la dérivée au

$$\begin{aligned}
a &= (f_a, d_a) \text{ et } b = (f_b, d_b) \\
\mathcal{D}(a + b) &= d_a + d_b \\
\mathcal{D}(a - b) &= d_a - d_b \\
\mathcal{D}(a \times b) &= d_a f_b + d_b f_a \\
\mathcal{D}\left(\frac{1}{a}\right) &= -\frac{d_a}{f_a^2} \text{ si } f_a \neq 0
\end{aligned}$$

FIG. 3.7 – Définition de la fonction \mathcal{D} .

point f . Comme nous l'avons mentionné précédemment, la fonction \mathcal{D} est définie par les règles de dérivation mathématique.

Cette méthode permet, par le biais des valeurs de la différentielle, une analyse de dépendances entre les variables présentes dans le programme. En effet, la différentielle est composée de dérivées partielles calculées par rapport aux variables indépendantes du programme. Si une dérivée partielle p associée à la variable x est nulle pour une variable indépendante v alors nous pouvons conclure que v ne dépend pas de x . A l'inverse, si p a la plus grande valeur non nulle parmi les dérivées partielles composant la différentielle de v alors la variable x est la plus influente dans le calcul de la variable v .

De plus, dans le cadre de la précision numérique, la connaissance de la dérivée de la fonction en un point permet de définir une approximation linéaire du résultat réel à partir de la valeur flottante. En effet, en considérant la valeur de l'ulp u d'une valeur flottant x , nous obtenons :

$$\mathbf{f}(x + u) \approx \mathbf{f}(x) + u\mathbf{f}'(x)$$

C'est en partie la base de la méthode CENA[Lan99] (Correction des erreurs numériques d'arrondi) qui essaie de compenser les erreurs d'arrondi au fil des calculs dans le but de calculer le résultat réel d'un programme numérique. Nous pouvons remarquer qu'il est possible d'appliquer plusieurs fois la méthode de différentiation automatique afin d'obtenir les dérivées d'ordre supérieur. Nous sommes alors capable, grâce aux dérivées d'ordre supérieur, de définir des approximations polynomiales.

Exemple 3.2 Prenons l'exemple de l'expression $p = x - ax$, tirée de [GP06], et calculons la dérivée de p par rapport à x . $a = (0.65, 0)$ avec 0.65 la valeur flottante et 0 la valeur de la dérivée de a par rapport à x . $x = (1, 1)$ où la première composante est la valeur flottante et la seconde représente la valeur de la dérivée de p par rapport à x .

$$\begin{aligned}
\mathcal{D}(p) &= \mathcal{D}(x) - \mathcal{D}(ax) \\
&= 1 - (0.65 \times 1 + 0 \times 1) \\
&= 0.35
\end{aligned}$$

x a une influence dans le calcul de p et nous en déduisons qu'une erreur initiale e sur x induit une approximation du résultat de a dans les flottants de $0.35e$.

Il existe deux réalisations possibles de la méthode de la différentiation automatique [BHN02, Gri00] : par surcharge d'opérateurs et par transformation de code source. La méthode par surcharge s'appuie sur le concept objet de certains langages de programmation et elle est appliquée par exemple au langage C++ à travers la librairie ADOL-C¹. C'est la plus simple des deux méthodes puisqu'elle nécessite uniquement la définition d'un nouveau type de données (c'est-à-dire une classe en programmation objets). L'inconvénient de cette méthode est une dégradation des performances tandis que son avantage est le peu de changements à effectuer dans le programme original.

¹<http://www.math.tu-dresden.de/~adol-c/>

La méthode de transformation de code source est utilisée dans les programmes ADIC² [BRMO97], ADIFOR³ [BCKM94] ou TAPENADE⁴ [HAP05] pour les langages C/C++ et Fortran respectivement. L'objectif est de modifier le programme original pour ajouter les instructions permettant le calcul des dérivées. Elle fait appel à des concepts issus de la compilation, par exemple la génération d'arbres de syntaxe abstraite. L'ajout de nouvelles instructions modifie la sémantique du programme et nécessite donc une grande expertise pour sa mise en œuvre. L'utilisation des techniques de compilation permet de mettre en place des analyses statiques et des optimisations afin d'obtenir de meilleures performances pour le code généré.

3.2.4 Domaine des flottants avec erreurs

Dans cette section, nous présentons la sémantique de l'erreur globale, notée $\llbracket \cdot \rrbracket_{\mathbb{E}}$. Nous présentons tout d'abord la sémantique concrète qui permet de calculer exactement la distance séparant un résultat flottant du résultat réel, puis la sémantique abstraite qui est basée sur l'arithmétique d'intervalles et qui permet, en pratique, de valider la précision numérique d'un programme. Nous présentons la sémantique de l'erreur globale qui permet de calculer globalement l'erreur due aux arrondis pour chaque variable du programme.

Sémantique concrète

L'objectif de l'arithmétique flottante avec erreurs est de mesurer la qualité d'un calcul flottant par rapport au même calcul réalisé avec l'arithmétique réelle. Une valeur réelle r est décomposée en deux valeurs (f, e) , avec f la valeur flottante et e l'erreur d'arrondi telle que :

$$r = f + e$$

e représente la distance séparant le flottant f du réel r . Ainsi, plus la valeur de e est petite et plus la qualité du calcul est grande.

Nous notons ρ l'environnement qui associe à chaque variable un flottant avec erreurs. Nous définissons la sémantique $\llbracket a \rrbracket_{\mathbb{E}}$, associée à cette arithmétique, suivant la structure d'une expression arithmétique a par :

$$\llbracket a \rrbracket_{\mathbb{E}}(\rho) = \begin{cases} (\uparrow_{\sim} c, \downarrow_{\sim} c) & \text{si } a = c \\ \rho(x) & \text{si } a = x \\ (\mathcal{F}_{\sim}(a_1 \diamond a_2), \mathcal{E}(a_1 \diamond a_2)) & \text{si } a = a_1 \diamond a_2 \end{cases}$$

La fonction \mathcal{F}_{\sim} , donnée à la figure 3.1, correspond à l'arithmétique flottante avec le mode d'arrondi au plus près. La fonction \mathcal{E} , donnée à la figure 3.8, permet de calculer l'erreur globale générée par l'expression arithmétique considérée. La fonction \mathcal{E} propage les erreurs entre les différents éléments d'une expression arithmétique. De plus, conformément à la norme IEEE 754, chaque opération introduit une nouvelle erreur issue de l'arrondi du résultat de cette opération. Par exemple, comme nous pouvons le voir à la figure 3.8, l'erreur pour une addition est $\mathcal{E}(a+b) = e_a + e_b + \downarrow_{\sim}(f_a + f_b)$: les erreurs sur les deux opérandes sont additionnées et l'erreur $\downarrow_{\sim}(f_a + f_b)$ due à l'addition flottante $\uparrow_{\sim}(f_a + f_b)$ est elle-même ajoutée au résultat. La sémantique de la soustraction est similaire à celle de l'addition. Quant à celle de la multiplication, elle découle du développement de $(f_a + e_a) \times (f_b + e_b)$. La définition du calcul de l'erreur pour l'opération de l'inverse est plus compliquée puisqu'elle fait intervenir une décomposition en série entière [Mar06, Mar05].

Cette sémantique permet de calculer le couple (f, e) à chaque point de contrôle du programme. Une instabilité numérique est détectée quand la valeur de l'erreur e est importante.

²<http://www-fp.mcs.anl.gov/adic/>

³<http://www-unix.mcs.anl.gov/autodiff/ADIFOR/>

⁴<http://www-sop.inria.fr/tropics/tapenade.html>

$$\begin{aligned}
a &= (f_a, e_a) \text{ et } b = (f_b, e_b) \\
\mathcal{E}(a + b) &= e_a + e_b + \downarrow_{\sim} (f_a + f_b) \\
\mathcal{E}(a - b) &= e_a - e_b + \downarrow_{\sim} (f_a - f_b) \\
\mathcal{E}(a \times b) &= e_a f_b + e_b f_a + e_a e_b + \downarrow_{\sim} (f_a \times f_b) \\
\mathcal{E}\left(\frac{1}{a}\right) &= \sum_{i=1}^{+\infty} (-1)^i \frac{e_a^i}{f_a^{i+1}} + \downarrow_{\sim} \left(\frac{1}{f_a}\right)
\end{aligned}$$

FIG. 3.8 – Définition de la fonction \mathcal{E} .

Sémantique abstraite

L'objectif de la sémantique abstraite, notée $\llbracket \cdot \rrbracket_{\mathbb{E}}^{\#}$, est d'étudier la précision numérique d'un programme pour des classes d'exécutions. Nous voulons valider le comportement numérique du programme pour des plages d'entrées possibles. Pour cela, les ensembles possibles d'entrées réelles R sont abstraits par un couple d'intervalles $([f], [e])$. L'intervalle $[f]$ est une sur-approximation de l'ensemble des flottants représentant R en machine et l'intervalle $[e]$ est une sur-approximation de l'ensemble des erreurs $\downarrow_{\sim}(x)$, $\forall x \in R$, avec le mode d'arrondi au plus près.

Nous étendons la fonction \downarrow_{\sim} à des valeurs intervalles grâce à la fonction \downarrow_{\sim}^I définie par l'équation (3.2). Cette fonction détermine l'intervalle d'erreurs pour le mode d'arrondi au plus près. Pour un intervalle $[a, b]$ avec a la borne inférieure et b la borne supérieure, l'erreur d'arrondi maximale η est donnée par la valeur de l'ulp de la plus grande borne, telle que définie à la section 3.1. L'ensemble des erreurs d'arrondi est alors donné par l'intervalle $[-\eta, \eta]$.

$$\downarrow_{\sim}^I([a, b]) = \frac{1}{2} \text{ulp}(\max(|a|, |b|)) \times [-1, 1] \quad (3.2)$$

La sémantique $\llbracket \cdot \rrbracket_{\mathbb{E}}^{\#}$ est définie sur la structure d'une expression arithmétique a par :

$$\llbracket a \rrbracket_{\mathbb{E}}^{\#} = (\mathcal{F}_{\sim}^{\#}(a), \mathcal{E}^{\#}(a))$$

$\mathcal{F}_{\sim}^{\#}$ et $\mathcal{E}^{\#}$ sont les extensions des fonctions \mathcal{F}_{\sim} et \mathcal{E} à l'arithmétique d'intervalles.

Exemple 3.3 Soit l'expression arithmétique $p = x - ax$ avec $x = 1$ dans les flottants et avec une erreur comprise dans l'intervalle $[-0.5, 0.5]$, c'est-à-dire que x varie dans les réels entre $[0.5, 1.5]$. Nous fixons $a = 0.65$ et nous considérons qu'aucune erreur n'est attachée à a . Nous considérons, pour simplifier, que les calculs flottants n'engendrent pas de nouvelle erreur.

$$\begin{aligned}
\mathcal{F}_{\sim}^{\#}(p) &= \mathcal{F}_{\sim}^{\#}(x) - \mathcal{F}_{\sim}^{\#}(ax) \\
&= [1, 1] - [0.65, 0.65] \times [1, 1] \\
&= [1, 1] - [0.65, 0.65] \\
&= [0.35, 0.35] \\
\mathcal{E}^{\#}(p) &= \mathcal{E}^{\#}(x) - \mathcal{E}^{\#}(ax) \\
&= [-0.5, 0.5] - ([0, 0] \times [1, 1] + [-0.5, 0.5] \times [0.35, 0.35] + [0, 0] \times [-0.5, 0.5]) \\
&= [-0.5, 0.5] - [-0.175, 0.175] \\
&= [-0.675, 0.675]
\end{aligned}$$

Le résultat réel est alors dans l'intervalle $[-0.325, 1.025]$ alors que le flottant est 0.35 il y a donc une erreur importante.

3.3 Intégration numérique

L'étude des programmes numériques passe également par la connaissance des algorithmes qui les composent. Dans le cas des modèles Simulink, étudiés dans la seconde partie de ce document, les principaux algorithmes utilisés sont ceux permettant la résolution de systèmes d'équations différentielles. Nous présentons, à la section 3.3.2, les principales méthodes numériques [Jed06, chap. 7] utilisées pour résoudre ce problème. Nous mettrons en évidence qu'elles ne fournissent qu'une solution approchée d'où la nécessité d'utiliser d'autres méthodes pour calculer une solution garantie. Nous présentons, à la section 3.3.3, la méthode d'intégration numérique garantie basée sur les formes de Taylor [NJC99].

3.3.1 Problématique

Soit une fonction $\mathbf{f} : U \rightarrow \mathbb{R}^n$ continue sur un ouvert U de $\mathbb{R} \times \mathbb{R}^n$. Nous considérons l'équation différentielle $\dot{x} = \mathbf{f}(t, x)$ avec la condition initiale $x(t_0) = x_0 \in \mathbb{R}^n$, $\dot{x} = dx/dt$. La solution unique x de cette équation, dans le cas où \mathbf{f} est continue et Lipschitz de rapport $k > 0$, est donnée par :

$$x(t) = x_0 + \int_{t_0}^t \mathbf{f}(s, x(s)) ds \quad (3.3)$$

Les théorèmes prouvant l'existence et l'unicité de la solution x se basent sur l'opérateur de Picard-Lindelöf et du théorème du point fixe de Banach. Pour plus de détails à ce propos, nous renvoyons le lecteur à [HNW93].

La difficulté de résoudre mathématiquement ce genre de problème a conduit à la définition de méthodes d'intégration numérique fournissant une solution approchée. Ces méthodes se basent toutes sur une discrétisation du problème transformant les équations différentielles en équations récurrentes. En particulier, la solution approchée est obtenue par une interpolation polynomiale sur un nombre fini de points de discrétisation. Les méthodes d'intégration numérique sont classées en deux catégories :

- les méthodes à pas séparés qui utilisent uniquement le point courant pour calculer x_{n+1} ;
- les méthodes à pas liés qui nécessitent la connaissance des n précédentes valeurs de x , pour calculer x_{n+1} .

Une autre information importante concernant ces méthodes est leur ordre. Une méthode d'intégration est d'ordre k , si l'erreur commise est nulle quand la fonction \mathbf{f} est un polynôme de degré inférieur ou égale à k .

3.3.2 Principales méthodes d'intégration

Méthode d'Euler

La méthode de *Euler* est la plus simple et la plus ancienne des méthodes d'intégration numérique à pas séparés. Cette méthode repose sur l'approximation $dx/dt \approx (x_{n+1} - x_n)/h$. Le pas d'intégration, noté h , représente la durée d'un pas d'intégration, c'est-à-dire $h = t_{n+1} - t_n$. Elle existe en deux versions :

- Euler explicite :

$$\frac{x_{n+1} - x_n}{h} \approx \mathbf{f}(t_n, x_n) \quad (3.4)$$

- Euler implicite :

$$\frac{x_{n+1} - x_n}{h} \approx \mathbf{f}(t_{n+1}, x_{n+1}) \quad (3.5)$$

L'équation (3.4) est dite explicite puisque la valeur x_{n+1} est obtenue uniquement à partir de x_n . Quant à l'équation (3.5) est qualifiée d'implicite puisque x_{n+1} dépend de la valeur de \mathbf{f} en x_{n+1} .

Runge-Kutta

La méthode de Runge et Kutta fait référence à un schéma numérique général à pas liés, donné à l'équation (3.6). Ce schéma est associé à plusieurs méthodes d'intégration numérique en fonction de la valeur k qui définit l'Ordre de la méthode.

$$\begin{cases} y_{i+1} &= y_i + h(b_1 k_1 + \dots + b_r k_r) \\ x_{i+1} &= x_i + h \\ k_j &= \mathbf{f}(x_i + c_j h, y_i + h(a_{j1} k_1 + \dots + a_{jr} k_r)) \end{cases} \quad (3.6)$$

Méthode d'ordre 1. Nous considérons le cas où $r = 1$. Nous obtenons alors un cas particulier en modélisant ainsi la méthode d'Euler :

- $b_1 = 1$ et $a_{11} = 0$ nous obtenons la méthode d'Euler explicite ;
- $b_1 = 1$ et $a_{11} = 1$ nous obtenons la méthode d'Euler implicite.

Méthode d'ordre 2. Il existe plusieurs méthode d'ordre $r = 2$ suivant les évaluations des b_i dans l'expression $b_1 k_1 + b_2 k_2$. Les principales sont :

- Méthode de *Heun* ($b_1 = 1/2$ et $b_2 = 1/2$) :

$$\begin{cases} x_{n+1} = x_n + h(k_1 + k_2)/2 \\ k_1 = \mathbf{f}(t_n, x_n) \\ k_2 = \mathbf{f}(t_n + h, x_n + k_1) \end{cases}$$

- La méthode de Runge-Kutta (proprement dite) ($b_1 = 0$ et $b_2 = 1$) :

$$\begin{cases} x_{n+1} = x_n + h k_2 \\ k_1 = \mathbf{f}(t_n, x_n) \\ k_2 = \mathbf{f}(t_n + h/2, x_n + h k_1) \end{cases}$$

Méthode d'ordre 3. Il existe encore plusieurs méthodes d'ordre $r = 3$ basées sur le schéma de Runge-Kutta. Nous présentons cependant une seule donnée par le système suivant :

$$\begin{cases} x_{n+1} = x_n + h(k_1 + 4k_2 + k_3)/6 \\ k_1 = \mathbf{f}(t_n, x_n) \\ k_2 = \mathbf{f}(t_n + h/2, x_n + h k_1/2) \\ k_3 = \mathbf{f}(t_n + h, x_n - h k_1 + 2h k_2) \end{cases}$$

Les méthodes précédentes font partie de la famille des méthodes à pas d'intégration fixe, c'est-à-dire que le pas d'intégration h ne varie pas. Néanmoins, pour augmenter l'efficacité, donc diminuer le temps de calcul, il est courant d'utiliser des méthodes à pas variable. Nous présentons la méthode de Merson qui est la première méthode d'intégration de Runge-Kutta emboîtée. Plus précisément, cette méthode utilise deux méthodes de Runge-Kutta, la première permet de calculer la solution approchée tandis que la seconde permet d'évaluer l'erreur de consistance pour contrôler le pas. Cette méthode est définie par :

$$\begin{cases} k_1 = \mathbf{f}(t_n, x_n) \\ k_2 = \mathbf{f}(t_n + h/3, x_n + h k_1/3) \\ k_3 = \mathbf{f}(t_n + h/3, x_n - h k_1 + h k_2/6) \\ k_4 = \mathbf{f}(t_n + h/2, x_n + h k_1/8 + 3h k_2/8) \\ k_5 = \mathbf{f}(t_n + h, x_n + h k_1/2 - 3h k_3/2 + 2h k_1) \\ x_{n+1} = x_n + h(k_1 + 4k_2 + k_3)/6 \\ x_{n+1}^* = x_n + h(k_1 - 3k_3 + 4k_4)/2 \end{cases}$$

L'erreur e_n à chaque pas d'intégration est donnée par :

$$e_n = |x_{n+1} - x_{n+1}^*|$$

Cette méthode utilise une méthode de Runge-Kutta d'ordre 5 pour calculer la solution et une méthode d'ordre 4 pour contrôler le pas. Soit ϵ la tolérance fixée *a priori*, l'algorithme de Merson utilise les règles suivantes :

- si $e_n > \epsilon$ alors il divise h par 2 ;
- si $e_n \leq \epsilon/64$ alors il multiplie h par 2 ;
- h est invariant dans les autres cas.

Ce rapide tour d'horizon des méthodes numériques d'intégration, servant à résoudre les équations différentielles, permet de souligner l'utilisation d'approximations, aussi complexes soient-elles, pour obtenir une solution. Autrement dit, ces méthodes ne calculent qu'une solution approchée qui peut être loin de la réalité mathématique comme le montre l'exemple 3.4. Il faut remarquer que la méthode de Runge-Kutta d'ordre 3 donne un bien meilleur résultat, c'est-à-dire une approximation plus fine. Cependant, mettre en défaut une méthode est toujours possible et cela dépend grandement du problème considéré.

Exemple 3.4 Nous considérons l'équation différentielle $\dot{x} = 1$ avec $x(0) = 1$. La solution mathématique de cette équation est $x(t) = \exp(t)$. La figure 3.9 donne les représentations graphiques des intégrations numériques par la méthode d'Euler et de Runge-Kutta d'ordre 2 ainsi que celle de la fonction \exp .

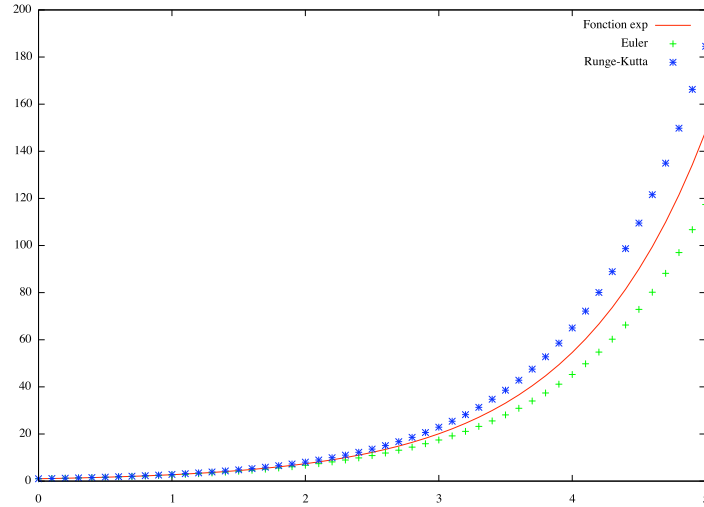


FIG. 3.9 – Résultats des intégrations numériques (Euler et Runge-Kutta ordre 2) par rapport au résultat mathématique (\exp).

3.3.3 Intégration numérique garantie

Dans cette section, nous présentons succinctement la méthode d'intégration numérique garantie par la méthode de Taylor. Un nombre plus important de détails peut être obtenu dans [NJC99]. Nous avons montré dans la section 3.3.2 que les méthodes d'intégration numérique ne calculent qu'une solution approchée. Cependant, dans certains cas, nous avons besoin de garantie sur le résultat.

Problématique

Nous considérons une équation différentielle avec un ensemble de valeurs initiales X_0 , telle que :

$$\dot{x}(t) = \mathbf{f}(x(t)) \text{ avec } x(0) \in X_0 \quad (3.7)$$

L'ensemble des solutions \mathbf{X} est donné par :

$$\mathbf{X} = \{x(t) \mid \exists x_0 \in X_0 \text{ tel que } \dot{x}(t) = \mathbf{f}(x(t)) \text{ avec } x(0) = x_0\}$$

Nous présentons la problématique générale de l'intégration numérique garantie qui considère un ensemble de valeurs initiales. Nous constatons que garantir la solution d'une équation suivant une valeur initiale unique est un cas particulier.

L'objectif de la méthode d'intégration numérique garantie est de calculer une approximation sûre de l'équation (3.7) (voir [BM06] pour une méthode basée sur l'algorithme de *Runge-Kutta*), c'est-à-dire de trouver une suite d'instant t_k et une suite d'intervalles $[x_k]$ telles que :

$$\forall x \in \mathbf{X}, \forall k \in \mathbb{N}, x(t_k) \in [x_k]$$

Méthode de Taylor

La méthode de Taylor utilise le développement de Taylor-Lagrange de la fonction x à tous les instants t_k et la solution garantie courante $[x_k]$ pour calculer le prochain intervalle solution $[x_{k+1}]$ suivant l'équation (3.8). La valeur de h représente le pas d'intégration.

$$\begin{aligned} x(t_{k+1}) &= x(t_k) + \sum_{i=1}^{N-1} \frac{h^i}{i!} \frac{d^i x}{dt^i}(t_k) + \frac{h^N}{N!} \frac{d^N x}{dt^N}(\tilde{t}) \\ &\in [x_k] + \sum_{i=1}^{N-1} h^i \mathbf{f}^{(i-1)}(x(t_k)) + h^N \mathbf{f}^{(N-1)}(x(\tilde{t})) \\ &\in [x_k] + \sum_{i=1}^{N-1} h^i \mathbf{f}^{(i-1)}([x_k]) + h^N \mathbf{f}^{(N-1)}([\tilde{x}]) = [x_{k+1}] \end{aligned} \quad (3.8)$$

$\mathbf{f}^{(j)}$ est j -ième dérivée de \mathbf{f} et $h^N \mathbf{f}^{(N-1)}(x(\tilde{t}))$ est le reste de Lagrange tel que $\tilde{t} \in]t_k, t_{k+1}[$.

La méthode de Taylor est une méthode d'intégration à pas séparés et variables, c'est-à-dire qu'elle n'utilise que la connaissance de la valeur courante pour calculer la prochaine valeur. Mais elle peut également modifier dynamiquement le pas d'intégration. L'ordre de la méthode peut arbitrairement être augmenté pour résoudre un problème grâce au développement en série de Taylor, cependant il reste fixe pendant tout le processus de résolution.

La méthode de Taylor est décomposée en deux parties pour calculer une solution garantie de l'équation (3.7) :

- la première étape permet de calculer un encadrement *a priori* de la valeur de $[\tilde{x}]$. En d'autres termes, il faut trouver $[\tilde{x}]$, à partir de l'intervalle garanti courant $[x_k]$ à l'instant t_k , tel que :

$$\forall x_k \in [x_k], \quad \dot{x}(t) = f(x) \quad \text{avec } x(t_k) = x_k$$

a une unique solution $x(t) \in [\tilde{x}]$. $[\tilde{x}]$ permet de fournir un encadrement du reste de la formule de Taylor-Lagrange et il est calculé en utilisant l'opérateur de Picard-Lindelöf et le théorème du point fixe de Banach. Pour plus de détails se référer à [NJC99, section 5] ;

- la seconde étape calcule une valeur plus précise de l'intervalle $[x_{k+1}]$ à partir de $[\tilde{x}]$. L'une des difficultés de la méthode est de limiter l'explosion de la taille des intervalles manipulés. L'article [NJC99, section 7] expose un certain nombre de méthodes permettant de limiter cette explosion.

Deuxième partie

Analyse statique de Simulink

Lola Jost : « Tu connais la métaphysique du puzzle, Barthélemy ?
[...] Il suffit d'une unique pièce et tout à coup l'univers tient en un
seul morceau. A condition, bien sûr, de se contenter d'un univers
raisonnable. Un univers à notre portée.[...] »

Dominique Sylvain, *Passage du Désir*.

La complexité des systèmes embarqués devient difficilement gérable au niveau de leur spécification. De plus en plus, les concepteurs ont recours à des outils logiciels pour les aider dans cette tâche. Ces outils reposent sur des langages de programmation de haut niveau et donc ces spécifications deviennent des programmes. Nous pouvons alors appliquer les techniques d'analyse statique pour valider des propriétés sur ces spécifications.

Présentation de Simulink

Le logiciel Simulink est le standard de fait dans l'industrie pour l'activité de modélisation, de spécification et de simulation des systèmes embarqués, pas nécessairement critiques. Par exemple, il est très fréquent de trouver une modélisation Simulink quasi complète des systèmes électromécaniques des véhicules automobiles chez les constructeurs.

Simulink est une extension du logiciel de calcul scientifique Matlab. Il permet de modéliser et de simuler des systèmes évoluant dans le temps. Par abus de langage, Simulink désigne également le langage graphique permettant de représenter ces systèmes. Les techniques et méthodes constituant le logiciel Simulink sont issues de l'analyse numérique. Le moteur de simulation, appelé *solver*, c'est-à-dire le cœur de l'outil, est centré sur les méthodes d'intégration numérique. L'analyse statique par interprétation abstraite des modèles Simulink doit alors être capable de prendre en compte ces algorithmes puisqu'ils définissent la sémantique du langage.

Ce chapitre est organisé de la manière suivante. Nous présentons le langage Simulink à la section 4.1, la génération des équations sémantiques à la section 4.2, le processus de simulation à la section 4.3 et, pour finir, à la section 4.4, un aperçu de l'analyse statique des modèles Simulink telle qu'elle est définie dans cette thèse.

4.1 Langage Simulink

Un modèle (c'est-à-dire un programme) Simulink est représenté graphiquement par un diagramme en blocs, c'est-à-dire un ensemble de boîtes connectées par des fils. Ces boîtes sont appelées *blocs* et les fils sont appelés *signaux*. Les signaux sont des fonctions du temps et les blocs sont des opérations sur ces fonctions. D'un point de vue sémantique, les comportements des modèles Simulink sont alors décrits par des fonctions du temps. Nous notons t la variable du temps continue et k la variable du temps discret.

Le langage Simulink est formé de très nombreux blocs allant des opérations arithmétiques ou de comparaisons aux générateurs de signaux aléatoires en passant par la définition d'opérations personnalisées écrites en Matlab, C/C++, Fortran ou encore Ada. Ces blocs sont regroupés en bibliothèques. Par exemple la bibliothèque *Math Operations* regroupe les opérations mathématiques, la bibliothèque *Logic and Bit Operations* collecte les opérations logiques et de manipulation de bits, la bibliothèque *Continuous* réunit les opérations sur les fonctions continues du temps ou encore la bibliothèque *Discrete* rassemble les opérations sur les fonctions discrètes du temps.

Simulink distingue, en particulier, deux types de modèles : ceux à temps continu et ceux à temps discret. Cette distinction se fait selon que les modèles utilisent les blocs de la bibliothèque *Continuous* ou *Discrete*. Un modèle qui mélange des opérations de ces deux bibliothèques est alors













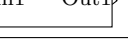
Bibliothèque	Blocs	Désignation	Description
<i>Sources</i>	 ℓ_1	$\text{In}(\ell_1)$	Entrée
	 ℓ_1	$\text{Cst}(c, \ell_1)$	Constante
<i>Sinks</i>	$\ell_1 \times$ 	$\text{Out}(\ell_1)$	Sortie
<i>Math operations</i>	 ℓ_3	$\text{Add}(\ell_1, \ell_2, \ell_3)$	Addition
	 ℓ_3	$\text{Sub}(\ell_1, \ell_2, \ell_3)$	Soustraction
	 ℓ_3	$\text{Mul}(\ell_1, \ell_2, \ell_3)$	Multiplication
	 ℓ_3	$\text{Div}(\ell_1, \ell_2, \ell_3)$	Division
	 ℓ_2	$\text{Gain}(g, \ell_1, \ell_2)$	Multiplication par une constante
<i>Logic and Bit Operations</i>	 ℓ_3	$\text{Cmp}(rel, \ell_1, \ell_2, \ell_3)$	Opération de comparaisons
<i>Routing Signal</i>	 ℓ_4	$\text{Switch}(p, \ell_1, \ell_2, \ell_3, \ell_4)$	Conditionnelle
<i>Continuous</i>	 ℓ_2	$\text{Integr}(init, \ell_1, \ell_2)$	Intégration
<i>Discrete</i>	 ℓ_2	$\text{UDelay}(init, \ell_1, \ell_2)$	Décalage temporel
<i>SubSystem</i>	 ℓ_2	$\text{Sys}(\ell_1, M, \ell_2)$	Sous-système

FIG. 4.1 – Sous ensemble du langage Simulink.

un modèle hybride. *A contrario*, un modèle n'utilisant pas ces opérations est alors soit à temps continu soit à temps discret. La distinction est donnée par le type des fonctions d'entrée. Nous assimilons les modèles à temps continu comme des descriptions des environnements physiques des logiciels embarqués représentés par des modèles à temps discret.

Nous concentrons notre étude sur un sous-ensemble de ce langage. La figure 4.1 liste les opérations qui composeront les modèles Simulink que nous analyserons. Nous considérons les blocs associés aux entrées **In** et aux sorties **Out** d'un système ainsi que le bloc générant un signal constant **Constant**. Nous considérons les opérations arithmétiques **Add**, **Sub**, **Product**, **Division** et le cas particulier d'une multiplication par une constante **Gain**, les opérations de comparaison **RelationalOperations** où *rel* représente une des opérations de comparaisons $\{==, <, \leq, >, \geq, \neq\}$. Dans la figure 4.1 nous donnons la représentation graphique de l'opérateur $==$. Nous prenons également en compte l'expression conditionnelle **Switch** et les opérateurs temporels : l'intégration continue du temps **Integrator** et le décalage temporel discret **UnitDelay**. De plus, nous considérons les sous-systèmes c'est-à-dire la désignation d'un diagramme en blocs par un unique bloc **SubSystem**. Nous donnons dans la section 4.2 les traductions de blocs plus complexes dans ce noyau de langage Simulink.

L'exemple 4.1 illustre, d'une part, la grande expressivité du langage Simulink et ses aptitudes à la modélisation des systèmes évoluant dans le temps. D'autre part, il montre la relation étroite entre le modèle mathématique et les modèles Simulink. De plus, il met en évidence la relation forte qui lie Simulink et les méthodes d'analyse numérique, en particulier les algorithmes d'intégration numérique. En effet, les comportements des systèmes décrits en Simulink sont obtenus par des méthodes numériques. La section 4.3 décrit le processus de simulation, c'est-à-dire d'exécution,

des modèles Simulink.

Exemple 4.1 Nous considérons un système de régulation du papillon des gaz d'une automobile¹. Le modèle Simulink est donné à la figure 4.2. Le système complet S (figure 4.2(a)) est une composition de plusieurs sous-systèmes :

- Un sous-système à temps discret S_1 (voir figure 4.2(b)) représentant un régulateur PI (Proportionnel Intégral) régi par le système d'équations :

$$\begin{cases} y(k) = y_p(k) + y_i(k) \\ y_p(k) = K_p e(k) \\ y_i(k+1) = y_i(k) + K_i T_s e(k) \quad 0 < y_i(k) < 1 \end{cases}.$$

$e(k)$ est la k -ième entrée du système et $y(k)$ est la k -ième sortie. K_p est la coefficient de régulation de la partie proportionnelle ; T_s est le pas d'intégration pour la méthode numérique d'intégration (ici Euler explicite, voir section 3.3.2) et K_i est le coefficient multiplicateur de la partie intégrale ;

- Un sous-système à temps continu S_3 (voir figure 4.2(c)) décrivant la dynamique du papillon des gaz suivant le système d'équations différentielles :

$$\begin{cases} T(t) = \text{Direction} \times \text{Effort} \times C_s \\ \dot{\omega}(t) = \frac{1}{J}(-K_s(\theta(t) - \theta_{eq}) - K_d\omega(t) + T(t)) \quad 0 < \theta < \pi/2 \\ \text{si } (\theta < 0 \wedge \text{sgn}(\dot{\omega}(t)) = -1) \vee ((\theta > \pi/2 \wedge \text{sgn}(\dot{\omega}(t)) = 1)) \text{ alors } \dot{\omega}(t) = 0 \end{cases}.$$

En fonction des consignes fournies par le régulateur PI, c'est-à-dire la direction (In2) du moteur électrique contrôlant le papillon des gaz ainsi que son effort (In1) (la vitesse de fermeture ou d'ouverture), ce système modélise la dynamique (position angulaire θ et la vitesse angulaire ω) du papillon des gaz. Le modèle contient un certain nombre de paramètres issus des éléments du système physique : K_s est le coefficient d'élasticité d'un ressort, C_s est le moment d'inertie du papillon, K_d est la constante de viscosité, J est une constante représentant l'inertie du papillon et θ_{eq} est la vitesse de déplacement du papillon de la position 0 à $\pi/2$ uniquement soumis à l'élasticité du ressort. sgn est la fonction de signe. La contrainte est exprimée par une extension du bloc **Integrator1**, qui est une combinaison d'un bloc **Integrator** et un bloc **Saturation**, contraignant le domaine des valeurs de sortie de l'intégration ;

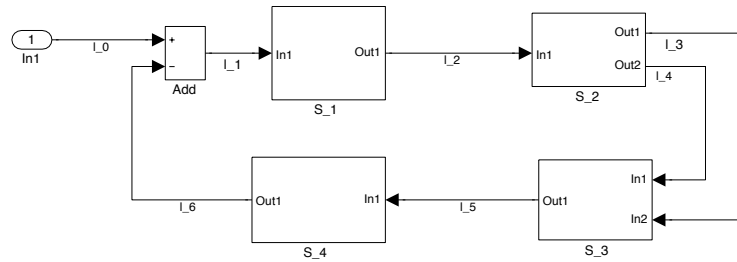
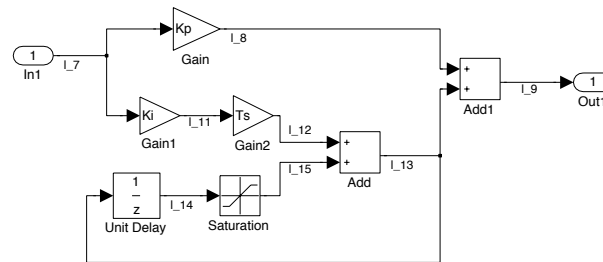
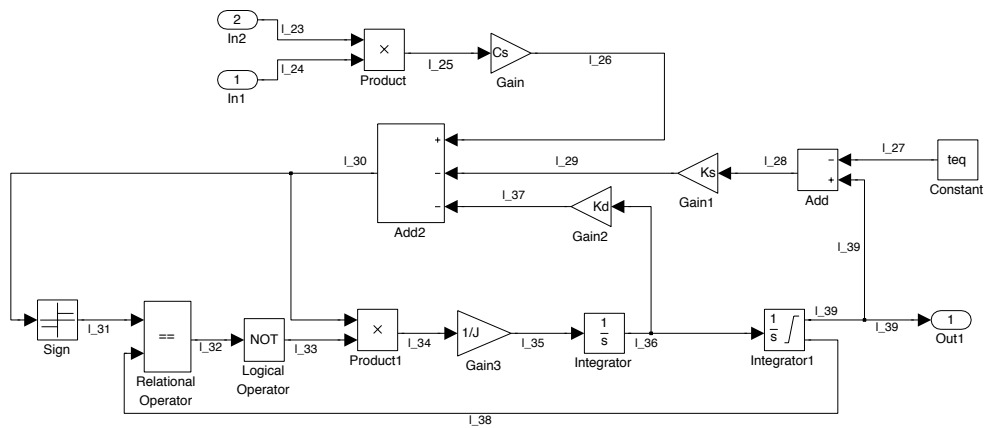
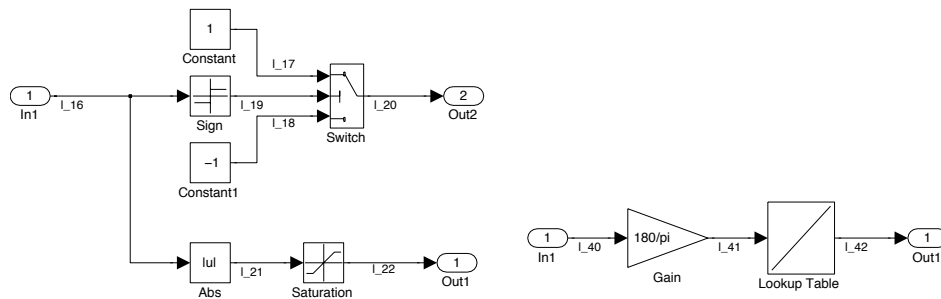
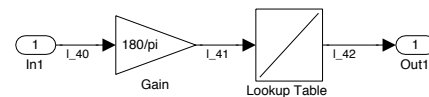
- Un sous-système S_4 (voir figure 4.2(e)), ayant le rôle d'un capteur, convertit la position angulaire \angle° du papillon des gaz en une tension électrique. Les valeurs de l'effort sont données par une interpolation linéaire entre 0.5 et 4.5 ;

$$\begin{cases} \angle^\circ = \frac{180}{\pi} \times u \\ y = \begin{cases} 0.5 & \text{si } \angle^\circ = 0^\circ \\ 4.5 & \text{si } \angle^\circ = 90^\circ \end{cases} \end{cases}.$$

- Un sous-système S_2 (voir figure 4.2(d)), jouant le rôle d'un actionneur, traduit le résultat du régulateur PI en des consignes direction (Out1) et effort (Out2) au système continu suivant les équations :

$$\begin{aligned} \text{Effort} &= \begin{cases} |u| & \text{si } 0 \leq |u| \leq 1 \\ 1 & \text{si } |u| > 1 \end{cases}, \\ \text{Direction} &= \begin{cases} 1 & \text{si } u \leq 0 \\ -1 & \text{si } u > 0 \end{cases}. \end{aligned}$$

¹Exemple issu de la formation Mathworks *SL01 : Simulink for System and Algorithm Modeling*.

(a) *Système complet (S).*(b) *Système à temps discret (S₁).*(c) *Système à temps continu (S₃).*(d) *Sous-système (S₂).*(e) *Sous-système (S₄).*FIG. 4.2 – *Contrôleur papillon des gaz en Simulink.*

4.2 Equations sémantiques

Nous montrons dans cette section comment nous associons un système d'équations à un modèle Simulink. Un modèle à temps continu est associé à un système d'équations différentielles, un modèle à temps discret est associé à un système d'équations récurrentes. Un modèle hybride est représenté par un système d'équations différentielles et récurrentes. Un modèle sans opérateur temporel est alors associé à un système d'équations algébriques.

Nous notons $E[M]$ le système d'équations associé au modèle Simulink M composé par des opérations listées à la figure 4.1. Une équation est associée à chaque bloc (c'est-à-dire opération) et représente la relation qui lie les entrées aux sorties de ce bloc. Une étiquette ℓ est attachée à chaque fil d'un modèle Simulink, nous notons $L[M]$ l'ensemble fini des étiquettes. De plus, une étiquette s est aussi associée à chaque bloc **Integrator** et **UnitDelay** afin de mettre en évidence les états du modèle, nous notons $S[M]$ l'ensemble fini des ces étiquettes. Nous notons par $V[M]$ l'ensemble fini des étiquettes d'un modèle Simulink M tel que $V[M] = L[M] \cup S[M]$.

La grammaire associée au langage des équations est donnée par l'équation (4.1). Nous considérons les expressions composées de constantes r , de variables x , des opérations arithmétiques $\diamond \in \{+, -, \times, \div\}$ et d'opérations de comparaisons $*$ $\in \{=, <, \leq, >, \geq, \neq\}$. De plus, nous considérons aussi les expressions conditionnelles $\text{if } (p_r(e_1), e_2, e_3)$ avec $p_r(e_1)$ un test de la forme $e_1 * r$ avec r une constante réelle et $*$ une opération de comparaisons. Nous considérons également les appels de fonctions $\mathbf{f}(\vec{\ell}_{in})$, qui représenteront les sous-systèmes à plusieurs entrées $\vec{\ell}_{in}$. Nous distinguons deux types d'équations, les équations algébriques représentées par la règle eq et les équations différentielles ou récurrentes décrites par la règle eq_s . Intuitivement, les équations algébriques seront associées aux opérations non temporelles (par exemple une addition) tandis que les équations différentielles ou récurrentes seront associées aux blocs **Integrator** et **UnitDelay** respectivement. Un sous-système est associé à la règle sys , il est constitué de deux systèmes d'équations : le premier représente la fonction de sortie et le second la fonction d'états. Un modèle est alors représenté par un ensemble de sous-systèmes avec des conditions initiales \vec{v}_0 et le nom du sous-système initial \mathbf{f}_0 .

$$\begin{aligned}
e &::= r \mid x \mid e_1 \diamond e_2 \mid e_1 * e_2 \mid \text{if } (p_r(e_1), e_2, e_3) \mid \mathbf{f}(\vec{x}_{in}) \\
\text{eq} &::= x = e \\
\text{eq}_s &::= \dot{x} = e \mid x(k+1) = e \\
\text{sys} &::= \mathbf{f} : \vec{x}_{in} \mapsto (\{\text{eq}\}, \{\text{eq}_s\}) \\
\text{mdl} &::= (\mathbf{f}_0, \{\text{sys}\})
\end{aligned} \tag{4.1}$$

Un modèle Simulink M est décrit par deux systèmes d'équations : le premier, noté $E^o[M]$, représente la fonction de sortie du modèle et le second, noté $E^s[M]$, représente la fonction d'état. Nous notons $E[M]$ le système d'équations associé au modèle Simulink M tel que $E[M] = E^o[M] \cup E^s[M]$. Les règles de génération des équations sont données à la figure 4.3. Un port d'entrée **In** génère une équation qui associe la valeur d'un signal ℓ_1 à la valeur de l'entrée. Une constante c génère un signal constant. Un port de sortie associe la valeur d'un signal ℓ_1 à la variable du sortie **Out**. Les équations générées par les blocs opérations arithmétiques $\{+, -, \times, \div\}$ et le bloc **Gain** sont très intuitives. Le bloc **Switch** est représenté par une expression conditionnelle. Il faut noter que le bloc **switch** n'autorise que les opérations de comparaisons $>$, \geq et \neq . Le bloc **Integrator** engendre la création d'une équation différentielle du premier ordre tandis que le bloc **Unit Delay** est associé à une équation récurrente du premier ordre. Ces deux blocs sont les seuls, dans la partie du langage Simulink considéré, introduisant la notion de temps. L'élément important est l'équation différentielle du premier ordre qui est un élément nouveau dans un langage devant être statiquement analysé. L'équation associée à un sous-système représente l'appel de la fonction \mathbf{f} associée au système d'équations engendré par le diagramme en bloc M . L'absence d'équation d'état est notée par \emptyset .

Le sous-ensemble du langage Simulink que nous utilisons permet de représenter des opérations plus complexes de la bibliothèque Simulink, notamment celles qui sont employées dans

$E[\text{In}(\ell_1)]$	$= (\{\ell_1 = \text{In}\}, \emptyset)$
$E[\text{Cst}(c, \ell_1)]$	$= (\{\ell_1 = c\}, \emptyset)$
$E[\text{Out}(\ell_1)]$	$= (\{\text{Out} = \ell_1\}, \emptyset)$
$E[\text{Add}(\ell_1, \ell_2, \ell_3)]$	$= (\{\ell_3 = \ell_1 + \ell_2\}, \emptyset)$
$E[\text{Sub}(\ell_1, \ell_2, \ell_3)]$	$= (\{\ell_3 = \ell_1 - \ell_2\}, \emptyset)$
$E[\text{Mul}(\ell_1, \ell_2, \ell_3)]$	$= (\{\ell_3 = \ell_1 \times \ell_2\}, \emptyset)$
$E[\text{Div}(\ell_1, \ell_2, \ell_3)]$	$= (\{\ell_3 = \ell_1 \div \ell_2\}, \emptyset)$
$E[\text{Gain}(g, \ell_1, \ell_2)]$	$= (\{\ell_2 = g \times \ell_1\}, \emptyset)$
$E[\text{Cmp}(rel, \ell_1, \ell_2, \ell_3)]$	$= (\{\ell_3 = \ell_1 \text{ rel } \ell_2\}, \emptyset)$
$E[\text{Switch}(p, \ell_1, \ell_2, \ell_3, \ell_4)]$	$= (\{\ell_4 = \text{if}(p(\ell_1), \ell_2, \ell_3)\}, \emptyset)$
$E[\text{Integr}(init, \ell_1, \ell_2)]$	$= (\{\ell_2(t) = x(t)\}, \{\dot{x}(t) = \ell_1(t)\})$
$E[\text{UDelay}(init, \ell_1, \ell_2)]$	$= (\{\ell_2(k) = x(k)\}, \{x(k+1) = \ell_1(k)\})$
$E[\text{Sys}(\vec{\ell}_1, M, \vec{\ell}_2)]$	$= (\{\vec{\ell}_2 = \mathbf{f}(\vec{\ell}_1)\}, \emptyset)$ avec $\mathbf{f} : \vec{\ell}_{in} \mapsto E[M]$

FIG. 4.3 – Règles de générations des équations.

l'exemple 4.1. Nous donnons une liste non exhaustive des blocs, bibliothèque par bibliothèque, qui peuvent être représentés par le langage donné à l'équation (4.1) :

- Dans la bibliothèque *Math Operations* :
 - Les blocs **Sgn** et **Abs**, décrivant la fonction signe ou la valeur absolue d'un signal, peuvent être définis avec une combinaison d'expressions conditionnelles. Les équations associées à ces blocs sont :

$$E[\text{Sgn}(\ell_1, \ell_2)] = (\{\ell_2 = \text{if}(\ell_1 = 0, 0, \text{if}(\ell_1 > 0, 1, -1))\}, \emptyset) \text{ et}$$

$$E[\text{Abs}(\ell_1, \ell_2)] = (\{\ell_2 = \text{if}(\ell_1 = 0, 0, \text{if}(\ell_1 > 0, \ell_1, -\ell_1))\}, \emptyset) ;$$

- Dans la bibliothèque *Discontinuities* :
 - Le bloc **Saturation** a pour fonction de contraindre un signal dans un domaine de valeurs bornées $[m, M]$, m est la borne inférieure et M la borne supérieure du domaine. Ce bloc peut également être défini à partir d'une combinaison d'expressions conditionnelles. Nous obtenons les équations suivantes :

$$E[\text{Saturation}(m, M, \ell_1, \ell_2)] = (\{\ell_2 = \text{if}(\ell_1 \geq M, M, \text{if}(\ell_1 \leq m, m, \ell_1))\}, \emptyset) ;$$

- Dans la bibliothèque *Continuous* :
 - Le bloc **State-Space** définit un système linéaire invariant du temps (*LTI system*) par un ensemble d'équations différentielles linéaires du premier ordre. La traduction dans notre représentation des modèles Simulink est alors immédiate :

$$\begin{cases} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{cases}$$

avec $u(t)$ le signal d'entrée, $y(t)$ le signal de sortie et $x(t)$ le signal associé à l'état du système. A, B, C, D sont des coefficients réels. Il faut noter que ce système d'équations peut aussi représenter un système multi-entrées et multi-sorties (*MIMO system*). Dans ce cas, $u(t)$, $y(t)$ et $x(t)$ sont des vecteurs et A, B, C, D sont des matrices ;

- Le bloc **TransferFcn** définit un système linéaire invariant du temps dans le domaine des fréquences. Une fonction de transfert est obtenue à partir d'une transformée de Laplace. Ce bloc, en suivant les règles de conversion entre fonction de transfert et représentation *State-Space*, peut également être représenté dans notre sous ensemble de langage Simulink ;
- Dans la bibliothèque *Discrete* :
 - Le bloc **Discrete-time integrator** est défini par un système d'équations récurrentes du premier ordre suivant une méthode d'intégration (*Euler* ou *Heun* [PTVF92, Jed06]). Le système S_1 à la figure 4.2(b) décrit dans sa partie inférieure un intégrateur discret ;

- Les blocs **Discrete State-Space** et **Discrete TransferFcn** sont les équivalents à temps discret des blocs **State-Space** et **TransferFcn**. Ils s'appuient sur des systèmes d'équations récurrentes du premier ordre et la transformée en Z. Ils peuvent être également décrits dans un système d'équations utilisant uniquement les opérations de notre langage. Par exemple, le bloc **Discrete State-Space** est décrit en Simulink par le système d'équations :

$$\begin{cases} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{cases},$$

avec $u(k)$ le signal d'entrée, $y(k)$ le signal de sortie et $x(k)$ le signal associé à l'état du système. A, B, C, D sont des coefficients réels. Ce bloc peut également représenter des systèmes multi-entrées et multi-sorties comme son analogue à temps continu. Encore une fois, la représentation dans notre formalisme est immédiate ;

- Dans la bibliothèque *Lookup Tables* :
 - Le bloc **Lookup Table** définit une table de correspondances entre un signal d'entrée et un signal de sortie. Cette définition s'appuie sur des algorithmes d'interpolation polynomiale. Nous prenons pour simplifier le cas où l'entrée et la sortie sont décrites par deux couples de valeurs (e_1, e_2) et (s_1, s_2) respectivement. Le calcul de la sortie est donné par une interpolation linéaire de l'entrée. Nous obtenons une équation de la forme

$$E[\text{lookup}((e_1, e_2), (s_1, s_2), \ell_1, \ell_2)] = \ell_2 = \ell_1 * \frac{s_1 - s_2}{e_1 - e_2} + \frac{e_1 s_2 - e_2 s_1}{e_1 - e_2}.$$

Exemple 4.2 Nous reprenons le système décrit dans l'exemple 4.1 et nous donnons les systèmes d'équations obtenus suivant les règles de générations décrites à la figure 4.3. Nous montrons ainsi qu'il est possible de retrouver le modèle mathématique sous-jacent aux modèles Simulink. En outre, nous obtenons un système d'équations différentielles pour le modèle à temps continu et un système d'équations récurrentes pour le modèle à temps discret.

- Les systèmes d'équations associés au modèle, noté S , de la figure 4.2(a) sont :

$$E^o[S] = \left\{ \begin{array}{l} \ell_0 = \text{In1} \\ \ell_1 = \ell_0 - \ell_6 \\ \ell_2 = S_1(\ell_1) \\ (\ell_3, \ell_4) = S_2(\ell_2) \\ \ell_5 = S_3(\ell_3, \ell_4) \\ \ell_6 = S_4(\ell_5) \end{array} \right\} \quad \text{et} \quad E^s[S] = \emptyset ;$$

- Les systèmes d'équations associés au modèle, noté S_1 , de la figure 4.2(b) sont :

$$E^o[S_1] = \left\{ \begin{array}{l} \ell_7 = \text{In1} \\ \ell_8 = K_p \times \ell_7 \\ \ell_9 = \ell_8 + \ell_{13} \\ \ell_{11} = K_i \times \ell_7 \\ \ell_{12} = T_s \times \ell_{11} \\ \ell_{13} = \ell_{12} + \ell_{15} \\ \ell_{14} = x(k) \\ \ell_{15} = \text{if}(\ell_{14} > 1, 1, \text{if}(\ell_{14} < 0, 0, \ell_{14})) \\ \text{Out1} = \ell_9 \end{array} \right\} \quad \text{et} \quad E^s[S_1] = \{x(k+1) = \ell_{13}\} ;$$

- Les systèmes d'équations associés au modèle, noté S_2 , de la figure 4.2(d) sont :

$$E^o[S_2] = \left\{ \begin{array}{l} \ell_{16} = \text{In1} \\ \ell_{17} = 1 \\ \ell_{18} = -1 \\ \ell_{19} = \text{if}(\ell_{16} = 0, 0, \text{if}(\ell_{16} > 0, 1, -1)) \\ \ell_{20} = \text{if}(\ell_{19} < 0, \ell_{17}, \ell_{18}) \\ \ell_{21} = \text{if}(\ell_{16} \geq 0, \ell_{16}, 0 - \ell_{16}) \\ \ell_{22} = \text{if}(\ell_{21} > \frac{\pi}{2}, \frac{\pi}{2}, \text{if}(\ell_{21} < 0, 0, \ell_{21})) \\ \text{Out2} = \ell_{20} \\ \text{Out1} = \ell_{22} \end{array} \right\} \quad \text{et} \quad E^s[S_2] = \emptyset ;$$

- Les systèmes d'équations associés au modèle, noté S_3 , de la figure 4.2(c) sont :

$$E^o[S_3] = \left\{ \begin{array}{l} \ell_{23} = \text{In2} \\ \ell_{24} = \text{In1} \\ \ell_{27} = \text{teq} \\ \ell_{25} = \ell_{23} \times \ell_{24} \\ \ell_{26} = C_s \times \ell_{25} \\ \ell_{28} = \ell_{39} - \ell_{27} \\ \ell_{29} = K_s \times \ell_{28} \\ \ell_{37} = K_d \times \ell_{36} \\ \ell_{31} = \text{if}(\ell_{30} = 0, 0, \\ \quad \text{if}(\ell_{30} > 0, 1, -1)) \end{array} \right\} \cup \left\{ \begin{array}{l} \ell_{30} = \ell_{26} - \ell_{29} - \ell_{37} \\ \ell_{32} = \ell_{31} == \ell_{38} \\ \ell_{33} = 1 - \ell_{32} \\ \ell_{34} = \ell_{30} \times \ell_{33} \\ \ell_{35} = \frac{1}{J} \times \ell_{34} \\ \ell_{36} = x_1(t) \\ \ell_{38} = \text{if}(x_2(t) > \text{Max}, \text{Max}, \\ \quad \text{if}(x_2(t) < \text{Min}, \text{Min}, x_2(t))) \\ \text{Out} = \ell_{39} \end{array} \right\}$$

$$\text{et} \quad E^s[S_3] = \left\{ \begin{array}{l} \dot{x}_1(t) = \ell_{35} \\ \dot{x}_2(t) = \ell_{36} \end{array} \right\} ;$$

- Les systèmes d'équations associés au modèle, noté S_4 , de la figure 4.2(e) sont :

$$E^o[S_4] = \left\{ \begin{array}{l} \ell_{40} = \text{In1} \\ \ell_{41} = \ell_{40} \times \frac{180}{\pi} \\ \ell_{42} = \text{lookup}(\ell_{41}) \\ \text{Out} = \ell_{42} \end{array} \right\} \quad \text{et} \quad E^s[S_4] = \emptyset .$$

Il faut souligner que la représentation des modèles Simulink telle qu'elle est définie dans cette section est très proche de la représentation mathématique *State-Space* [Son98, HP05]. Un système dynamique est représenté par une fonction f qui donne la sortie $y(t)$ du système à partir de l'entrée $u(t)$ et de l'état $x(t)$ et par une fonction g qui décrit l'évolution des états \bar{x} . En d'autres termes un système S est représenté par :

$$S = \left\{ \begin{array}{l} \bar{x}(t) = g(u(t), x(t)) \\ y(t) = f(u(t), x(t)) \end{array} \right. , \quad (4.2)$$

\bar{x} représente soit une évolution continue (\dot{x}) soit une évolution discrète ($x(k+1)$). t représente dans ce cas soit le temps continu soit le temps discret. Cette représentation permet de mettre en évidence la partie du modèle Simulink sur laquelle il faut appliquer les algorithmes numériques.

Nous remarquons également que le système d'équations généré par les règles de la figure 4.3 est en forme SSA (*Single Static Assignment*) [App98, chap. 19]. Cette représentation assure qu'une variable ne possède qu'une seule définition. Les propriétés associées à cette forme, en particulier la simplicité de la représentation, font qu'elle est souvent utilisée en optimisation à la compilation. Elle est, par conséquent, très adaptée pour l'analyse statique par interprétation abstraite.

4.3 Processus de simulation

Nous avons montré dans les sections précédentes le lien étroit existant entre la modélisation mathématique et les modèles Simulink. En l'occurrence, ces derniers représentent des systèmes d'équations différentielles et/ou récurrentes. Dans l'outil Simulink, la résolution de ces systèmes d'équations est obtenue par application des méthodes de l'analyse numérique [PTVF92, Jed06], en particulier les méthodes d'intégration numérique.

Le moteur de simulation, appelé *solver*, est paramétré par les méthodes numériques et un nombre important d'options comme le type des données utilisées pour les calculs (par exemple nombres flottants simple ou double précision ou encore nombres à virgule fixe). Ces paramètres et ces options induisent des comportements différents de la part du moteur de simulation et ainsi différentes solutions des modèles Simulink. La critique faite à l'encontre de Simulink, du point de vue de la théorie des langages de programmation, est l'absence de sémantique rendant difficile l'application de méthodes formelles. Nous adoptons le point de vue de [CCM⁺03a] en considérant que Simulink possède plusieurs sémantiques dépendantes des paramètres et options du moteur de simulation. En particulier, l'analyse statique que nous définissons dans cette thèse est capable de prendre en compte ces différences.

Le *solver* de Simulink a un fonctionnement très proche d'un compilateur. En effet, un certain nombre d'étapes préalables à la simulation est effectué : nous pouvons citer l'évaluation des expressions paramètres des blocs, le typage des données, ou encore une optimisation par réduction du nombre de blocs ou le calcul de l'ordre d'évaluation des blocs. Cependant, la boucle de simulation des modèles Simulink telle qu'elle est décrite dans la documentation² est très simple et elle est décrite par l'algorithme suivant :

- 1: **répéter**
- 2: Calcul des sorties
- 3: Calcul des états
- 4: Calcul du prochain pas de temps
- 5: **jusqu'à** Temps fin de simulation

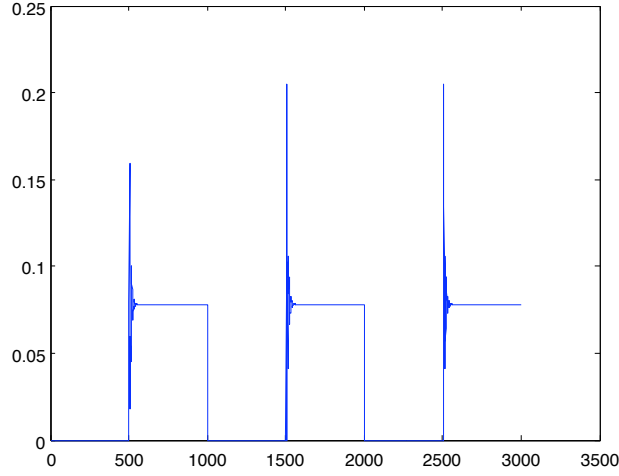
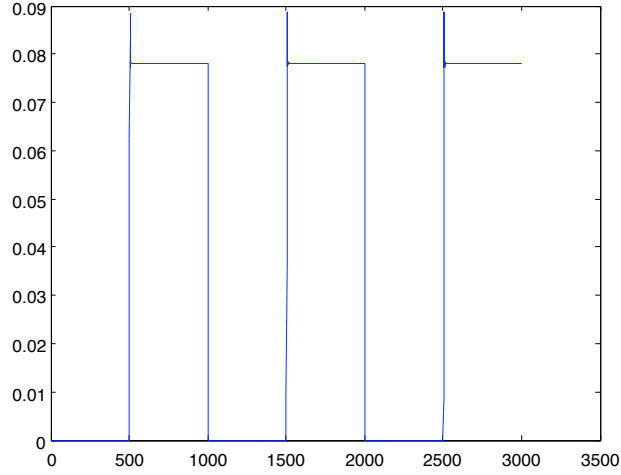
L'application des méthodes d'intégration numérique est effectuée à l'étape 2 de cet algorithme. Le résultat de cette boucle de simulation est alors la production de séquences de valeurs. Ces séquences sont un résultat approché des fonctions associées à la solution du modèle mathématique. L'un des objectifs de l'analyse statique de modèles Simulink est d'apporter une estimation de la distance séparant les résultats de la simulation des résultats mathématiques.

Les travaux présentés dans ce document s'intéressent uniquement aux deux premières étapes de cet algorithme. En effet, nous faisons l'hypothèse que les algorithmes d'intégration numériques sont à pas fixe. Cette hypothèse est issue de l'idée que les systèmes embarqués utilisent des capteurs fonctionnant avec une fréquence d'échantillonnage fixe afin de répondre aux contraintes du temps réel. La simulation des spécifications de ces systèmes en Simulink doit alors être contrainte de la même façon. De plus, d'un point de vue fonctionnel, nous assimilons l'activité du capteur, c'est-à-dire l'échantillonnage temporel, à la discrétisation temporelle induite par les algorithmes d'intégration numérique. La possibilité de générer automatiquement du code à partir des spécifications Simulink est également contrainte par l'utilisation de méthodes d'intégration numérique à pas fixe.

Exemple 4.3 Nous reprenons une nouvelle fois le système dont le modèle Simulink a été donné à l'exemple 4.1. Nous donnons deux résultats de simulation produit par Simulink à la figure 4.4. Le premier résultat (voir figure 4.4(a)) est obtenu en utilisant la méthode Euler comme méthode d'intégration numérique. Le second résultat (voir figure 4.4(b)) est obtenu en utilisant la méthode de Runge-Kutta d'ordre 4. Dans les deux cas nous considérons un pas d'intégration fixe de 0.01 pour une simulation de 30 secondes. L'axe des abscisses représente le temps. L'entrée du système est une fonction périodique rectangulaire d'amplitude 1 et de période 10 secondes.

Les résultats montrent une grande similitude dans les comportements, ils sont périodiques et de même période. Cependant, le domaine de valeurs qui est de $[0, 0.2]$ avec la méthode d'Euler et

²<http://www.mathworks.com>

(a) *Sortie du système S_3 (Euler).*(b) *Sortie du système S_3 (Runge-Kutta d'ordre 4).*FIG. 4.4 – *Simulations avec différentes méthodes d'intégration.*

de $[0, 0.09]$ avec la méthode Runge-Kutta montre un écart qui est uniquement due à la différence entre méthodes numériques.

4.4 Aperçu de l'analyse statique

Nous présentons brièvement dans la suite de cette section les différents éléments qui composent notre analyse statique de modèles Simulink. Nous rappelons que l'objectif de la simulation abstraite est de calculer automatiquement la distance entre le modèle mathématique et la simulation numérique telle qu'elle est réalisée par Simulink. Cette distance nous donne une information sur la qualité numérique des simulations d'exécutions des logiciels embarqués critiques.

La section 4.3 a présenté le processus de simulation, c'est-à-dire l'exécution des modèles Simulink. Nous avons alors mis en évidence que les solutions approchées des systèmes d'équations, différentielles et/ou récurrentes, associés à ces modèles étaient données par des séquences de valeurs. Nous considérons alors que la sémantique de Simulink est donnée par des ensembles de

séquences.

Nous définissons, dans les chapitres suivants, l'analyse statique de modèles Simulink à partir de deux abstractions des séquences. La figure 4.5 donnent une vision globale des abstractions qui vont être définies pour mener à bien l'analyse statique des modèles hybrides Simulink. \mathcal{V} est un domaine numérique et \mathcal{V}^\sharp est le domaine numérique abstrait associé. $\mathbb{N} \rightarrow \wp(\mathcal{V})$ est l'ensemble des séquences sur $\wp(\mathcal{V})$ et $\mathbb{N} \rightarrow \mathcal{V}^\sharp$ est sa version abstraite. μ est une fonction de partition de \mathbb{N} et $\mathbb{P}(\mathbb{N})$ est le treillis des partitions de \mathbb{N} ordonné par la relation de raffinement. Ces deux abstractions ont pour objectif, d'une part, de prendre en compte des ensembles de comportements grâce à la correspondance de Galois (α_1, γ_1) et, d'autre part, de travailler uniquement avec des séquences de longueur finie grâce à la correspondance de Galois (α_μ, γ_μ) , nous travaillerons uniquement sur les partitions finies de \mathbb{N} .

$$\begin{array}{ccc}
 \langle \mathbb{N} \rightarrow \wp(\mathcal{V}), \preceq_\subseteq \rangle & \xleftrightarrow[\alpha_1]{\gamma_1} & \langle \mathbb{N} \rightarrow \mathcal{V}^\sharp, \preceq_\subseteq \rangle \\
 \alpha_\mu \downarrow \uparrow \gamma_\mu & & \alpha_\mu \downarrow \uparrow \gamma_\mu \\
 \langle \mathbb{P}(\mathbb{N}) \rightarrow \wp(\mathcal{V}), \preceq_\subseteq \rangle & \xleftrightarrow[\alpha_1]{\gamma_1} & \langle \mathbb{P}(\mathbb{N}) \rightarrow \mathcal{V}^\sharp, \preceq_\subseteq \rangle
 \end{array}$$

FIG. 4.5 – *Diagramme des abstractions.*

Nous allons définir deux domaines numériques abstraits. Le premier est basé sur les formes de Taylor et il est défini à la section 5.1. Le second est une extension du domaine des nombres flottants avec erreurs utilisant des formes de Taylor pour améliorer le calcul des termes d'erreurs ; il est défini à la section 5.2.

De plus, il est possible de définir un ensemble d'abstractions du domaine des séquences en se basant sur le treillis des partitions. En effet, une relation d'équivalence \sim_μ sur un ensemble X_1 est définie par une fonction $\mu : X_1 \rightarrow D$ telle que :

$$\forall a, b \in X_1, a \sim b \Leftrightarrow \mu(a) = \mu(b).$$

La relation d'équivalence \sim_μ définit alors une partition $P_\mu(X_1)$ de l'ensemble X_1 . En fonction de la place de $P_\mu(X_1)$ dans le treillis des partitions de X_1 , nous pouvons en déduire une information sur la précision des analyses statiques ainsi définies. Nous présentons dans la section 5.3 le domaine des séquences et ses abstractions.

Au chapitre 6, nous définirons en nous appuyant sur les domaines des formes de Taylor, des nombres flottants avec erreurs différenciées et des séquences une analyse statique par interprétation abstraite des modèles hybrides Simulink. Le chapitre 7 présentera des résultats expérimentaux obtenus à partir d'un prototype d'analyseur statique développé au cours de la thèse.

Nous présentons dans ce chapitre les domaines abstraits qui seront utilisés dans la définition d'une analyse statique de modèles Simulink. Le domaine des formes de Taylor est une extension du domaine des intervalles. Il permettra d'utiliser les algorithmes d'intégration numérique avec des valeurs intervalles. Le domaine des nombres flottants avec erreurs différenciées est issu de la combinaison du domaine des flottants avec erreurs et du domaine des formes de Taylor. Ce qui permet d'améliorer le calcul des termes d'erreurs. Le domaine des séquences permet de représenter de manière finie des suites infinies de valeurs. Il sera utilisé pour définir une analyse statique associée à des simulations de modèles Simulink sur un temps infini.

Le chapitre est organisé de manière suivante : le domaine numérique des formes de Taylor est défini à la section 5.1. Le domaine numérique des nombres flottants avec erreurs différenciées est défini à la section 5.2. Le domaine des séquences est défini à la section 5.3.

5.1 Domaine des formes de Taylor

Dans cette section, nous définissons l'un des deux domaines numériques utilisés dans la définition d'une analyse statique de modèles Simulink.

Le domaine numérique considéré est celui des formes de Taylor. Il permet de calculer le domaine des valeurs réelles des variables d'un programme. Ce domaine est une extension de celui des intervalles et il a l'avantage d'être un domaine relationnel, c'est-à-dire qu'il prend en compte de manière automatique les relations entre les variables. De plus, l'utilisation des techniques de différenciation automatique permet d'avoir une implémentation efficace et permet de définir des formes de Taylor avec un ordre variable. La flexibilité dans les ordres des formes de Taylor permet alors d'avoir une paramétrisation de la précision et du coût des analyses statiques utilisant ce domaine.

Le domaine numérique des formes de Taylor est issu d'une combinaison des techniques de différenciation automatique et de l'arithmétique d'intervalles. Nous apportons dans cette thèse une approche sémantique de la différenciation automatique qui permet l'immersion des formes de Taylor dans la théorie de l'interprétation abstraite.

Cette section est découpée comme suit. La génération des coefficients des formes de Taylor et l'arithmétique sur ces formes sont définies à la section 5.1.1. Les définitions des domaines concrets et abstraits sont introduites aux sections 5.1.2 et 5.1.3 respectivement.

5.1.1 Génération et arithmétique des formes de Taylor

Nous empruntons le vocabulaire de la différentiation automatique. Nous appelons *variables indépendantes* les variables à partir desquelles nous différentions les fonctions. En général elles représentent les variables d'entrée du programme. Les *variables dépendantes* sont les variables pour lesquelles nous voulons connaître les valeurs des différentielles. En général elles représentent les sorties du programmes.

La génération des coefficients s'appuie sur la règle de différentiation en chaîne que nous rappelons à la proposition 5.1. Moore [Moo79] a défini grâce à cette règle une manière automatique et efficace de générer les coefficients des formes de Taylor.

Proposition 5.1 *Soit \mathbf{f} une fonction continue sur $[a, b]$ et dérivable en un point x de $[a, b]$ et soit \mathbf{g} une fonction définie sur un intervalle I contenant l'image de \mathbf{f} , dérivable en $\mathbf{f}(x)$. Alors, la fonction :*

$$\mathbf{h}(x) = \mathbf{g}(\mathbf{f}(x)) \quad (a \leq x \leq b) \text{ est dérivable en } x \text{ et de dérivée } \mathbf{h}'(x) = \mathbf{g}'(\mathbf{f}(x))\mathbf{f}'(x)$$

Nous notons \mathcal{T}^N l'ensemble de formes de Taylor avec un reste d'ordre N . Nous rappelons la définition d'une forme de Taylor d'ordre N à l'équation (5.1). $\mathbf{f} : \mathbb{R} \rightarrow \mathbb{R}$ est une fonction N fois continûment différentiable sur l'intervalle $[x_1, x_2]$, R_N est le reste de Lagrange et nous notons $h = |x_2 - x_1|$ la largeur de l'intervalle.

$$\mathbf{f}(x_2) = \mathbf{f}(x_1) + \sum_{i=1}^{N-1} \frac{h^i}{i!} \mathbf{f}^{(i)}(x_1) + R_N \quad (5.1)$$

$\mathbf{f}^{(i)}$ représente la i -ième dérivée de \mathbf{f} . Cette définition est aisément adaptable aux fonctions de plusieurs variables.

Nous représentons une forme de Taylor d'ordre N par un vecteur de dimension $N + 1$ où la dernière composante est le reste. Nous notons $T^N(\mathbf{f})(x)$ la forme de Taylor d'ordre N associée à la fonction \mathbf{f} au point x et elle est définie par :

$$T^N(\mathbf{f})(x) = \left(\mathbf{f}(x), h\mathbf{f}^{(1)}(x), \dots, \frac{h^{N-1}}{(N-1)!} \mathbf{f}^{(N-1)}(x), R_N \right) \quad (5.2)$$

Nous notons par $(T^N(\mathbf{f})(x))_i$ le i -ième coefficient de la forme de Taylor d'ordre N associée à la fonction \mathbf{f} au point x avec $(T^N(\mathbf{f})(x))_0 = \mathbf{f}(x)$. L'arithmétique de Taylor est définie à la figure 5.1. Nous notons par $+_{\mathcal{T}}, -_{\mathcal{T}}, \times_{\mathcal{T}}, \div_{\mathcal{T}}$ l'addition, la soustraction, la multiplication et la division de formes de Taylor.

$$\begin{aligned} T^N(\mathbf{f}_1)(x) +_{\mathcal{T}} T^N(\mathbf{f}_2)(x) &= \forall i, (T^N(\mathbf{f}_1)(x))_i + (T^N(\mathbf{f}_2)(x))_i \\ T^N(\mathbf{f}_1)(x) -_{\mathcal{T}} T^N(\mathbf{f}_2)(x) &= \forall i, (T^N(\mathbf{f}_1)(x))_i - (T^N(\mathbf{f}_2)(x))_i \\ T^N(\mathbf{f}_1)(x) \times_{\mathcal{T}} T^N(\mathbf{f}_2)(x) &= \forall i, \sum_{j=0}^i (T^N(\mathbf{f}_1)(x))_j (T^N(\mathbf{f}_2)(x))_{i-j} \\ T^N(\mathbf{f}_1)(x) \div_{\mathcal{T}} T^N(\mathbf{f}_2)(x) &= \forall i, \frac{1}{T^N(\mathbf{f}_2)(x)} \left\{ (T^N(\mathbf{f}_1)(x))_i - \sum_{r=1}^i (T^N(\mathbf{f}_2)(x))_r \left(\frac{(T^N(\mathbf{f}_1)(x))}{(T^N(\mathbf{f}_2)(x))} \right)_{i-r} \right\} \end{aligned}$$

FIG. 5.1 – Opérations arithmétiques sur les formes de Taylor.

Nous restreignons notre présentation aux opérations utiles pour l'analyse de modèles Simulink. Il existe cependant des règles de générations des coefficients des formes de Taylor pour les fonctions \sin , \cos , \exp , etc. [Moo79].

5.1.2 Domaine concret

Le domaine concret $\langle \wp(\mathcal{T}^N), \subseteq \rangle$ forme un treillis complet. Nous notons $\wp(S)$ l'ensemble des parties de l'ensemble S . Nous définissons la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{T}}$ des expressions à la figure 5.2 en étendant les opérations arithmétiques à l'arithmétique de Taylor.

Nous notons θ un environnement, c'est-à-dire une fonction associant une forme de Taylor à chaque variable. Nous notons Θ l'ensemble de ces environnements et $E \subseteq \Theta$ est un ensemble d'environnements concrets. Une valeur constante c est associée à une fonction constante, c'est-à-dire que toutes ses dérivées sont nulles. \diamond représente l'une des opérations arithmétiques parmi $\{+, -, \times, \div\}$ et $\diamond_{\mathcal{T}}$ son équivalent en arithmétique de Taylor. La notation $*$ représente une des opérations de comparaison $\{<, \leq, > \geq, =, \neq\}$. La fonction $\mathcal{A}(T^N(\mathbf{f}_1)(x))$ calcule la valeur réelle associée à la forme de Taylor de la fonction \mathbf{f}_1 au point x . La sémantique des comparaisons génère un ensemble de valeurs booléennes suivant le résultat de la comparaison entre les valeurs réelles associées aux formes de Taylor. La sémantique de l'expression conditionnelle est donnée par l'union des formes de Taylor obtenues en évaluant la branche "vrai" et celles obtenues par l'évaluation de la branche "faux".

$$\begin{aligned}
\llbracket c \rrbracket_{\mathbb{T}}(E) &= \{(c, 0, \dots, 0)\} \\
\llbracket \ell \rrbracket_{\mathbb{T}}(E) &= \{\theta(\ell) \mid \theta \in E\} \\
\llbracket e_1 \diamond e_2 \rrbracket_{\mathbb{T}}(E) &= \{t_1 \diamond_{\mathcal{T}} t_2 \mid \forall \theta \in E \wedge t_1 \in \llbracket e_1 \rrbracket_{\mathbb{T}}(\theta) \wedge t_2 \in \llbracket e_2 \rrbracket_{\mathbb{T}}(\theta)\} \\
\llbracket e_1 * e_2 \rrbracket_{\mathbb{T}}(E) &= \{b \mid \forall \theta \in E \wedge t_1 \in \llbracket e_1 \rrbracket_{\mathbb{T}}(\theta) \wedge t_2 \in \llbracket e_2 \rrbracket_{\mathbb{T}}(\theta) \wedge b = \mathcal{A}(t_1) * \mathcal{A}(t_2)\} \\
\llbracket \text{if}(p_r(e_1), e_2, e_3) \rrbracket_{\mathbb{T}}(E) &= \{t_2 \mid \forall \theta \in E \wedge p_r(\mathcal{A}(\llbracket e_1 \rrbracket_{\mathbb{T}}(\theta))) = \text{true} \wedge t_2 \in \llbracket e_2 \rrbracket_{\mathbb{T}}(\theta)\} \\
&\quad \cup \{t_3 \mid \forall \theta \in E \wedge p_r(\mathcal{A}(\llbracket e_1 \rrbracket_{\mathbb{T}}(\theta))) = \text{false} \wedge t_3 \in \llbracket e_3 \rrbracket_{\mathbb{T}}(\theta)\}
\end{aligned}$$

FIG. 5.2 – Sémantique concrète des expressions.

Les formes de Taylor gardent une trace des relations entre les variables du programme. En effet, les dérivées d'une fonction permettent de calculer l'influence des variables dans un calcul. L'exemple 5.1 met en évidence ce phénomène.

Exemple 5.1 *Considérons la suite d'instructions suivante :*

instruction	valeur	d/dx_1	d/dx_2
$x1 = v_1;$	v_1	1	0
$x2 = v_2;$	v_2	0	1
$y = (x1 + x2) \times 0.5;$	$(v_1 + v_2) \times 0.5$	0.5	0.5
$z = x1 - 2 \times y;$	$-v_2$	0	-1

Nous considérons x_1 et x_2 les variables indépendantes du programme. Les dérivées du premier ordre des expressions par rapport à x_1 et x_2 montrent que la valeur de z ne dépend que de x_2 . Cet exemple met en évidence la conservation des relations entre variable dans les formes de Taylor.

5.1.3 Domaine abstrait

Nous définissons le domaine numérique abstrait des formes de Taylor en nous fondant sur le domaine des intervalles [CC77]. Cependant, l'application directe de cette abstraction conduit en général à des résultats imprécis et inutilisables. Néanmoins, en utilisant une extension du théorème des accroissements finis, c'est-à-dire la formule de Taylor-Lagrange, nous pouvons définir une abstraction plus fine, nommée *fonction d'inclusion de Taylor* [JKDW01].

Pour toute fonction $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}$, $(N - 1)$ fois continûment différentiable sur l'intervalle $[a, b]$ et N fois différentiable sur $]a, b[$, la formule de Taylor-Lagrange est :

$$\forall u \in [a, b], \exists z \in]a, b[, \mathbf{f}(u) = \sum_{i=0}^{N-1} \frac{u-c}{i!} \mathbf{f}^{(i)}(c) + \mathbf{f}^{(N)}(z)(u-c)$$

c représente le centre de l'intervalle $[a, b]$ et $\mathbf{f}^{(i)}$ est la i -ième dérivée de \mathbf{f} . D'où,

$$\mathbf{f}([a, b]) \subseteq \sum_{i=0}^{N-1} \frac{u-c}{i!} \mathbf{f}^{(i)}(c) + [\mathbf{f}^{(N)}]([a, b])([a, b] - c)$$

$[\mathbf{f}^{(N)}]$ représente l'évaluation de $\mathbf{f}^{(N)}$ avec l'arithmétique d'intervalles c'est-à-dire qu'elle calcule une borne de la N -ième dérivée sur $[a, b]$. Nous obtenons alors une abstraction basée sur le domaine des intervalles qui en général produit de meilleurs résultats.

Soit $(\alpha_{\mathbb{I}}, \gamma_{\mathbb{I}})$ la correspondance de Galois entre le treillis complet $(\wp(\mathbb{R}), \subseteq)$ et celui des intervalles $(\mathbb{I}, \sqsubseteq_{\mathbb{I}})$. Nous notons par $\mathbf{m}([a, b])$ le centre de l'intervalle $[a, b]$ défini à l'équation (3.1) par $(a+b)/2$. Nous notons $\mathcal{T}^{\#}$ l'ensemble des formes de Taylor abstraites et nous notons $T^N(\mathbf{f})$ une forme de Taylor abstraite. Nous notons par $\mathcal{A}^{\#}((T^N(\mathbf{f}))(x))$ la fonction d'évaluation des formes de Taylor centrées au point x en une valeur intervalle. Dans la proposition 5.2, la relation d'ordre $\sqsubseteq_{\mathcal{T}}^{\#}$ compare les valeurs intervalles associées aux formes de Taylor. L'opération d'union $\sqcup_{\mathcal{T}}^{\#}$ associe le centre de l'enveloppe convexe des $N-1$ premiers éléments, $[r_1, r_2]$ représente l'enveloppe convexe des points r_1 et r_2 , et elle calcule l'union des N -ième éléments des formes de Taylor. L'opération d'intersection $\sqcap_{\mathcal{T}}^{\#}$ est similaire à $\sqcup_{\mathcal{T}}^{\#}$ excepté le N -ième élément qui est obtenu en calculant l'intersection des intervalles.

Proposition 5.2 $\langle \mathcal{T}^{\#}, \sqsubseteq_{\mathcal{T}}^{\#}, \sqcup_{\mathcal{T}}^{\#}, \sqcap_{\mathcal{T}}^{\#}, \perp_{\mathcal{T}}^{\#}, \top_{\mathcal{T}}^{\#} \rangle$ est un treillis complet et il est défini par :

$$\begin{aligned} T^N(\mathbf{f}_1)(x) \sqsubseteq_{\mathcal{T}}^{\#} T^N(\mathbf{f}_2)(x) &\stackrel{\text{def}}{=} (\mathcal{A}^{\#}(T^N(\mathbf{f}_1)(x))) \sqsubseteq_{\mathbb{I}}^{\#} (\mathcal{A}^{\#}(T^N(\mathbf{f}_2)(x))) \\ T^N(\mathbf{f}_1)(x) \sqcup_{\mathcal{T}}^{\#} T^N(\mathbf{f}_2)(x) &\stackrel{\text{def}}{=} (z_0^{\#}, z_1^{\#}, \dots, z_N^{\#}) \text{ où} \\ \forall i < N, z_i^{\#} &= \mathbf{m} \left(\left[\left(T^N(\mathbf{f}_1)(x) \right)_i, \left(T^N(\mathbf{f}_2)(x) \right)_i \right] \right) \text{ et } z_N^{\#} = \left(T^N(\mathbf{f}_1)(x) \right)_N \sqcup_{\mathbb{I}} \left(T^N(\mathbf{f}_2)(x) \right)_N \\ T^N(\mathbf{f}_1)(x) \sqcap_{\mathcal{T}}^{\#} T^N(\mathbf{f}_2)(x) &\stackrel{\text{def}}{=} (z_0^{\#}, z_1^{\#}, \dots, z_N^{\#}) \text{ où} \\ \forall i < N, z_i^{\#} &= \mathbf{m} \left(\left[\left(T^N(\mathbf{f}_1)(x) \right)_i, \left(T^N(\mathbf{f}_2)(x) \right)_i \right] \right) \text{ et } z_N^{\#} = \left(T^N(\mathbf{f}_1)(x) \right)_N \sqcap_{\mathbb{I}} \left(T^N(\mathbf{f}_2)(x) \right)_N \end{aligned}$$

Preuve Nous considérons le produit cartésien de dimension $N+1$ du treillis des intervalles $(\mathbb{I}, \sqsubseteq_{\mathbb{I}})$. Le produit cartésien de treillis complets est un treillis complet. La représentation centrée est alors assimilée à une fonction de réduction. \square

Le treillis complet des formes de Taylor centrées intervalles, muni de la fonction \mathbf{m} est vu comme un produit réduit de treillis d'intervalles. En effet, l'abstraction par des intervalles, coefficient par coefficient, d'un ensemble de forme de Taylor conduit à considérer un grand nombre d'éléments redondants. La fonction d'inclusion centrée permet alors d'obtenir une forme normale réduite. Cette réduction permet de gagner en précision puisqu'un grand nombre d'éléments abstraits représentant les mêmes formes de Taylor concrètes est ainsi éliminé.

Proposition 5.3 (Correspondance de Galois de formes de Taylor centrées)

$$\langle \wp(\mathcal{T}^N), \subseteq \rangle \xleftrightarrow[\alpha_{\mathcal{T}}]{\gamma_{\mathcal{T}}} \langle \mathcal{T}^{\#}, \sqsubseteq_{\mathcal{T}}^{\#} \rangle$$

avec $\alpha_{\mathcal{T}}(\{T^N(\mathbf{f})(x_j) : 1 \leq j \leq n\}) \stackrel{\text{def}}{=} (z_0^{\#}, z_1^{\#}, \dots, z_N^{\#})$ où

$$\forall i < N, z_i^{\#} = \mathbf{m} \left(\left\{ \left(T^N(\mathbf{f})(x_j) \right)_i : 1 \leq j \leq n \right\} \right) \text{ et } z_N^{\#} = \alpha_{\mathbb{I}} \left(\{ (T^N(\mathbf{f}_j)(x))_N : 1 \leq j \leq n \} \right)$$

et $\gamma_{\mathcal{T}}(T^N(\mathbf{f})(x)) \stackrel{\text{def}}{=} \{(z_0, z_1, \dots, z) \mid z \in Z_N\}$ où

$$\forall i < d, z_i = (T^N(\mathbf{f})(x))_i \text{ et } Z_N = \gamma_{\mathbb{I}} \left((T^N(\mathbf{f})(x))_N \right)$$

Preuve Le produit cartésien de correspondances de Galois forme une correspondance de Galois. \square

Nous notons θ^\sharp l'environnement abstrait qui associe à chaque variable une forme de Taylor centrée intervalle. La sémantique abstraite est notée $\llbracket \cdot \rrbracket_{\mathbb{T}}^\sharp$ et elle est donnée à la figure 5.3. Les opérations abstraites définissant le domaine abstrait $\llbracket \cdot \rrbracket_{\mathbb{T}}^\sharp$ sont alors obtenues en utilisant l'arithmétique d'intervalle uniquement sur le reste des formes de Taylor centrées, nous notons ces opérations $\diamond_{\mathcal{T}}^\sharp$ avec $\diamond_{\mathcal{T}} \in \{+, -, \times, \div\}$. La sémantique des comparaisons est donnée par le résultat de la comparaison $*_{\mathbb{I}}$ dans l'arithmétique d'intervalles. Elle est définie par :

$$\begin{aligned} [a_1, b_1] <_{\mathbb{I}} [a_2, b_2] &= \begin{cases} 1 & \text{si } b_1 < a_2 \\ 0 & \text{sinon} \end{cases} \\ [a_1, b_1] \leq_{\mathbb{I}} [a_2, b_2] &= \begin{cases} 1 & \text{si } b_1 \leq a_2 \\ 0 & \text{sinon} \end{cases} \\ [a_1, b_1] >_{\mathbb{I}} [a_2, b_2] &= \begin{cases} 1 & \text{si } a_1 > b_2 \\ 0 & \text{sinon} \end{cases} \\ [a_1, b_1] \geq_{\mathbb{I}} [a_2, b_2] &= \begin{cases} 1 & \text{si } a_1 \geq b_2 \\ 0 & \text{sinon} \end{cases} \\ [a_1, b_1] =_{\mathbb{I}} [a_2, b_2] &= \begin{cases} 1 & \text{si } a_1 = a_2 \wedge b_1 = b_2 \\ 0 & \text{sinon} \end{cases} \\ [a_1, b_1] \neq_{\mathbb{I}} [a_2, b_2] &= \begin{cases} 1 & \text{si } a_1 \neq a_2 \vee b_1 \neq b_2 \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

$*$ $\in \{<, \leq, >, \geq, =, \neq\}$. La sémantique abstraite d'une expression conditionnelle est sur-approximée en évaluant systématiquement les deux branches de l'expression if.

$$\begin{aligned} \llbracket c \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) &= (c, 0, \dots, 0) \\ \llbracket \ell \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) &= \theta^\sharp(\ell) \\ \llbracket e_1 \diamond e_2 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) &= t_1 \diamond_{\mathcal{T}}^\sharp t_2 \text{ avec } t_1 \in \llbracket e_1 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) \wedge t_2 \in \llbracket e_2 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) \\ \llbracket e_1 * e_2 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) &= i_1 *_{\mathbb{I}} i_2 \text{ avec } i_1 = \mathcal{A}^\sharp(\llbracket e_1 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp)) \wedge i_2 = \mathcal{A}^\sharp(\llbracket e_2 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp)) \\ \llbracket \text{if}(p(e_1), e_2, e_3) \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) &= \llbracket e_2 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) \sqcup_{\mathcal{T}}^\sharp \llbracket e_3 \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp) \end{aligned}$$

FIG. 5.3 – Sémantique abstraite des expressions.

Le théorème 5.4 énonce la sûreté du domaine des formes de Taylor centrées intervalles par rapport au domaine concret des formes de Taylor. Autrement dit, l'ensemble des valeurs obtenues par l'évaluation des formes de Taylor concrètes est contenu dans l'ensemble calculé à partir de formes de Taylor centrées. Nous rappelons que Θ est l'ensemble des environnements concrets.

Théorème 5.4 Pour toute expression arithmétique ou de comparaison e ,

$$(\forall E \subseteq \Theta \wedge \theta^\sharp = \alpha_{\mathcal{T}}(E)) \Rightarrow \bigcup_{\theta \in E} (\llbracket e \rrbracket_{\mathbb{T}}(\theta)) \subseteq \gamma_{\mathcal{T}}(\llbracket e \rrbracket_{\mathbb{T}}^\sharp(\theta^\sharp))$$

Preuve Par induction structurale sur la forme des expressions.

- $e = c$ ou $e = x$, preuve triviale ;
- $e = e_1 \diamond e_2$ avec $\diamond \in \{+, -, \times, \div\}$. Par hypothèse d'induction nous avons :

$$\bigcup_{\theta \in E} \llbracket e_1 \rrbracket_{\mathbb{T}}(\theta) \subseteq \gamma_{\mathcal{T}} \left(\llbracket e_1 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#}) \right) \text{ et } \bigcup_{\theta \in E} \llbracket e_2 \rrbracket_{\mathbb{T}}(\theta) \subseteq \gamma_{\mathcal{T}} \left(\llbracket e_2 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#}) \right)$$

L'arithmétique sur les formes de Taylor centrées utilise les opérations du domaine des intervalles puis réduit le résultat par la fonction \mathbf{m} . La sûreté est induite par celle du domaine des intervalles et le théorème de Taylor-Lagrange ;

- $e = e_1 * e_2$ avec $*$ $\in \{<, \leq, >, \geq, ==, \neq\}$. Par hypothèse d'induction nous avons :

$$\bigcup_{\theta \in E} \llbracket e_1 \rrbracket_{\mathbb{T}}(\theta) \subseteq \gamma_{\mathcal{T}} \left(\llbracket e_1 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#}) \right) \text{ et } \bigcup_{\theta \in E} \llbracket e_2 \rrbracket_{\mathbb{T}}(\theta) \subseteq \gamma_{\mathcal{T}} \left(\llbracket e_2 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#}) \right)$$

et par correction du domaine des intervalles ;

- $e = \text{if}(p(e_1), e_2, e_3)$. Par hypothèse d'induction nous avons :

$$\bigcup_{\theta \in E} \llbracket e_1 \rrbracket_{\mathbb{T}}(\theta) \subseteq \gamma_{\mathcal{T}} \left(\llbracket e_1 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#}) \right)$$

$$\bigcup_{\theta \in E} \llbracket e_2 \rrbracket_{\mathbb{T}}(\theta) \subseteq \gamma_{\mathcal{T}} \left(\llbracket e_2 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#}) \right)$$

$$\bigcup_{\theta \in E} \llbracket e_3 \rrbracket_{\mathbb{T}}(\theta) \subseteq \gamma_{\mathcal{T}} \left(\llbracket e_3 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#}) \right)$$

L'interprétation de $\llbracket e_2 \rrbracket_{\mathbb{T}}^{\#}$ dans l'environnement $\theta^{\#}$ est une sur-approximation des valeurs de $\llbracket e_2 \rrbracket_{\mathbb{T}}$ obtenue en ne considérant que les environnements θ qui vérifient la condition.

L'interprétation de $\llbracket e_3 \rrbracket_{\mathbb{T}}^{\#}$ dans l'environnement $\theta^{\#}$ est une sur-approximation des valeurs de $\llbracket e_3 \rrbracket_{\mathbb{T}}$ obtenue en ne considérant que les environnements θ qui ne vérifient pas la condition.

L'union de $\llbracket e_2 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#})$ et de $\llbracket e_3 \rrbracket_{\mathbb{T}}^{\#}(\theta^{\#})$ est donc clairement une sur-approximation.

□

L'exemple 5.2 montre la précision des résultats d'un calcul utilisant les formes de Taylor par rapport aux résultats obtenus par une arithmétique d'intervalles. Cette précision est essentiellement due à la conservation des relations entre les variables qui sont contenues dans les dérivées.

Exemple 5.2 *Considérons la suite d'instructions précédente (voir exemple 5.1). Nous posons $v_1 = [0, 1]$ et $v_2 = [-1, 0]$.*

Instruction	Intervalle	Valeur centrée	d/dx_1	d/dx_2	Intervalle centré
$x1 = [0, 1];$	$[0, 1]$	0.5	$[1, 1]$	$[0, 0]$	$[0, 1]$
$x2 = [-1, 0];$	$[-1, 0]$	-0.5	$[0, 0]$	$[-1, -1]$	$[-1, 0]$
$y = (x1 + x2) \times 0.5;$	$[-0.5, 0.5]$	0	$[0.5, 0.5]$	$[0.5, 0.5]$	$[-0.5, 0.5]$
$z = x1 - 2 \times y;$	$[-1, 2]$	0.5	$[0, 0]$	$[-1, -1]$	$[-1, 0]$

La seconde colonne donne le résultat de l'évaluation des instructions avec l'arithmétique par intervalles. La dernière colonne donne le résultat intervalle obtenu par une forme de Taylor centrée au premier ordre. Les dérivées d/dx_1 et d/dx_2 sont calculées avec une arithmétique d'intervalles. Nous constatons que la valeur mathématique de z qui est $[-1, 0]$ est obtenue par les formes de Taylor tandis que l'arithmétique d'intervalles engendre un résultat pessimiste $[-1, 2]$.

Le treillis abstrait des formes de Taylor centrées intervalles est de hauteur infinie. Nous définissons alors un opérateur d'élargissement, noté $\nabla_{\mathcal{T}}$, par :

$$T^{N\#}(\mathbf{f}_1)(x) \nabla_{\mathcal{T}} T^{N\#}(\mathbf{f}_2)(x) \stackrel{\text{def}}{=} (z_0^{\#}, z_1^{\#}, \dots, z_N^{\#}) \text{ avec}$$

$$\forall i < N, z_i^{\#} = m \left([(T^{N\#}(\mathbf{f}_1)(x))_i, (T^{N\#}(\mathbf{f}_1)(x))_i] \nabla_{\mathbb{I}} [(T^{N\#}(\mathbf{f}_2)(x))_i, (T^{N\#}(\mathbf{f}_2)(x))_i] \right)$$

$$\text{et } z_N^{\#} = (T^{N\#}(\mathbf{f}_1)(x))_N \nabla_{\mathbb{I}} (T^{N\#}(\mathbf{f}_2)(x))_N.$$

Les $N - 1$ premiers éléments de la forme de Taylor centrée sont considérés comme des intervalles singletons sur lesquels nous appliquons l'opérateur d'élargissement $\nabla_{\mathbb{I}}$, puis nous gardons le centre du résultat. Le dernier élément de la forme de Taylor centrée est obtenu directement par l'application de $\nabla_{\mathbb{I}}$ (voir équation (2.2)).

Nous terminons cette section par l'exemple 5.3 qui compare le domaine des intervalles et celui des formes de Taylor dans les schémas d'intégration numérique. Nous montrons ainsi l'avantage d'utiliser le domaine des formes de Taylor par rapport à celui des intervalles.

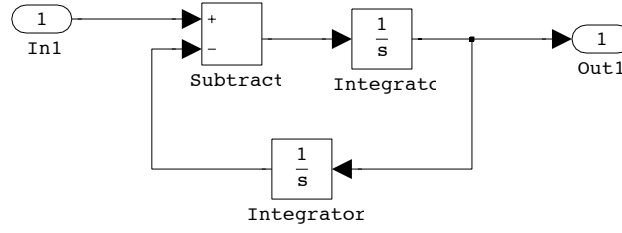
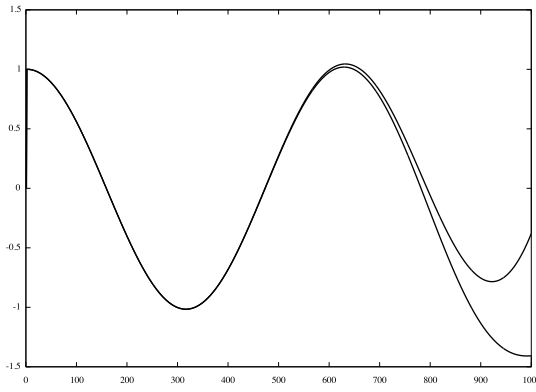
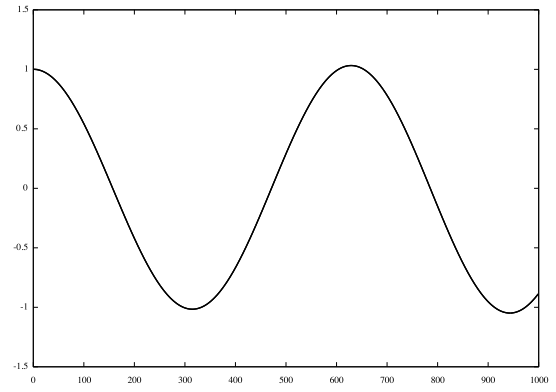


FIG. 5.4 – Modèle Simulink à temps continu calculant la fonction sinus.

Exemple 5.3 Nous prenons un système Simulink à temps continu représentant une équation différentielle d'ordre 2 dont la solution est la fonction sinus. La figure 5.4 donne le schéma en blocs de ce modèle Simulink. L'entrée du modèle est l'intervalle $[0, 0.0001]$ et les valeurs initiales des intégrateurs sont la valeur 1 pour celui qui est à gauche du bloc soustraction et la valeur 0 pour l'autre. La figure 5.5 donne le résultat des simulations utilisant, dans les deux cas, l'algorithme d'Euler comme méthode numérique avec un pas d'intégration de 0.01 et pour une durée de 10 secondes. La simulation numérique utilisant le domaine des intervalles (voir figure 5.5(a)) diverge



(a) Domaines des intervalles.



(b) Domaines des formes de Taylor.

FIG. 5.5 – Résultats des simulations numériques avec le domaine des intervalles et le domaine des formes de Taylor.

un peu avant la sixième seconde alors que celle utilisant le domaine des formes de Taylor reste stable (voir figure 5.5(b)).

5.2 Domaine des nombres flottants avec erreurs différentiées

Dans cette section, nous présentons le second domaine numérique qui sera utilisé dans la définition d'une analyse statique de modèles Simulink. Ce domaine est basé sur une combinaison du domaine des nombres flottants avec erreurs (voir la section 3.2.4) et du domaine des formes de Taylor (voir la section 5.1). Cette composition permet d'améliorer le calcul des termes d'erreurs en prenant en compte les relations entre celles-ci [CM07a, CM08a].

Cette section est organisée comme suit. La section 5.2.1 montre les défaillances du domaine des flottants avec erreurs dans l'estimation des calculs d'erreurs. La section 5.2.2 définit le domaine concret des flottants avec erreurs différentiées et l'abstraction de ce dernier est donnée à la section 5.2.3.

5.2.1 Motivation

Nous introduisons, au travers de l'exemple 5.4, les résultats générés par le domaine abstrait des flottants avec erreurs (voir la section 3.2.4) qui sont parfois peu précis. Le problème est essentiellement induit par les sur-approximations intrinsèques à l'arithmétique d'intervalles. Le programme P donné à l'exemple 5.4 est l'implémentation du calcul de la racine carrée par une méthode de *Newton*. L'algorithme s'appuie sur la récurrence définie par l'équation :

$$x_{n+1} = \frac{x_n}{2} (3 - ax_n^2).$$

Le résultat de P , c'est-à-dire la racine carrée r d'un nombre a , est $r = x_p \times a$ avec x_p la valeur du point fixe de la relation de récurrence. Cet algorithme nécessite une connaissance d'une "bonne valeur" initiale proche de la solution, nous ne considérons pas ce problème ici en supposant cette valeur connue.

Exemple 5.4 Soit le programme P calculant la racine carrée, par une méthode de *Newton*, d'un nombre a strictement positif. a est égal à 25 avec une erreur de 10^{-2} , c'est-à-dire que la valeur réelle est comprise dans l'intervalle $[24.99, 25.01]$.

```
double xn = 0.1;
double xn1 = 0.0;
double a = [25, 25] - [-0.01, 0.01]; // [24.99, 25.01]
int cond = 0;
double temp = 0.0;
double res = 0.0;

while (cond < 1) {
    xn1 = 0.5 * xn * (3.0 - a * xn * xn);
    temp = xn1 - xn;
    if (temp < 1e-12) { cond = 1; }
    if (temp > -1e-12) { cond = 1; }
    xn = xn1;
    res = xn1 * a;
}
```

Nous rappelons que le schéma numérique de la méthode de *Newton* est le suivant :

$$x_{n+1} = x_n - \frac{f(x)}{f'(x)}$$

Dans cet exemple, nous calculons l'inverse de la racine carrée c'est-à-dire que nous définissons la fonction f par

$$f(x) = \frac{1}{x^2} - a.$$

L'avantage de calculer l'inverse de la racine carrée est d'obtenir une récurrence sans division, a priori avec une meilleure stabilité numérique. Au final, la racine carrée est obtenue en multipliant le résultat de la récurrence par la valeur a .

L'utilisation de l'arithmétique d'intervalles pour l'abstraction des parties flottantes et de l'erreur ne permet pas d'apprécier la qualité numérique du résultat de P . En effet, la valeur flottante est $[5.0, 5.0]$ alors que l'évolution de l'intervalle d'erreur grandit. La figure 5.6 montre cette évolution en fonction des itérations. Les lignes pointillées représentent les distances entre la racine carrée de 25.0 et les valeurs des racines carrées de 25.1 et de 24.9, c'est-à-dire les erreurs liées uniquement aux données. Ces erreurs ont été obtenue par un calcul exact des racines carrées. L'évolution de l'erreur conduit à l'idée que la valeur réelle est très différente du résultat flottant.

En arithmétique d'intervalles, il existe deux causes de sur-approximation : la dépendance entre les variables et l'effet enveloppant. Nous présentons dans les paragraphes suivants ces deux phénomènes. Le domaine abstrait des nombres flottants avec erreurs différenciées, autrement dit

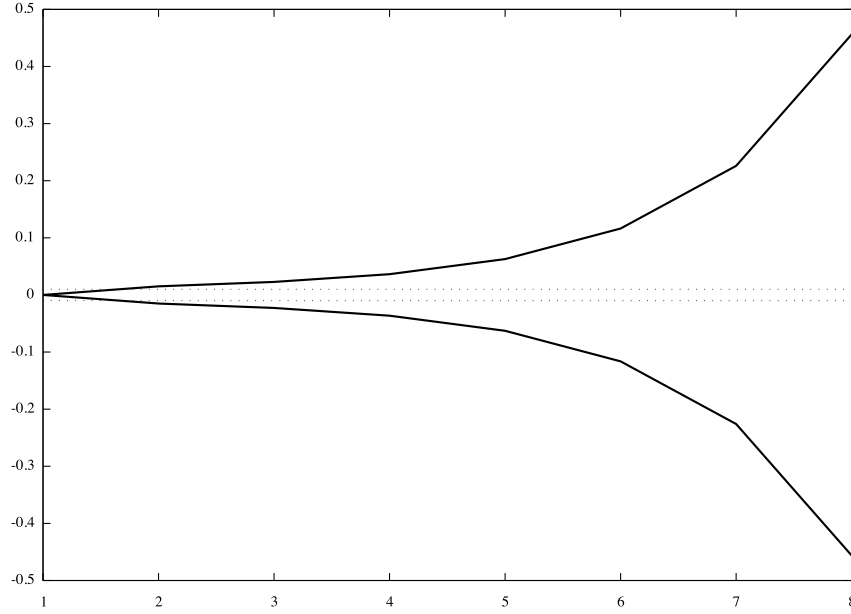


FIG. 5.6 – Evolution de l'intervalle d'erreur en fonction des itérations.

l'utilisation du domaine des formes de Taylor pour le calcul des termes d'erreurs, permet de réduire la sur-approximation induite par la dépendance entre variables. En effet, les formes de Taylor conservent une information sur les variables au sein des différentielles.

La dépendance entre variables. La définition des opérations de l'arithmétique d'intervalles est basée sur des opérations entre ensembles de valeurs. Le schéma général de cette définition est :

$$A \diamond B = \{a \diamond b \mid a \in A \wedge b \in B\},$$

où \diamond est une des quatre opérations arithmétiques $+$, $-$, \times , \div et, A et B sont des ensembles. Cette définition conduit, dans le cas de la soustraction, au résultat suivant, en notant X un intervalle :

$$X - X = \{x_1 - x_2 \mid x_1 \in X \wedge x_2 \in X\},$$

c'est-à-dire que l'arithmétique d'intervalles considère X comme deux ensembles différents. D'où le résultat numérique suivant :

$$[0, 1] - [0, 1] = [-1, 1] \neq [0, 0].$$

L'effet enveloppant. (*Wrapping effect*) La sur-approximation induite par l'effet enveloppant vient de la nature même des intervalles. Plus précisément, l'image d'un pavé, c'est-à-dire un intervalle dans un espace de dimension supérieure ou égale à 2, par une fonction f n'est pas forcément (voire même rarement) un pavé. Cependant, l'arithmétique d'intervalles contraint le résultat de l'évaluation de f à être un pavé de côté parallèle aux axes. Ce pavé a, en général, un volume beaucoup plus important que l'image de la fonction. La figure 5.7 donne un aperçu de la sur-approximation induite par l'effet enveloppant. Nous considérons un pavé centré à l'origine dans un espace à deux dimensions. Nous appliquons successivement deux rotations de $\pi/2$. Le résultat de cette opération devrait être le pavé d'origine mais nous obtenons un résultat tout autre.

La solution la plus souvent adoptée est de subdiviser l'intervalle initial pour ainsi obtenir un ensemble de pavés plus proche de l'image de la fonction [JKDW01].

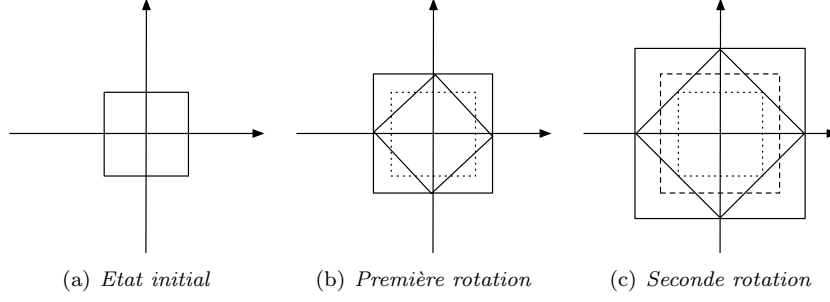


FIG. 5.7 – Résultat d'une double rotation dans l'arithmétique d'intervalles.

5.2.2 Domaine concret

La définition du domaine concret des nombres flottants avec erreurs différentiées, notée $\llbracket \cdot \rrbracket_{\text{ED}}$, se fonde sur le langage des expressions arithmétiques étiquetées. Nous rappelons la grammaire de ce langage :

$$a^\dagger ::= r^\dagger \mid x \mid a_1^{\dagger 1} +^{\dagger 3} a_2^{\dagger 2} \mid a_1^{\dagger 1} -^{\dagger 3} a_2^{\dagger 2} \mid a_1^{\dagger 1} \times^{\dagger 3} a_2^{\dagger 2} \mid a_1^{\dagger 1} \div^{\dagger 3} a_2^{\dagger 2}.$$

Nous partons du constat qu'à chaque point de contrôle \dagger d'une expression arithmétique une nouvelle erreur est introduite. L'erreur introduite au point de contrôle \dagger sera notée o^\dagger . Elle correspond soit à l'erreur de représentation d'une constante, soit à l'erreur d'arrondi du résultat d'une opération. La fonction \mathcal{E} , qui calcule l'erreur globale (voir figure 3.8), est une fonction des o^\dagger . Nous allons alors appliquer la méthode de différentiation automatique à la fonction \mathcal{E} par rapport aux o^\dagger . Les erreurs o^\dagger sont associées aux variables indépendantes de la différentiation automatique.

Dans la sémantique concrète $\llbracket \cdot \rrbracket_{\text{ED}}$, une valeur réelle r est représentée par une paire (f, t) avec f la valeur flottante et t la forme de Taylor associée à la fonction \mathcal{E} calculée par rapport aux variables o^\dagger . La sémantique $\llbracket \cdot \rrbracket_{\text{ED}}$ est définie suivant la structure d'une expression arithmétique a telle que :

$$\llbracket a \rrbracket_{\text{ED}} = (\mathcal{F}(a), T^N(\mathcal{E}(a))),$$

\mathcal{F} est l'implémentation de la norme IEEE 754 (voir figure 3.1). La fonction $T^N(\mathcal{E}(a))$ est la forme de Taylor d'ordre N associée à la fonction \mathcal{E} .

La sémantique d'une constante réelle c au point de contrôle \dagger est donnée par la paire $(\uparrow_o(c), \downarrow_o(c), \frac{\downarrow_o(c)}{\partial o^\dagger}, 0, \dots, 0)$. $\uparrow_o(c)$ est la partie représentable de c dans les flottants, $\downarrow_o(c)$ est l'erreur de représentation et $\frac{\downarrow_o(c)}{\partial o^\dagger}$ est le vecteur de dérivées partielles associé à la différentielle du premier ordre dont toutes les composantes sont nulles sauf la \dagger -ième qui vaut 1. Les différentielles d'ordre supérieur sont nulles. La sémantique des expressions est alors donnée par l'arithmétique de Taylor.

La valeur réelle r associée au nombre flottant avec erreur différentiée (f, t) est égale à :

$$r = f + \mathcal{A}(t),$$

avec \mathcal{A} est la fonction qui évalue une forme de Taylor en une valeur réelle. Nous notons \mathcal{C} la fonction qui associe la valeur flottante f du nombre flottant avec erreurs différentiées (f, t) . Nous notons \mathcal{C}_r la fonction qui associe la valeur réelle associée à (f, t) . Ces deux fonctions sont définies par :

$$\mathcal{C}(f, t) = f \tag{5.3}$$

$$\mathcal{C}_r(f, t) = f + \mathcal{A}(t). \tag{5.4}$$

Dans ce domaine, le calcul des dérivées partielles ne permet qu'une analyse de dépendances puisque le terme d'erreur est calculé dans les réels. Par contre, nous nous servirons de leurs valeurs dans le domaine abstrait pour calculer des ensembles d'erreurs, représentés par des intervalles, mais en réduisant de manière significative la sur-approximation induite par la dépendance des variables.

5.2.3 Domaine abstrait

La sémantique abstraite, notée $\llbracket \cdot \rrbracket_{\mathbb{ED}}^\sharp$, permet, comme la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{E}}^\sharp$, d'évaluer la précision numérique pour des classes d'exécutions.

Dans le domaine abstrait, un ensemble de valeurs réelles R est représenté par une paire $([f], [t])$ avec $[f]$ l'intervalle de flottants et $[t]$ la forme de Taylor centrée. La sémantique abstraite est définie par induction sur la structure d'une expression arithmétique a telle que :

$$\llbracket a \rrbracket_{\mathbb{ED}}^\sharp = (\mathcal{F}^\sharp(a), T^{N^\sharp}(\mathcal{E}(a))).$$

La fonction \mathcal{F}^\sharp est l'extension de la fonction \mathcal{F} à l'arithmétique d'intervalles. La fonction $T^{N^\sharp}(\mathcal{E}(a))$ est la version abstraite de la fonction $T^N(\mathcal{E}(a))$. L'ensemble réel R associé à la valeur abstraite $([f], [t])$ est égal à :

$$R = [f] + \mathcal{A}^\sharp([t]),$$

avec \mathcal{A}^\sharp est la fonction qui évalue une forme de Taylor en une valeur intervalle. Nous notons \mathcal{C}^\sharp la fonction qui associe l'intervalle flottant à la valeur abstraite flottante avec erreurs $([f], [t])$, et nous notons \mathcal{C}_r^\sharp la fonction qui associe l'intervalle réel associé $([f], [t])$. Ces deux fonctions sont définies par :

$$\mathcal{C}([f], [t])^\sharp = [f] \text{ et} \quad (5.5)$$

$$\mathcal{C}_r([f], [t])^\sharp = [f] + \mathcal{A}^\sharp([t]). \quad (5.6)$$

Le théorème 5.5 énonce que le domaine des nombres flottants avec erreurs différentiées forme un treillis complet. Il faut remarquer que nous pouvons donner deux ordres à ce treillis, soit celui qui suit les valeurs flottantes, soit celui qui suit les valeurs réelles. L'ordre donné dans le théorème 5.5 est celui associé aux flottants. Celui associé aux réels est donné par :

$$([f_1], [t_1]) \sqsubseteq_{\mathbb{ED}}^\sharp ([f_2], [t_2]) \Leftrightarrow \mathcal{C}_r^\sharp([f_1], [t_1]) \sqsubseteq_{\mathbb{I}} \mathcal{C}_r^\sharp([f_2], [t_2]).$$

Théorème 5.5 *L'ensemble $\langle \mathcal{ED}^\sharp, \sqsubseteq_{\mathbb{ED}}^\sharp, \sqcup_{\mathbb{ED}}^\sharp, \sqcap_{\mathbb{ED}}^\sharp, \perp_{\mathbb{ED}}, \top_{\mathbb{ED}} \rangle$ forme un treillis complet tel que :*

$$\begin{aligned} ([f_1], [t_1]) \sqsubseteq_{\mathbb{ED}}^\sharp ([f_2], [t_2]) &\Leftrightarrow [f_1] \sqsubseteq_{\mathbb{I}} [f_2] \\ ([f_1], [t_1]) \sqcup_{\mathbb{ED}}^\sharp ([f_2], [t_2]) &= ([f_1] \sqcup_{\mathbb{I}} [f_2], [t_1] \sqcup_{\mathcal{T}}^\sharp [t_2]) \\ ([f_1], [t_1]) \sqcap_{\mathbb{ED}}^\sharp ([f_2], [t_2]) &= ([f_1] \sqcap_{\mathbb{I}} [f_2], [t_1] \sqcap_{\mathcal{T}}^\sharp [t_2]) \end{aligned}$$

Preuve Le produit cartésien des treillis complets $\langle \mathbb{I}, \sqsubseteq_{\mathbb{I}} \rangle$ et $\langle \mathcal{T}^{\sharp}, \sqsubseteq_{\mathcal{T}}^\sharp \rangle$ est un treillis complet. \square

De plus, le théorème 5.6 énonce l'existence d'une correspondance de Galois entre le domaine concret $\langle \wp(\mathcal{ED}), \subseteq \rangle$ et le domaine abstrait $\langle \mathcal{ED}^\sharp, \sqsubseteq_{\mathbb{ED}}^\sharp \rangle$.

Théorème 5.6 *$\langle \wp(\mathcal{ED}), \subseteq \rangle \xleftrightarrow[\alpha_{\mathcal{ED}}]{\gamma_{\mathcal{ED}}} \langle \mathcal{ED}^\sharp, \sqsubseteq_{\mathbb{ED}}^\sharp \rangle$ est une correspondance de Galois.*

Preuve Le produit cartésien des correspondances de Galois $\langle \wp(\mathbb{F}), \subseteq \rangle \xleftrightarrow[\alpha_{\mathbb{I}}]{\gamma_{\mathbb{I}}} \langle \mathbb{I}, \sqsubseteq_{\mathbb{I}} \rangle$ avec \mathbb{F} l'ensemble des nombres flottants et, $\langle \wp(\mathcal{T}^N), \subseteq \rangle \xleftrightarrow[\alpha_{\mathcal{T}}]{\gamma_{\mathcal{T}}} \langle \mathcal{T}^{\sharp}, \sqsubseteq_{\mathcal{T}}^\sharp \rangle$ forme une correspondance de Galois. \square

Le théorème 5.7 montre la correction de la sémantique abstraite par rapport à la sémantique concrète. Nous notons ϕ un environnement qui à chaque variable associe un nombre flottant avec erreurs différentiées et nous notons Φ l'ensemble de ces environnements.

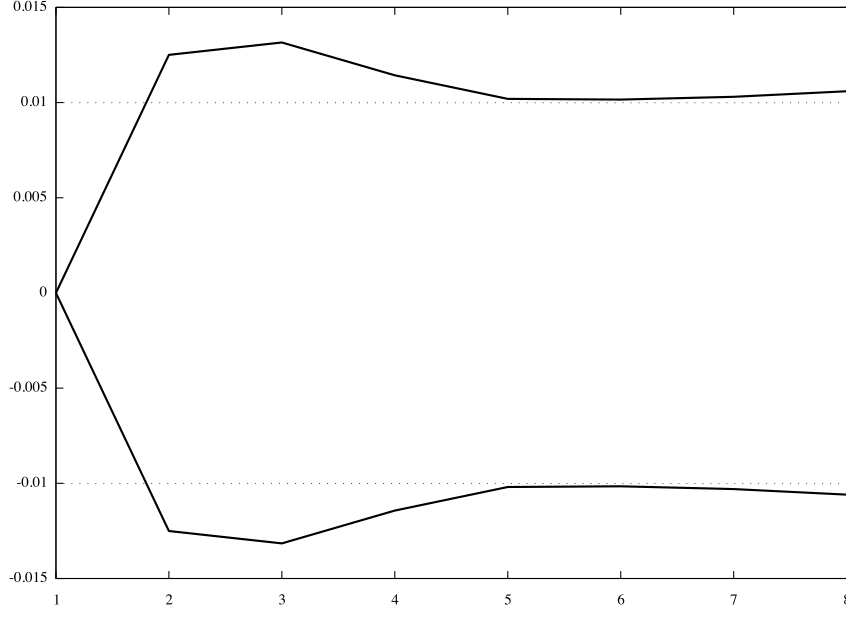


FIG. 5.8 – Evolution de l'intervalle d'erreurs différenciées au second ordre en fonction des itérations.

Théorème 5.7 Pour toute expression arithmétique ou de comparaison e ,

$$(\forall E \subseteq \Phi \wedge \phi^\sharp = \alpha_{\mathcal{ED}}(E)) \Rightarrow \bigcup_{\phi \in E} \llbracket e \rrbracket_{\mathcal{ED}}(\phi) \subseteq \gamma_{\mathcal{ED}}(\llbracket e \rrbracket_{\mathcal{ED}}^\sharp(\phi^\sharp))$$

Preuve Par induction structurale sur la forme des expressions et par la correction du domaine des intervalles et du domaine des formes de Taylor. \square

Exemple 5.5 Reprenons l'exemple 5.4 du calcul de la racine carrée. L'évolution de l'erreur différenciée au second ordre est donnée à la figure 5.8. Nous constatons maintenant une convergence des erreurs ce qui montre une stabilité numérique de l'algorithme, c'est-à-dire l'erreur de méthode est très faible. De plus, nous obtenons un meilleur encadrement des erreurs liées aux données représentées par les lignes pointillées.

5.3 Domaine des séquences

Nous supposons dans cette section que nous connaissons les séquences de valeurs solutions des systèmes d'équations $E \llbracket M \rrbracket$ associées à un modèle Simulink M . Nous définissons deux abstractions des séquences permettant, d'une part, de manipuler des ensembles de comportements et, d'autre part, de gérer les simulations sur un temps non borné.

5.3.1 Abstractions des séquences

Nous modélisons une séquence par une fonction totale du temps discret, c'est-à-dire par une fonction $\mathbf{f} : \mathbb{N} \rightarrow \wp(\mathcal{V})$. Dans la suite de ce chapitre, nous considérons le treillis complet $\langle \mathbb{N} \rightarrow \wp(\mathcal{V}), \preceq_\subseteq \rangle$ dont la relation d'ordre \preceq_\subseteq est définie par :

$$\forall \mathbf{f}_1, \mathbf{f}_2 \in \{\mathbb{N} \rightarrow \wp(\mathcal{V})\}, \mathbf{f}_1 \preceq_\subseteq \mathbf{f}_2 \Leftrightarrow \forall n \in \mathbb{N}, \mathbf{f}_1(n) \subseteq \mathbf{f}_2(n)$$

Il faut remarquer que la propriété de treillis complet associée à $\langle \mathbb{N} \rightarrow \wp(\mathcal{V}), \preceq_{\subseteq} \rangle$ est induite par celle de $\langle \wp(\mathcal{V}), \subseteq \rangle$. S'il existe une correspondance de Galois $(\alpha_{\mathcal{V}}, \gamma_{\mathcal{V}})$ telle que :

$$(\wp(\mathcal{V}), \subseteq) \xrightleftharpoons[\alpha_{\mathcal{V}}]{\gamma_{\mathcal{V}}} (\mathcal{V}^{\sharp}, \sqsubseteq^{\sharp})$$

alors, nous avons le treillis complet des fonctions totales $\{\mathbb{N} \rightarrow \mathcal{V}^{\sharp}\}$ ordonnées par $\preceq_{\sqsubseteq^{\sharp}}$:

$$\forall \mathbf{f}_1, \mathbf{f}_2 \in \{\mathbb{N} \rightarrow \mathcal{V}^{\sharp}\}, \mathbf{f}_1 \preceq_{\sqsubseteq^{\sharp}} \mathbf{f}_2 \Leftrightarrow \forall n \in \mathbb{N}, \mathbf{f}_1(n) \sqsubseteq^{\sharp} \mathbf{f}_2(n)$$

Pour chaque fonction \mathbf{f} dans $\{\mathbb{N} \rightarrow \wp(\mathcal{V})\}$, nous définissons la correspondance de Galois $(\alpha'_{\mathcal{V}}(\mathbf{f}), \gamma'_{\mathcal{V}}(\mathbf{g}))$ telle que :

$$\begin{aligned} \alpha'_{\mathcal{V}}(\mathbf{f}) &= \alpha_{\mathcal{V}} \circ \mathbf{f} \\ \gamma'_{\mathcal{V}}(\mathbf{g}) &= \gamma_{\mathcal{V}} \circ \mathbf{g} \text{ avec } \mathbf{g} = \alpha'_{\mathcal{V}}(\mathbf{f}) \end{aligned} \quad (5.7)$$

A la figure 5.9, nous illustrons cette correspondance de Galois par deux diagrammes montrant la construction des fonctions $\alpha'_{\mathcal{V}}$ et $\gamma'_{\mathcal{V}}$.

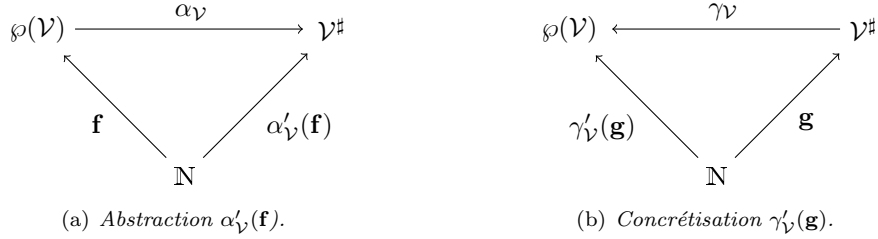


FIG. 5.9 – Diagrammes de $\alpha'_{\mathcal{V}}(\mathbf{f})$ et de $\gamma'_{\mathcal{V}}(\mathbf{g})$.

Proposition 5.8 ([NNH99])

$$\langle \{\mathbb{N} \rightarrow \wp(\mathcal{V})\}, \preceq_{\subseteq} \rangle \xrightleftharpoons[\alpha'_{\mathcal{V}}(\mathbf{f})]{\gamma'_{\mathcal{V}}(\mathbf{g})} \langle \{\mathbb{N} \rightarrow \mathcal{V}^{\sharp}\}, \preceq_{\sqsubseteq^{\sharp}} \rangle$$

est une correspondance de Galois.

Cette première abstraction des séquences permet de manipuler des ensembles de comportements des modèles Simulink. Par exemple, dans le cas de l'utilisation d'une abstraction par des intervalles, nous obtenons :

$$\alpha'_{\mathbb{I}}(\{\sin(u \times k) \mid k \in \mathbb{N}, 1 \leq u \leq 3.5\}) = \sin([1, 3.5] \times k)$$

où k est la variable du temps discret et u est un paramètre réel. Nous pouvons alors considérer une famille de fonction sinus comme une unique valeur abstraite.

La seconde abstraction définie ci-dessous s'appuie sur le partitionnement de l'ensemble \mathbb{N} pour représenter les séquences de longueur infinie par des séquences de longueur finie. Une partition de \mathbb{N} est donnée par une fonction de partition $\mu : \mathbb{N} \rightarrow D$ qui définit une relation d'équivalence \sim_{μ} :

$$\forall n_1, n_2 \in \mathbb{N}, n_1 \sim_{\mu} n_2 \Leftrightarrow \mu(n_1) = \mu(n_2)$$

Nous notons $P_{\mu}(\mathbb{N})$ la partition engendrée par la relation d'équivalence \sim_{μ} , c'est-à-dire :

$$P_{\mu}(\mathbb{N}) = \{\{n \mid n \in \mathbb{N}, n \sim_{\mu} k\} \mid \forall k \in D\}$$

Nous notons $P_{\mu}(\mathbb{N})|_k$ la k -ième partie de l'ensemble $P_{\mu}(\mathbb{N})$, c'est-à-dire : $P_{\mu}(\mathbb{N})|_k = \{n \mid n \in \mathbb{N}, n \sim_{\mu} k\}$. Nous contraignons la fonction μ par l'hypothèse que son co-domaine soit de dimension

finie. Cette contrainte permet de ne manipuler que des séquences abstraites de longueur finie. Nous donnons un exemple d'une telle fonction à l'équation (5.8).

$$\mu_1(k) = \begin{cases} k & \text{if } k \leq p \\ p+1 & \text{sinon} \end{cases} \quad (5.8)$$

Cette fonction sépare les p premiers éléments et regroupe tous les autres dans un seul. Nous représentons ainsi la stratégie d'itérations par dépliage initial des boucles [BCC⁺03]. Un second exemple est la partition périodique de période p décrite à l'équation (5.9).

$$\mu_2(k) = \begin{cases} 0 & \text{si } k \equiv 0 \pmod{p} \\ 1 & \text{si } k \equiv 0 \pmod{p+1} \\ \vdots & \vdots \\ p-1 & \text{si } k \equiv 0 \pmod{2p-1} \end{cases} \quad (5.9)$$

Cette partition permet de regrouper, de façon naturelle, tous les éléments d'une fonction périodique.

Théorème 5.9 $\{\mathbb{N} \rightarrow \mathcal{V}^\sharp, \preceq_\square^\sharp\} \xleftrightarrow[\alpha_\mu]{\gamma_\mu} \{D \rightarrow \mathcal{V}^\sharp, \preceq_\square^\sharp\}$ est une correspondance de Galois. Avec,

$$\alpha_\mu(\mathbf{f}) = \begin{cases} D \rightarrow \mathcal{V}^\sharp \\ k \mapsto \bigsqcup_{n \in P_\mu(\mathbb{N})|_k} \mathbf{f}(n) \end{cases} \quad \text{et} \quad \gamma_\mu(\mathbf{f}^\sharp) = \begin{cases} \mathbb{N} \rightarrow \mathcal{V}^\sharp \\ n \mapsto f^\sharp(k) \text{ tel que } k \sim_\mu n \end{cases}$$

Preuve (α_μ, γ_μ) est une correspondance de Galois entre $\{\mathbb{N} \rightarrow \wp(\mathcal{V}), \preceq_\square^\sharp\}$ et $\{D \rightarrow \mathcal{V}^\sharp, \preceq_\square^\sharp\}$ avec $\mu : \mathbb{N} \rightarrow D$ une fonction de partition où D est de dimension finie.

– α_μ est croissante : $\forall \mathbf{f}_1, \mathbf{f}_2 \in \{\mathbb{N} \rightarrow \mathcal{V}^\sharp, \preceq_\square^\sharp\}$

$$\begin{aligned} \mathbf{f}_1 \preceq_\square^\sharp \mathbf{f}_2 &\Leftrightarrow \forall n \in \mathbb{N}, \mathbf{f}_1(n) \sqsubseteq^\sharp \mathbf{f}_2(n) \\ &\Leftrightarrow \bigsqcup_{n \in \mathbb{N}}^\sharp \mathbf{f}_1(n) \sqsubseteq^\sharp \bigsqcup_{n \in \mathbb{N}}^\sharp \mathbf{f}_2(n) \\ &\Leftrightarrow \bigsqcup_{n \in P_\mu(\mathbb{N})|_1}^\sharp \mathbf{f}_1(n) \sqcup^\sharp \dots \sqcup^\sharp \bigsqcup_{n \in P_\mu(\mathbb{N})|_k}^\sharp \mathbf{f}_1(n) \sqsubseteq^\sharp \bigsqcup_{n \in P_\mu(\mathbb{N})|_1}^\sharp \mathbf{f}_2(n) \sqcup^\sharp \dots \sqcup^\sharp \bigsqcup_{n \in P_\mu(\mathbb{N})|_k}^\sharp \mathbf{f}_2(n) \\ &\Leftrightarrow \forall i \in D, \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\sharp \mathbf{f}_1(n) \sqsubseteq^\sharp \forall i \in D, \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\sharp \mathbf{f}_2(n) \\ &\Leftrightarrow \alpha_\mu(\mathbf{f}_1) \preceq_\square^\sharp \alpha_\mu(\mathbf{f}_2) \end{aligned}$$

– γ_μ est croissante : $\forall \mathbf{f}_1^\sharp, \mathbf{f}_2^\sharp \in \{D \rightarrow \mathcal{V}^\sharp, \preceq_\square^\sharp\}$

$$\begin{aligned} \mathbf{f}_1^\sharp \preceq_\square^\sharp \mathbf{f}_2^\sharp &\Leftrightarrow \forall i \in D, \mathbf{f}_1^\sharp(i) \sqsubseteq^\sharp \mathbf{f}_2^\sharp(i) \\ &\Leftrightarrow \forall i \in D, \forall n \in P_\mu(\mathbb{N})|_i, \mathbf{f}_1(n) = \mathbf{f}_1^\sharp(i) \sqsubseteq^\sharp \mathbf{f}_2(n) = \mathbf{f}_2^\sharp(i) \\ &\Leftrightarrow \forall n \in \mathbb{N} = \bigsqcup_{i \in D}^\sharp P_\mu(\mathbb{N})|_i, \mathbf{f}_1(n) \sqsubseteq^\sharp \mathbf{f}_2(n) \\ &\Leftrightarrow \mathbf{f}_1 \preceq_\square^\sharp \mathbf{f}_2 \end{aligned}$$

– $\forall \mathbf{f} \in \{\mathbb{N} \rightarrow \mathcal{V}^\sharp\}$ et $\forall \mathbf{f}^\sharp \in \{D \rightarrow \mathcal{V}^\sharp\}$

$$- \gamma_\mu(\alpha_\mu(\mathbf{f})) \stackrel{?}{\succeq_{\square^\#}} \mathbf{f}$$

$$\begin{aligned} \gamma_\mu(\alpha_\mu(\mathbf{f})) &\Rightarrow \gamma_\mu \left(\forall i \in D, \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}(n) \right) \\ &\Rightarrow \forall j \in \bigsqcup_{i \in D}^\# P_\mu(\mathbb{N})|_i = \mathbb{N}, \mathbf{g}(j) = \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}(n) \text{ avec } j \sim_\mu i \\ &\Rightarrow \mathbf{g} \succeq_{\square^\#} \mathbf{f} \end{aligned}$$

$$- \alpha_\mu(\gamma_\mu(\mathbf{f}^\#)) \stackrel{?}{\preceq_{\square^\#}} \mathbf{f}^\#$$

$$\begin{aligned} \alpha_\mu(\gamma_\mu(\mathbf{f}^\#)) &\Rightarrow \alpha_\mu (\forall j \in \mathbb{N}, \mathbf{f}(j) = \mathbf{f}^\#(i) \text{ avec } j \sim_\mu i) \\ &\Rightarrow \forall i \in D, g^\#(i) = \bigsqcup_{n \in P_\mu(\mathbb{N})|_i}^\# \mathbf{f}(n) = \mathbf{f}^\#(i) \\ &\Rightarrow \mathbf{g}^\# \preceq_{\square^\#} \mathbf{f}^\# \end{aligned}$$

□

Cette seconde abstraction sur les séquences permet de définir différentes stratégies d'itérations nécessaires pour effectuer une analyse statique.

Exemple 5.6 Soit la séquence $s = v_0, v_1, v_2, \dots, v_n, \dots$. L'utilisation la fonction de partition μ_1 , définie à l'équation (5.8), génère la séquence suivante :

$$s_1 = v_0, v_1, v_2, \dots, \bigcup_{i \geq n} v_i.$$

L'utilisation de la fonction de partition μ_2 , définie à l'équation (5.9), génère la séquence suivante :

$$s_2 = \{v_0, v_p, v_{2p}, \dots\}, \{v_1, v_{p+1}, v_{2p+1}, \dots\}, \dots, \{v_{p-1}, v_{2p-1}, v_{3p-1}, \dots\}.$$

De plus, nous remarquons que l'ensemble des partitions de \mathbb{N} , noté $\mathbb{P}(\mathbb{N})$, forme un treillis complet ordonné par la relation de raffinement \preceq :

$$\forall P, Q \in \mathbb{P}(\mathbb{N}), P \preceq Q \Leftrightarrow \exists q \in Q, q = \bigcup_{p \in S} p \text{ avec } S \subseteq P$$

La conséquence est que nous pouvons utiliser cette relation d'ordre pour comparer la précision de deux analyses statiques : l'analyse a_1 est plus précise que l'analyse a_2 si a_1 a un nombre d'éléments composant la partition plus important que le nombre d'éléments associé à la partition de a_2 .

5.3.2 Sémantique des équations

Nous présentons dans cette section la sémantique associée aux équations d'un modèle Simulink M . La sémantique de Simulink s'appuyant sur le domaine des séquences permet d'expliciter la construction de celles-ci. Les équations apparaissant dans le système d'équations $E[M]$ sont classées en deux catégories :

- les équations permettant de calculer les sorties de M et elles sont de la forme $\ell(k) = e$ et elles constituent l'ensemble d'équations $E^o[M]$;
- les équations associées aux états de M et elles sont de la forme $\ell(k+1) = e$ et elles constituent l'ensemble d'équations $E^s[M]$.

Nous rappelons que k représente la variable du temps discret. Les équations sont un invariant temporel décrivant le signal associé à chaque variable ℓ du modèle M .

La sémantique concrète des équations, notée $\llbracket \cdot \rrbracket_{\mathbb{K}}$, est définie à l'équation (5.10). Elle représente la construction des séquences suivant l'évolution temporelle de k en s'appuyant sur le domaine $\llbracket \cdot \rrbracket_{\mathbb{V}}$. Nous notons $\sigma : \mathbb{V}[\llbracket M \rrbracket] \rightarrow (\mathbb{N} \rightarrow \wp(\mathcal{V}))$ l'environnement qui à chaque variable associe une séquence et Σ représente l'ensemble des environnements. Nous notons σ_i le i -ième élément de la séquence et la notation $\sigma_i[\ell \leftarrow v]$ signifie que la valeur v est affectée au i -ième élément de la séquence associée à la variable ℓ . Nous notons $\sigma_{|i}$ l'environnement formé des éléments en i -ième position des séquences de σ , c'est-à-dire $\sigma_{|i} : \mathbb{V}[\llbracket M \rrbracket] \rightarrow \wp(\mathcal{V})$.

$$\begin{aligned} \llbracket \ell(k) = e \rrbracket_{\mathbb{K}\mathbb{V}}(\sigma) &= \sigma_k[\ell \leftarrow \llbracket e \rrbracket_{\mathbb{V}}(\sigma_{|k})] \\ \llbracket \ell(k+1) = e \rrbracket_{\mathbb{K}\mathbb{V}}(\sigma) &= \sigma_{k+1}[\ell \leftarrow \llbracket e \rrbracket_{\mathbb{V}}(\sigma_{|k})] \end{aligned} \quad (5.10)$$

La sémantique abstraite définie à l'équation (5.11), notée $\llbracket \cdot \rrbracket_{\mathbb{K}\mathbb{V}}^{\#}$, s'appuie sur la fonction de partition μ et sur le domaine abstrait $\llbracket \cdot \rrbracket_{\mathbb{V}}^{\#}$. Nous notons $\sigma_{\mu}^{\#} : \mathbb{V}[\llbracket M \rrbracket] \rightarrow (D \rightarrow \mathcal{V}^{\#})$ l'environnement abstrait qui à chaque variable associe une séquence abstraite, nous notons $\sigma_{\mu,i}^{\#}$ le i -ième élément de la séquence abstraite et nous notons $\sigma_{\mu|i}^{\#} : \mathbb{V}[\llbracket M \rrbracket] \rightarrow \mathcal{V}^{\#}$ l'environnement abstrait formé des i -ième éléments de $\sigma_{\mu}^{\#}$. A chaque instant du temps discret k est associé un instant abstrait $\mu(k)$. A chaque instant abstrait $\mu(k)$ de la séquence abstraite sont collectées toutes les valeurs concrètes des instants k en relation avec $\mu(k)$ par \sim_{μ} .

$$\begin{aligned} \llbracket \ell(k) = e \rrbracket_{\mathbb{K}\mathbb{V}}^{\#}(\sigma_{\mu}^{\#}) &= \sigma_{\mu, \mu(k)}^{\#} \left[\ell \leftarrow \sigma_{\mu, \mu(k)}^{\#}(\ell) \sqcup_{\mathcal{V}}^{\#} \llbracket e \rrbracket_{\mathbb{V}}^{\#}(\sigma_{\mu|\mu(k)}^{\#}) \right] \\ \llbracket \ell(k+1) = e \rrbracket_{\mathbb{K}\mathbb{V}}^{\#}(\sigma_{\mu}^{\#}) &= \sigma_{\mu, \mu(k+1)}^{\#} \left[\ell \leftarrow \sigma_{\mu, \mu(k+1)}^{\#}(\ell) \sqcup_{\mathcal{V}}^{\#} \llbracket e \rrbracket_{\mathbb{V}}^{\#}(\sigma_{\mu|\mu(k)}^{\#}) \right] \end{aligned} \quad (5.11)$$

Le théorème 5.10 montre la correction de la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{K}\mathbb{V}}^{\#}$ au regard de la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{K}\mathbb{V}}$. Pour toute partition μ , l'ensemble des valeurs des séquences concrètes est inclus dans la concrétisation des séquences abstraites. Nous supposons la correction de la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{V}}^{\#}$ par rapport à la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{V}}$ associée.

Théorème 5.10 *Pour toute équation e , pour toute partition μ ,*

$$(\forall E \subseteq \Sigma \wedge \sigma_{\mu}^{\#} = \alpha_{\mu}(E)) \Rightarrow \bigcup_{\sigma \in E} \llbracket e \rrbracket_{\mathbb{K}\mathbb{V}}(\sigma) \subseteq \gamma_{\mu}(\llbracket e \rrbracket_{\mathbb{K}\mathbb{V}}^{\#}(\sigma_{\mu}^{\#}))$$

Preuve Par induction structurelle sur la forme des expressions e et par la correction de la sémantique $\llbracket \cdot \rrbracket_{\mathbb{V}}^{\#}$. \square

Analyse statique des modèles Simulink

Le chapitre précédent a présenté les domaines abstraits nécessaires à la définition de l'analyse statique de modèles Simulink. En particulier, le domaine des formes de Taylor, défini à la section 5.1, permet d'adapter les méthodes d'intégration numérique à l'arithmétique d'intervalles. Le domaine des flottants avec erreurs différenciées, présenté à la section 5.2, rend possible l'évaluation de la distance entre le résultat d'un calcul avec l'arithmétique réelle et le résultat du même calcul avec l'arithmétique flottante. Enfin, le domaine des séquences, défini à la section 5.3, offre un moyen de représenter de manière finie des séquences de longueurs infinies et donc de gérer des simulations sur un temps non borné.

Ce chapitre est organisé comme suit. A la section 6.1, nous présentons l'analyse statique de modèles Simulink et nous donnons un exemple de modèle qui nous servira d'illustration pour la définition de cette analyse. A la section 6.2, nous présentons les modifications de la boucle de simulation permettant de définir une analyse statique de modèles Simulink, c'est-à-dire que nous définissons la sémantique des modèles Simulink pour une itération. A la section 6.3, nous définissons l'analyse statique de modèles Simulink proprement dite en mettant l'accent sur le calcul de point fixe.

6.1 Présentation de l'analyse statique

Dans cette section, nous introduisons informellement l'analyse statique de modèles Simulink, c'est-à-dire la *simulation abstraite*. Nous introduisons également un exemple qui nous servira d'illustration lors de la définition de cette analyse.

L'objectif de la simulation abstraite est de calculer automatiquement un critère de correction des comportements numériques d'un modèle hybride Simulink. Un tel modèle est une description mathématique, c'est-à-dire une spécification, des systèmes embarqués de contrôle. Il décrit l'environnement physique et le logiciel dans un même formalisme. La simulation numérique des modèles Simulink donne une solution approchée des comportements de ceux-ci. Notre critère de correction est donné par une sur-approximation de la distance entre les solutions mathématiques et les solutions fournies par simulation.

Nous décomposons suivant le type de modèles Simulink, à temps continu ou à temps discret, deux façons de calculer le critère de correction. Pour être plus précis, ce critère représente des erreurs de natures différentes suivant le type de modèle. En effet, les erreurs survenant dans la résolution sur ordinateur de problèmes mathématiques ont des origines différentes. Il est fréquent de classer l'origine de ces erreurs [Hig02] de la manière suivante :

- les erreurs de modèles : le modèle mathématique n'est qu'une représentation simplifiée de la réalité ;
- les erreurs de troncature : les méthodes de résolution numériques se basent sur une discrétisation du modèle mathématique (c'est-à-dire l'intégration numérique) ;
- les erreurs de données : les entrées sont des valeurs issues de mesures imparfaites ;
- les erreurs d'arrondi : l'utilisation d'arithmétiques en précision finie induit des résultats approchés.

Nous considérons que les modèles Simulink sont des spécifications de systèmes embarqués de contrôle. Nous ne considérons pas les erreurs de modèles en supposant que le modèle mathématique est correct vis-à-vis de la réalité. Nous prenons comme hypothèse que les modèles à temps discret représentent la partie logicielle de ces systèmes et que les modèles à temps continu sont une description de l'environnement physique. Pour les modèles à temps discret, c'est-à-dire les logiciels embarqués, nous considérons que les approximations sont issues des erreurs d'arrondi. Ce qui signifie que le critère de correction de ces modèles doit fournir la distance entre le résultat mathématique (utilisant une arithmétique réelle) et le résultat de la simulation (utilisant une arithmétique flottante). Ce critère est obtenu en utilisant la sémantique des nombres flottants avec erreurs différenciées sur les modèles à temps discret (voir section 5.2). La sémantique des modèles à temps discret est définie à la section 6.2.1.

L'environnement physique est perçu par le logiciel embarqué par le biais de capteurs. L'activité des capteurs se traduit, d'une part, par un échantillonnage temporel et d'autre part, par une quantification des données. L'échantillonnage consiste à mesurer, en général à une cadence fixe, les valeurs de grandeurs physiques. Il en résulte une discrétisation temporelle des phénomènes physiques. La quantification est la représentation d'une valeur réelle en un codage numérique. Le critère de correction des modèles à temps continu permet de prendre en compte l'erreur introduite par l'échantillonnage. En effet, nous prenons comme hypothèse que l'algorithme numérique d'intégration représente la discrétisation temporelle introduite par l'échantillonnage. Le critère de correction est alors obtenu en calculant la distance séparant les résultats des simulations numériques de ceux obtenus par une intégration numérique garantie. Les algorithmes d'intégration numérique ne donnent qu'un résultat approché, qui peut au fil des calculs s'éloigner du résultat mathématique. Les algorithmes d'intégrations numériques garantis fournissent un encadrement sûr du résultat mathématique. En connaissant ces deux résultats, nous sommes en mesure de donner un critère de correction des comportements numériques des modèles à temps continu. La sémantique des modèles à temps continu est définie à la section 6.2.2. Elle s'appuie sur le domaine des formes de Taylor (voir section 5.1) pour représenter les résultats de la simulation et ceux issus des mathématiques.

Un modèle hybride Simulink est une composition de sous-systèmes de deux natures différentes, c'est-à-dire à temps continu et à temps discret. Ceci se traduit par une utilisation conjointe de deux sémantiques. Le passage d'une sémantique à une autre doit alors être pris en compte. Le passage d'un sous-système à temps continu à un sous-système à temps discret correspond à l'utilisation d'un capteur dans le système embarqué. La fonction **sample** décrite à la section 6.2.3 modélise ce premier changement de sémantique. Nous allons alors utiliser cette fonction pour prendre en considération les erreurs issues de la quantification. Nous prenons ainsi en compte toutes les approximations introduites par un capteur. De plus, le passage d'un sous-système à temps discret à un sous-système à temps continu correspond à l'utilisation d'un actionneur dans le système embarqué. La fonction **activate**, définie à la section 6.2.3, modélise ce second changement de sémantique. Cependant, cette fonction ne génère aucun nouveau type d'erreur.

De plus, la force de la simulation abstraite, par rapport à la simulation numérique, est la prise en compte d'ensemble de comportements. En effet, les spécifications des systèmes embarqués sont, en général, définies pour des conditions particulières d'utilisation. Ce qui signifie que les entrées des systèmes sont contraintes à des domaines de valeurs bien définis. La simulation numérique fournit une seule solution approchée sans information sur la qualité numérique de ce résultat. La validation par simulation nécessite alors d'effectuer une simulation pour toutes les valeurs des entrées ce qui est en général très coûteux et irréalisable en pratique. La simulation abstraite calcule, quant à elle, une sur-approximation d'un ensemble de solutions et elle fournit un critère de corrections évaluant

la qualité numérique de celles-ci. Elle permet ainsi de valider numériquement les modèles Simulink en tenant compte de toutes les approximations numériques pouvant survenir à la simulation.

Nous introduisons maintenant un modèle Simulink qui nous servira d'illustration tout au long de la définition de la sémantique des modèles hybrides Simulink. Le système S est donné

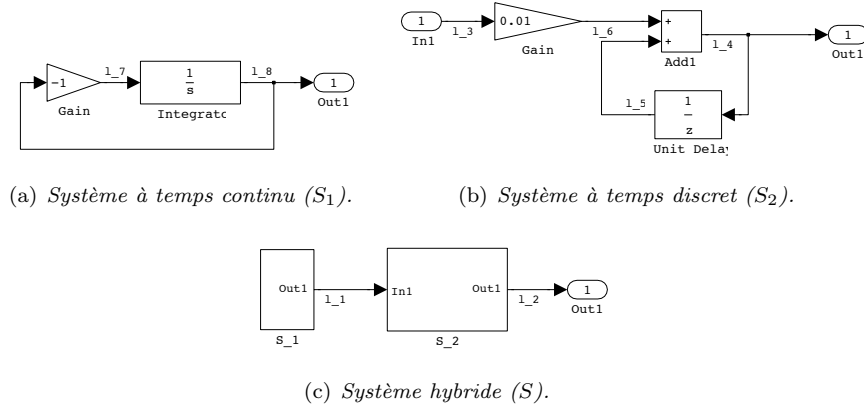


FIG. 6.1 – Exemple simple de modèle hybride Simulink.

à la figure 6.1. Il est composé de deux sous-systèmes S_1 et S_2 . Le sous-système S_1 est donné à la figure 6.1(a) et il représente un système à temps continu sans entrée modélisant l'équation différentielle définie à l'équation (6.1). La solution y de cette équation est une fonction exponentielle décroissante qui prend la valeur 5 en 0, c'est-à-dire $y(t) = 5 \exp(-t)$.

$$\dot{x}(t) = -x(t) \quad \text{avec } x(0) = 5. \quad (6.1)$$

Le second sous-système S_2 est donné à la figure 6.1(b) et il calcule l'intégrale de la fonction d'entrée au cours du temps. Le système complet S , donné à la figure 6.1(c), est une composition des systèmes S_1 et S_2 , ce qui signifie que la sortie de S_1 est l'entrée de S_2 . Le système S est alors un système hybride puisque M_1 est un système à temps continu et M_2 est un système à temps discret. Nous rappelons qu'un modèle Simulink M est représenté, dans notre formalisme, par un système d'équations $E[M]$. Nous rappelons également que $E^o[M]$ et $E^s[M]$ sont respectivement les systèmes d'équations calculant les sorties et les états de M tels que $E[M] = E^o[M] \cup E^s[M]$. Les systèmes d'équations de S , S_1 et S_2 sont :

$$\begin{aligned} E^o[S] &= \left\{ \begin{array}{l} \ell_1 = S_1() \\ \ell_2 = S_2(\ell_1) \\ \text{out1} = \ell_2 \end{array} \right\} & \text{et} & \quad E^s[S] = \emptyset; \\ E^o[S_1] &= \left\{ \begin{array}{l} \ell_7 = -1 \times \ell_8 \\ \ell_8 = x_1(t) \\ \text{out1} = \ell_8 \end{array} \right\} & \text{et} & \quad E^s[S_1] = \{x_1(t) = \ell_7\}; \\ E^o[S_2] &= \left\{ \begin{array}{l} \ell_3 = \text{in1} \\ \ell_6 = 0.01 \times \ell_3 \\ \ell_4 = \ell_6 + \ell_5 \\ \ell_5 = x_2(k) \\ \text{out1} = \ell_4 \end{array} \right\} & \text{et} & \quad E^s[S_2] = \{x_2(k+1) = \ell_4\}. \end{aligned}$$

Les variables t et k représentent respectivement le temps continu et le temps discret. La variable x_1 représente l'état de l'intégration continue (**Integrator**) et la variable x_2 représente l'état du bloc

associé au décalage temporel discret (`UnitDelay`). Ce dernier a pour sortie la précédente valeur de son entrée.

La simulation de ce modèle Simulink avec un pas de temps de 0.01 seconde, sur une durée de 10 secondes et avec une méthode d'intégration numérique suivant l'algorithme d'*Euler*, fournit les résultats donnés à la figure 6.2. La sortie du système S_1 est donnée à la figure 6.2(a) et elle représente une exponentielle décroissante. La sortie du système S_2 est donnée à la figure 6.2(b) et elle montre que la somme tend vers la valeur 5.

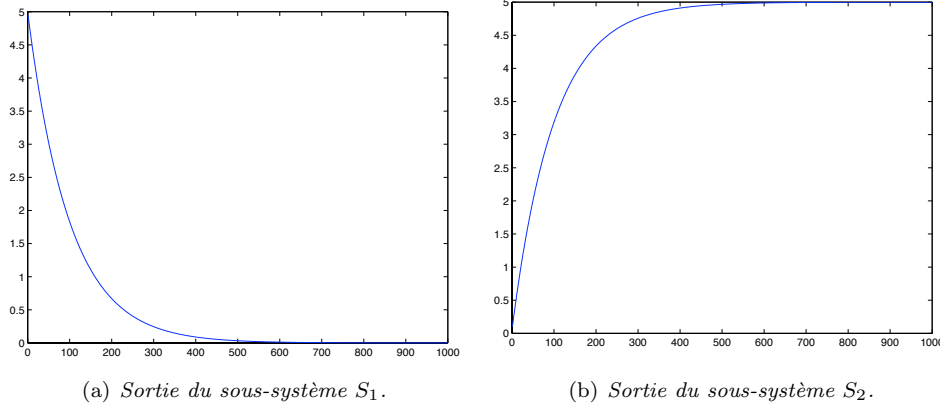


FIG. 6.2 – Résultats de la simulation du système S .

Comme nous l'avons mentionné précédemment, la simulation abstraite permet d'étudier un ensemble de comportements du modèle. Par exemple, l'analyse statique de l'exemple précédent pourrait permettre l'étude des comportements numériques du sous-système à temps continu S_1 pour des valeurs initiales proches de la valeur 5. Nous pouvons ainsi modéliser une incertitude sur la valeur initiale en analysant le système à partir d'un ensemble de valeurs initiales. Il en résulte un ensemble de fonctions, solutions de l'équation différentielle, associée au système S_1 . Les sorties de S_1 sont les entrées du système à temps discret S_2 . Nous pouvons alors étudier les comportements numériques de S_2 suivant l'ensemble des solutions de S_1 en prenant en compte la dynamique des entrées (suivant une fonction exponentielle décroissante). De plus, le passage de S_1 à S_2 nécessite un changement de sémantique, du monde continu au monde discret. Dans ce cas, nous pouvons modéliser un capteur par la fonction `sample` et ainsi analyser l'influence d'une quantification sur les comportements numériques du sous-système S_2 .

Dans la suite de ce chapitre, nous définissons la sémantique des modèles hybrides Simulink. Nous utiliserons des outils classiques de la théorie des langages de programmation tels que le calcul d'un plus petit point fixe. Nous montrons ainsi la force de notre analyse statique qui s'appuie sur des théories établies. Cependant, l'approche proposée est une formalisation originale de la sémantique des modèles hybrides Simulink. D'une part, nous incorporons dans la sémantique la résolution d'équations différentielles et d'autre part, cette sémantique repose sur une interaction étroite de deux sémantiques intermédiaires. Nous proposons ainsi une nouvelle méthode de validation des systèmes hybrides décrits en Simulink.

6.2 Sémantique des modèles pour une itération

A la section 5.3, nous avons défini la sémantique des équations, de la forme $\ell(k) = e$ ou $\ell(k+1) = e$, en utilisant le domaine des séquences. Dans cette section, pour un modèle Simulink M , nous définissons la sémantique de $E[M]$, c'est-à-dire la sémantique d'un système d'équations. Un modèle M est défini dans le langage Simulink par un diagramme en blocs qui décrit une composition d'opérations élémentaires, par exemple une addition ou un décalage temporel. Le système d'équations $E[M]$ représente l'ensemble des opérations élémentaires d'un modèle M . Le

graphe de dépendance associé aux équations de $E[M]$ définit un ordre sur celles-ci. Il permet ainsi d'associer à $E[M]$ une séquence d'opérations S définissant l'ordre d'évaluation des équations (c'est-à-dire des opérations élémentaires). Les équations composant la séquence d'opérations sont ordonnées par une relation de précédence. Elle est donnée, par exemple, par un tri topologique [CLRS01] du graphe de dépendance¹. Par exemple, pour le modèle de la figure 6.1(c), le système d'équations $E^o[S_2]$ est associé à la composition suivante :

$$\ell_3 = \mathbf{in1}; \quad \ell_6 = 0.01 \times \ell_3; \quad \ell_5 = x_2(k); \quad \ell_4 = \ell_6 + \ell_5; \quad \mathbf{out1} = \ell_4. \quad (6.2)$$

Le point-virgule représente l'opération de mise en séquence. L'ordre d'évaluation des équations de $E[M]$ permet de réduire le nombre d'itérations pour l'obtention d'un point fixe nécessaire au calcul de la sémantique (voir section 6.3). Nous notons $I^o[M]$ la séquence d'équations associée au système d'équations $E^o[M]$ et $I^s[M]$ celle associée à $E^s[M]$.

Nous avons également supposé à la section 5.3 que nous connaissions les séquences de valeurs, solutions de $E[M]$. Cependant, la boucle de simulation construit élément par élément les séquences, solutions de $E[M]$. Elle est, par conséquent, un générateur de séquences. En effet, à une itération i de la boucle de simulation, à partir d'un état initial et des entrées du système, la boucle calcule la sortie du modèle, à partir du système d'équations $E^o[M]$, et l'état initial de la prochaine itération, à partir du système d'équations $E^s[M]$. La boucle de simulation se traduit par les trois étapes suivantes :

- i) le calcul des sorties de M , c'est-à-dire les solutions de $E^o[M]$;
- ii) le calcul des états de M , c'est-à-dire les solutions de $E^s[M]$;
- iii) le calcul du prochain pas de temps.

Nous allons alors définir la sémantique des modèles Simulink en suivant le schéma général de cette boucle et, plus précisément, en nous concentrant sur les deux premières étapes. Nous supposons pour simplifier que le pas de temps h est fixe pour toute la durée de l'analyse statique. Nous définissons la sémantique des modèles Simulink en deux temps. Dans cette section, nous considérons une itération i particulière et nous définissons la sémantique d'un modèle sur une itération. Dans la section 6.3, nous décrivons la sémantique d'une succession d'itérations, c'est-à-dire le calcul du point fixe.

Les équations de $I^o[M]$ sont définies par la grammaire décrite à la section 4.2. Nous rappelons que les expressions composant ces équations sont définies par la règle :

$$e ::= r \mid x \mid e_1 \diamond e_2 \mid e_1 * e_2 \mid \text{if } (p_r(e_1), e_2, e_3) \mid \mathbf{f}(\vec{x}_{in}), \quad (6.3)$$

où r est une valeur réelle, x une variable, \diamond une opération arithmétique, $*$ une opération de comparaisons, p_r est un prédicat. L'élément important dans l'équation (6.3) est l'appel de fonctions $\mathbf{f}(\vec{x}_{in})$. Il représente les sous-systèmes des modèles Simulink ainsi que l'aspect hybride de ceux-ci. En effet, par ce biais, un sous-système à temps discret peut appeler un sous-système à temps continu et réciproquement.

L'aspect hybride des modèles Simulink entraîne l'utilisation conjointe de deux sémantiques, la première pour les modèles à temps continu et la seconde pour les modèles à temps discret. Par conséquent, nous devons prendre en compte le passage d'une sémantique à une autre. Cette opération peut être vue comme une conversion explicite de types (*cast*) dans les langages de programmation. Nous définissons, à la section 6.2.3, les fonctions **activate** et **sample** qui permettent ces conversions. Ces fonctions modélisent les capteurs et les actionneurs d'un système embarqué. En effet, les capteurs permettent de mesurer des grandeurs physiques et les actionneurs représentent les commandes qui influencent l'environnement physique. En s'appuyant sur le modèle hybride du système ainsi qu'en utilisant des modèles pour les capteurs et les actionneurs, nous sommes en mesure de spécifier très précisément un système embarqué. Nous sommes également capables d'estimer les imprécisions numériques introduites par tous ces éléments.

¹Si le graphe de dépendance contient un cycle, il est nécessaire d'utiliser d'autres algorithmes comme celui du parcours en profondeur d'abord [NNH99].

Cette section est organisée de la manière suivante. Nous définissons tout d'abord la sémantique des modèles à temps discret et des modèles à temps continu aux sections 6.2.1 et 6.2.2 en supposant qu'ils ne contiennent pas de sous-système. Cette hypothèse permet de présenter les traits importants des sémantiques sans se préoccuper, dans un premier temps, du passage d'une sémantique à une autre. Puis, à la section 6.2.3, nous définissons la fonction **activate** permettant le passage de la sémantique des modèles à temps continu à la sémantique des modèles à temps discret et la fonction **sample** qui permet le passage inverse. Enfin, à la section 6.2.4, nous définissons la sémantique des modèles hybrides en utilisant les précédentes définitions.

6.2.1 Sémantique des modèles à temps discret

Nous définissons la sémantique des modèles à temps discret pour une itération i de la boucle de simulation. Nous supposons que le modèle ne possède pas de sous-système. Cette sémantique est la plus simple des deux sémantiques utilisées pour définir l'analyse statique de modèles hybrides Simulink. En effet, le calcul des états d'un modèle à temps discret ne nécessite pas la mise en œuvre d'un algorithme complexe. Nous notons $(\cdot)_{\text{KED}}$ la sémantique issue de la combinaison du domaine des séquences et du domaine des nombres flottants avec erreurs différenciées et nous notons $(\cdot)_{\mathbb{D}}$ la sémantique des modèles Simulink à temps discret. De plus, nous supposons que l'instant i de l'évaluation est encodé dans les équations. Par exemple, l'équation $\ell(i) = e$ est la i -ième évaluation de l'équation $\ell = e$ ou, autrement dit, l'évaluation de $\ell = e$ au i -ième instant. Nous notons alors $E_i[M]$ le système d'équations de M à l'instant i et nous notons $E_i^o[M]$ et $E_i^s[M]$, les systèmes d'équations décrivant les sorties et les états de M à l'instant i . Par extension, $I_i^o[M]$ et $I_i^s[M]$ décrivent les séquences d'équations (c'est-à-dire l'ordre d'évaluation) à l'instant i des systèmes $E_i^o[M]$ et $E_i^s[M]$.

Soit M un modèle Simulink à temps discret. Nous notons $V[M]$ l'ensemble des variables de M . Nous rappelons que \mathcal{ED} est l'ensemble des nombres flottants avec erreurs différenciées (voir section 5.2) et que $\mathbb{N} \rightarrow \wp(\mathcal{ED})$ représente l'ensemble des séquences (concrètes) de nombres flottants avec erreurs différenciées. Nous notons ϕ l'environnement concret qui, à chaque variable de M , associe une séquence dont les éléments sont un ensemble de nombres flottants avec erreurs différenciées, c'est-à-dire :

$$\phi : V[M] \rightarrow (\mathbb{N} \rightarrow \wp(\mathcal{ED})).$$

Nous notons Φ l'ensemble des environnements concrets. Nous définissons la sémantique d'un modèle Simulink à temps discret à partir de la séquence d'équations associée à $E[M]$. Nous présentons tout d'abord la sémantique d'une composition de deux équations quelconques avant de la définir pour le calcul des sorties et des états. Nous nous appuyons sur la définition classique de la sémantique dénotationnelle de la composition d'instructions dans les langages de programmation impératifs [Win93, section 5.2]. La sémantique concrète $(\cdot)_{\mathbb{D}}$ de la composition de deux équations eq_1 et eq_2 est :

$$(eq_1; eq_2)_{\mathbb{D}}(\phi) = (eq_2)_{\text{KED}}((eq_1)_{\text{KED}}(\phi)).$$

Chaque équation est évaluée dans la sémantique $(\cdot)_{\text{KED}}$. Nous évaluons l'équation eq_2 dans l'environnement résultat de l'évaluation de l'équation eq_1 dans l'environnement ϕ .

Nous rappelons que \mathcal{ED}^\sharp est l'ensemble abstrait des nombres flottants avec erreurs différenciées (voir section 5.2) et que $D \rightarrow \mathcal{ED}^\sharp$ représente l'ensemble des séquences (abstraites) de nombres flottants avec erreurs différenciées, défini par la fonction de partition $\mu : \mathbb{N} \rightarrow D$ (voir section 5.3). Nous notons ϕ_μ^\sharp l'environnement abstrait qui, à chaque variable d'un modèle Simulink M , associe une séquence abstraite utilisant la fonction de partition μ et dont les éléments sont des nombres flottants avec erreurs différenciées abstraits, c'est-à-dire :

$$\phi_\mu^\sharp : V[M] \rightarrow (D \rightarrow \mathcal{ED}^\sharp).$$

Nous notons Φ^\sharp l'ensemble des environnements abstraits. La sémantique abstraite $(\cdot)_{\mathbb{D}}^\sharp$ de la composition de deux équations est obtenue de la même manière que la version concrète par :

$$(eq_1; eq_2)_{\mathbb{D}}^\sharp(\phi_\mu^\sharp) = (eq_2)_{\text{KED}}^\sharp((eq_1)_{\text{KED}}^\sharp(\phi_\mu^\sharp)).$$

La sémantique concrète d'un modèle Simulink à temps discret M à l'itération i est donnée par le calcul des sorties du modèle, suivi du calcul des états. Elle est définie par l'évaluation de la séquence d'équations $I^o[M]$ suivie de celle de la séquence d'équations $I^s[M]$. Autrement dit, nous évaluons la séquence d'équations $I^o[M]; I^s[M]$. Néanmoins, la présence potentielle de boucles dans le graphe de dépendance de $E^o[M]$ induit une définition de la sémantique $\langle \cdot \rangle_{\mathbb{D}}$ comme le calcul d'un plus petit point fixe (lfp). En effet, les boucles de rétroactions dans les modèles Simulink sont présentes, dans notre formalisme, uniquement dans le système d'équations $E^o[M]$. Par exemple, le système S du régulateur du papillon des gaz, décrit à la figure 4.2(a), possède une telle boucle. La sémantique de $E^o[M]$ suit alors la sémantique classique des structures répétitives dans les langages de programmation [Win93, section 5.2]. La sémantique du système d'équations $E_i^o[M]$, à l'instant i , est alors obtenue en calculant le plus petit point fixe, solution de $I_i^o[M]$. Ce calcul a pour objectif de collecter l'ensemble des valeurs pouvant être atteintes pour toutes les itérations de la boucle. La définition de la sémantique $\langle \cdot \rangle_{\mathbb{D}}$ d'un modèle M à temps discret est

$$\langle E[M] \rangle_{\mathbb{D}}(\phi) = \langle I_i^s[M] \rangle_{\mathbb{D}}(\text{lfp}(\langle I_i^o[M] \rangle_{\mathbb{D}}(\phi))).$$

La sémantique abstraite $\langle \cdot \rangle_{\mathbb{D}}^{\#}$ est définie à partir de la même séquence d'équations mais en s'appuyant sur la fonction de partition μ et la sémantique abstraite $\langle \cdot \rangle_{\mathbb{KED}}^{\#}$, c'est-à-dire :

$$\langle E[M] \rangle_{\mathbb{D}}^{\#}(\phi_{\mu}^{\#}) = \langle I_i^s[M] \rangle_{\mathbb{D}}^{\#} \left(\text{lfp} \left(\langle I_i^o[M] \rangle_{\mathbb{D}}^{\#}(\phi_{\mu}^{\#}) \right) \right).$$

La sémantique concrète $\langle \cdot \rangle_{\mathbb{D}}$ est une fonction croissante du treillis complet Φ dans lui-même, c'est-à-dire $\langle \cdot \rangle_{\mathbb{D}} : \Phi \rightarrow \Phi$. La sémantique abstraite $\langle \cdot \rangle_{\mathbb{D}}^{\#}$ est une fonction croissante du treillis complet $\Phi^{\#}$ dans lui-même, c'est-à-dire $\langle \cdot \rangle_{\mathbb{D}}^{\#} : \Phi^{\#} \rightarrow \Phi^{\#}$. Dans les deux cas, par le théorème de Tarski [Win93, section 5.5], nous savons que le plus petit point fixe existe.

L'exemple 6.1 présente une évaluation à l'itération i du système d'équations $E[S_2]$ du système de la figure 6.1(b) dans la sémantique abstraite $\langle \cdot \rangle_{\mathbb{D}}^{\#}$.

Exemple 6.1 Prenons le système S_2 de la figure 6.1(b). Supposons qu'à l'instant i l'environnement $\phi_i^{\#}$ soit défini par :

$$\phi_i^{\#}(\text{in1}) = (f_1, t_1) \text{ et } \phi^{\#}(x_2) = (f_2, t_2).$$

La notation (f, t) représente un nombre flottant avec erreurs différenciées. Pour simplifier les notations, nous ne détaillons pas la différenciation des erreurs. Nous ne considérons les environnements qu'à une itération i donnée, c'est-à-dire que nous ne donnons pas la séquence associée à chaque variable mais uniquement leur valeur du i -ième élément de la séquence. L'évaluation de $E[S_2]$ dans l'environnement $\phi_i^{\#}$ suivant la sémantique $\langle \cdot \rangle_{\mathbb{D}}^{\#}$ conduit à l'environnement suivant :

$$\begin{aligned} \phi_{\mu,i}^{\#}(\text{in1}) &= (f_1, t_1); \\ \phi_{\mu,i}^{\#}(x_2) &= (f_2, t_2); \\ \phi_{\mu,i}^{\#}(l_3) &= (f_1, t_1); \\ \phi_{\mu,i}^{\#}(l_6) &= (0.01 \times f_1, 0.01 \times t_1 + \downarrow(l_6)); \\ \phi_{\mu,i}^{\#}(l_5) &= (f_2, t_2); \\ \phi_{\mu,i}^{\#}(l_4) &= (0.01 \times f_1 + f_2, 0.01 \times t_1 + \downarrow(l_6) + t_2 + \downarrow(l_4)); \\ \phi_{\mu,i}^{\#}(\text{out1}) &= (0.01 \times f_1 + f_2, 0.01 \times t_1 + \downarrow(l_6) + t_2 + \downarrow(l_4)); \\ \phi_{\mu,i+1}^{\#}(x_2) &= (0.01 \times f_1 + f_2, 0.01 \times t_1 + \downarrow(l_6) + t_2 + \downarrow(l_4)). \end{aligned}$$

Nous supposons que la constante 0.01 est représentée exactement en mémoire sinon une erreur pourrait lui être associée. La notation $\downarrow(l_j)$ représente l'erreur d'arrondi de l'évaluation de l'expression associée à la variable l_j . Par exemple, $\downarrow(l_6) = \downarrow(0.01 \times f_1)$. La sortie de S_2 est donnée par la valeur $\phi_{\mu,i}^{\#}(\text{out1})$ et la valeur initiale de la prochaine itération est donnée par $\phi_{\mu,i+1}^{\#}(x_2)$.

Nous rappelons que l'ensemble des environnements concrets Φ forme un treillis complet [NNH99]. En effet, un environnement est une fonction totale des variables dans les valeurs. Si l'ensemble des valeurs forme un treillis complet alors l'environnement forme un treillis complet. L'ordre défini sur un tel treillis est obtenu en appliquant élément par élément la relation d'ordre des valeurs. Autrement dit, la relation d'ordre \sqsubseteq_Φ sur Φ est définie par

$$\forall \phi_1, \phi_2 \in \Phi, \quad \phi_1 \sqsubseteq_\Phi \phi_2 \Leftrightarrow \forall v \in V[M], \quad \phi_1(v) \subseteq \phi_2(v) \quad (6.4)$$

Nous notons \subseteq la relation d'inclusion des ensembles.

Le théorème 6.1 assure la correction de la sémantique abstraite $(\cdot)_\mathbb{D}^\sharp$ vis-à-vis de la sémantique concrète $(\cdot)_\mathbb{D}$ pour une itération i de la boucle de simulation. Il faut noter que la correction de la sémantique abstraite $(\cdot)_\mathbb{D}^\sharp$ au regard de la sémantique concrète $(\cdot)_\mathbb{D}$ est obtenue par la correction de $(\cdot)_\mathbb{KED}^\sharp$ par rapport à $(\cdot)_\mathbb{KED}$.

Théorème 6.1 *Soit M un modèle à temps discret sans sous-système, μ une fonction de partition et $P \subseteq \Phi$ tel que $\phi_\mu^\sharp = \alpha_\Phi(P)$ alors*

$$\bigcup_{\phi \in P} (E_i[M])_\mathbb{D}(\phi) \sqsubseteq_\Phi \gamma_\Phi \left((E_i[M])_\mathbb{D}^\sharp(\phi_\mu^\sharp) \right).$$

Les fonctions d'abstraction α_Φ et de concrétisation γ_Φ des environnements sont définies par :

$$\begin{aligned} \alpha_\Phi(P) &= \phi_\mu^\sharp \text{ tel que } \forall v \in V[M], \quad \phi^\sharp(v) = (\alpha_\mu \circ \alpha'_{\mathcal{ED}})(P(v)) ; \\ \gamma_\Phi(\phi_\mu^\sharp) &= P \text{ tel que } \forall v \in V[M], \quad P(v) = (\gamma'_{\mathcal{ED}} \circ \gamma_\mu)(\phi^\sharp(v)). \end{aligned}$$

Les fonctions $\alpha'_{\mathcal{ED}}$ et $\gamma'_{\mathcal{ED}}$ sont les fonctions d'abstraction et de concrétisation des éléments des séquences définies par l'équation (5.7). Ces éléments sont dans ce cas des nombres flottants avec erreurs différenciées. Les fonctions α_μ et γ_μ sont les fonctions d'abstraction et de concrétisation des séquences définies au théorème 5.9 dont les valeurs sont des nombres flottants avec erreurs différenciées.

Preuve La sémantique abstraite de $E[M]$ est obtenue par l'évaluation en séquence de ces équations. Chaque équation est évaluée dans la sémantique abstraite $(\cdot)_\mathbb{KED}^\sharp$. Par le théorème 5.7 et le théorème 5.10, nous savons que la sémantique abstraite $(\cdot)_\mathbb{KED}^\sharp$ associée à une équation est correcte par rapport à $(\cdot)_\mathbb{KED}$. Nous en déduisons que la sémantique d'une suite d'équations dans cette sémantique est également correcte. La sémantique $(\cdot)_\mathbb{D}^\sharp$ est alors correcte par rapport à la version concrète $(\cdot)_\mathbb{D}$. \square

6.2.2 Sémantique des modèles à temps continu

Nous présentons la sémantique des modèles à temps continu pour une itération i de la boucle de simulation. Nous supposons que le modèle ne possède pas de sous-système. Cette sémantique permet de calculer le résultat d'un ensemble de simulations ainsi que le résultat issu du modèle mathématique que nous appelons également résultat garanti. Contrairement au domaine de nombres flottants avec erreurs différenciées, le domaine des formes de Taylor ne permet pas de représenter la distance entre le résultat mathématique et le résultat de la simulation. Nous associons alors une forme de Taylor à chacun de ces résultats. Comme pour la sémantique $(\cdot)_\mathbb{D}$, nous considérons une évolution implicite du temps discret, c'est-à-dire que nous supposons que l'instant i de l'évaluation est encodé dans les équations. Nous notons alors $E_i[M]$ le système d'équations de M à l'instant i . Nous notons alors $E_i[M]$ le système d'équations de M à l'instant i et nous notons $E_i^o[M]$ et $E_i^s[M]$ les systèmes d'équations décrivant les sorties et les états de M à l'instant i . Par extension, $I_i^o[M]$ et $I_i^s[M]$ décrivent les séquences d'équations (c'est-à-dire l'ordre d'évaluation) à l'instant i des systèmes $E_i^o[M]$ et $E_i^s[M]$ respectivement.

Nous définissons la sémantique $(\cdot)_\mathbb{C}$ des systèmes à temps continu à partir de la sémantique issue de la combinaison du domaine des séquences et du domaine des formes de Taylor que nous

notons $(\cdot)_\mathbb{KT}$. Nous notons $V[M]$ l'ensemble des variables d'un modèle M à temps continu. Nous rappelons que \mathcal{T} représente l'ensemble des formes de Taylor (voir section 5.1). Nous notons $\mathbb{N} \rightarrow (\wp(\mathcal{T}) \times \wp(\mathcal{T}))$ l'ensemble des séquences dont les éléments sont des couples de formes de Taylor. Il faut souligner l'utilisation d'un couple de formes de Taylor pour définir les valeurs des variables. Cette représentation inhabituelle des valeurs dans la sémantique des langages de programmation nous permet de prendre en compte les valeurs issues de la simulation et celles associées aux mathématiques. Soit θ l'environnement concret qui associe, à chaque variable de M , une séquence dont les éléments sont des paires d'ensembles de formes de Taylor, c'est-à-dire :

$$\theta : V[M] \rightarrow (\mathbb{N} \rightarrow (\wp(\mathcal{T}) \times \wp(\mathcal{T}))).$$

Nous notons Θ l'ensemble des environnements concrets. Cette représentation permet de conserver pour chaque variable le résultat de la simulation et le résultat garanti. Nous pouvons ainsi calculer simplement le critère de correction, c'est-à-dire la distance entre les deux résultats. Nous notons θ_{sim} l'environnement qui, à chaque variable de M , associe l'ensemble des séquences issu de la simulation, c'est-à-dire :

$$\theta_{sim} : V[M] \rightarrow (\mathbb{N} \rightarrow \wp(\mathcal{T})).$$

Nous notons θ_{math} l'environnement qui, à chaque variable de M , associe l'ensemble des séquences contenant les solutions mathématiques, c'est-à-dire :

$$\theta_{math} : V[M] \rightarrow (\mathbb{N} \rightarrow \wp(\mathcal{T})).$$

Ces deux notations permettent de définir plus facilement la sémantique des modèles à temps continu. Elles aident à mettre en évidence les particularités de la sémantique des simulations et celle des calculs garantis. Nous notons \uplus l'opération qui fusionne θ_{sim} et θ_{math} en un unique environnement θ . La fonction \uplus est une opération habituelle de création de couples de valeurs. Autrement dit :

$$\begin{aligned} \theta_{sim} \uplus \theta_{math} &= \theta \text{ tel que} \\ \forall v \in V[M], \forall n \in \mathbb{N}, \quad \theta_n(v) &= (\theta_{sim,n}(v), \theta_{math,n}(v)), \end{aligned}$$

où $\theta_n(v)$ représente la n -ième valeur de la séquence de la variable v .

Nous rappelons que \mathcal{T}^\sharp représente l'ensemble des formes de Taylor centrées intervalles (voir section 5.1). Nous notons $\mu : \mathbb{N} \rightarrow D$ une fonction de partition. Nous notons $D \rightarrow (\mathcal{T}^\sharp \times \mathcal{T}^\sharp)$ l'ensemble des séquences abstraites dont les valeurs sont des paires de formes de Taylor abstraites. Soit θ_μ^\sharp l'environnement qui, à chaque variable de M , associe une séquence abstraite dont les éléments sont des paires de formes de Taylor centrées intervalles, c'est-à-dire :

$$\theta_\mu^\sharp : V[M] \rightarrow (D \rightarrow (\mathcal{T}^\sharp \times \mathcal{T}^\sharp)).$$

Nous notons Θ^\sharp l'ensemble des environnements abstraits. Nous notons $\theta_{sim,\mu}^\sharp$ l'environnement abstrait qui, à chaque variable de M , associe la séquence issue de la simulation, c'est-à-dire :

$$\theta_{sim,\mu}^\sharp : V[M] \rightarrow (D \rightarrow \mathcal{T}^\sharp).$$

Nous notons $\theta_{math,\mu}^\sharp$ l'environnement abstrait qui, à chaque variable de M , associe la séquence contenant les formes de Taylor contenant les solutions mathématiques, c'est-à-dire :

$$\theta_{math,\mu}^\sharp : V[M] \rightarrow (D \rightarrow \mathcal{T}^\sharp).$$

La version abstraite de l'opération de fusion \uplus^\sharp est définie de la manière suivante :

$$\begin{aligned} \theta_{sim,\mu}^\sharp \uplus^\sharp \theta_{math,\mu}^\sharp &= \theta_\mu^\sharp \text{ tel que} \\ \forall v \in V[M], \forall n \in D, \quad \theta_{\mu,n}(v) &= (\theta_{sim,\mu,n}(v), \theta_{math,\mu,n}(v)), \end{aligned}$$

où $\theta_{\mu,n}^\sharp(v)$ représente la n -ième valeur abstraite de la séquence abstraite associée à la variable v .

Avant de définir la sémantique $(\cdot)_C$ des modèles à temps continu, nous définissons la sémantique de la composition de deux équations quelconques. Nous nous appuyons, une nouvelle fois, sur la définition classique, en sémantique dénotationnelle, de la composition d'instructions dans les langages de programmation impératifs [Win93, section 5.2]. La sémantique concrète $(\cdot)_C$ d'une séquence de deux équations eq_1 et eq_2 est définie par :

$$\begin{aligned} (eq_1; eq_2)_C(\theta) &= \theta'_{sim} \uplus \theta'_{math} \text{ avec} \\ \theta'_{sim} &= (eq_2)_{KT}((eq_1)_{KT}(\theta_{sim})) \text{ et } \theta'_{math} = (eq_2)_{KT}((eq_1)_{KT}(\theta_{math})). \end{aligned}$$

La sémantique abstraite $(\cdot)_C^\#$ d'une séquence de deux équations est définie de la même manière que la version concrète et à partir de la fonction de partition μ , d'où :

$$\begin{aligned} (eq_1; eq_2)_C^\#(\theta^\#) &= \theta'_{sim, \mu}^\# \uplus \theta'_{math, \mu}^\# \text{ avec} \\ \theta'_{sim, \mu}^\# &= (eq_2)_{KT}^\#((eq_1)_{KT}^\#(\theta_{sim, \mu}^\#)) \text{ et } \theta'_{math, \mu}^\# = (eq_2)_{KT}^\#((eq_1)_{KT}^\#(\theta_{math, \mu}^\#)). \end{aligned}$$

Nous présentons dans la suite de cette section la sémantique des modèles à temps continu en définissant, d'une part, le calcul des sorties et, d'autre part, le calcul des états.

Sémantique concrète et abstraite du calcul des sorties. Soit $I_i^o[M]$ la suite d'équations associée au système d'équations $E_i^o[M]$ d'un modèle Simulink M à l'instant i . La présence potentielle de boucle dans le graphe de dépendance du système d'équations lié aux sorties de M , induit une définition de la sémantique $(\cdot)_C$ comme un calcul de plus petit point fixe (lfp). Nous nous fondons encore une fois sur la sémantique habituelle des structures répétitives dans les langages de programmation [Win93, section 5.2]. La sémantique concrète de $E_i^o[M]$ est alors le calcul du plus petit point fixe solution de $I_i^o[M]$. Elle est définie par :

$$(E_i^o[M])_C(\theta) = \text{lfp}((I_i^o[M])_C(\theta)).$$

La sémantique abstraite fondée sur la fonction de partition μ est, quant à elle, donnée par :

$$(E_i^o[M])_C^\#(\theta_\mu^\#) = \text{lfp}((I_i^o[M])_C^\#(\theta_\mu^\#)).$$

La sémantique concrète $(\cdot)_C$ est une fonction croissante du treillis complet Θ , c'est-à-dire $(\cdot)_C : \Theta \rightarrow \Theta$. La sémantique abstraite $(\cdot)_C^\#$ est une fonction croissante du treillis complet $\Theta^\#$, c'est-à-dire $(\cdot)_C^\# : \Theta^\# \rightarrow \Theta^\#$. Dans les deux cas, par le théorème de Tarski [Win93, section 5.5], nous savons que le plus petit point fixe existe.

Sémantique concrète et abstraite du calcul des états. La différence avec la sémantique des modèles à temps discret est que le calcul des états induit un traitement plus important. En effet, il nécessite l'utilisation d'algorithmes d'intégration numérique, par exemple l'algorithme d'*Euler* explicite (voir section 3.3.2). Nous abordons dans ce paragraphe une des nouveautés de l'analyse statique des modèles Simulink. Plus précisément, la sémantique des modèles à temps continu nécessite une résolution d'équations différentielles. La présence de ce genre d'équations dans les langages de programmation est une nouveauté. L'approche adoptée dans ce contexte est de remplacer ces équations par le schéma numérique d'intégration associé à un algorithme donné. Nous mettons ainsi en évidence les approximations numériques liées aux erreurs de méthodes. Nous définissons une analyse statique paramétrée par l'algorithmique d'intégration numérique et nous obtenons ainsi une méthode générique pour l'analyse statique de modèles Simulink.

Nous rappelons que l'algorithme d'*Euler* utilise la relation de récurrence suivante :

$$x_{n+1} = x_n + h\mathbf{f}(x_n), \quad (6.5)$$

avec h le pas d'intégration et \mathbf{f} est la fonction définissant l'équation différentielle, c'est-à-dire

$$\dot{x}(t) = \mathbf{f}(x). \quad (6.6)$$

Les schémas numériques d'intégration sont tous décrits par des relations de récurrence. Il s'ensuit que ces schémas introduisent une forte dépendance entre les variables. L'utilisation de formes de Taylor permet de prendre en compte ces dépendances. Ces formes définissent un domaine numérique abstrait (voir section 5.1) basé sur les intervalles mais en réduisant l'augmentation de la largeur de ceux-ci. En effet, l'une des deux sources d'imprécision de l'arithmétique d'intervalles est la dépendance entre variables (voir section 5.2) qui est réduite dans le cas des formes de Taylor.

Les équations du système $E^s[M]$ sont de la forme $\dot{\ell} = \ell_1$, où ℓ et ℓ_1 sont des variables. A l'opposé de l'équation (6.6), la fonction décrivant une équation différentielle, dans $E^s[M]$, est alors donnée par un ensemble d'équations appartenant à $E^o[M]$. Dans notre formalisme, le découpage d'un modèle Simulink M par un système d'équations de sorties et un autre pour les états a conduit à une définition en deux parties des équations différentielles. Par exemple, le système S_1 de l'exemple 6.1, dans l'équation $\dot{x}_1(t) = \ell_7$ de $E^s[S_1]$ la fonction composant l'équation différentielle est donnée par les équations $\ell_7 = -1 \times \ell_8$ et $\ell_8 = x_1(t)$ qui appartiennent à $E^o[S_3]$. Par conséquent, le calcul des états d'un modèle à temps continu M est dépendant de l'évaluation de $E_i^o[M]$.

Nous définissons le calcul des états pour des simulations utilisant l'algorithme d'*Euler* explicite. Cet algorithme ne demande qu'une seule évaluation de $E_i^o[M]$ pour être appliqué. Il faut remarquer que l'utilisation d'algorithmes plus complexes d'intégration numérique, par exemple celui de *Rung-Kutta*, nécessite plusieurs évaluations de $E_i^o[M]$. Nous abordons la résolution d'équations différentielles dans la sémantique des modèles à temps continu $(\cdot)_C$. Soit $\dot{\ell} = \ell_1$ une équation du système d'équations $E_i^s[M]$ alors nous la substituons par l'équation $\ell = \ell + h \times \ell_1$. Nous notons $E_{i,Euler}^s[M]$ le nouveau système d'équations où toutes les équations différentielles ont été remplacées par le schéma numérique d'*Euler*. Nous mettons ainsi en évidence les approximations nécessaires à la résolution des équations différentielles, introduites par le moteur de simulations de Simulink. La notation $I_{i,Euler}^s[M]$ représente la séquence d'équations associée à $E_{i,Euler}^s[M]$. La sémantique $(\cdot)_C$ de $E^s[M]$ est alors donnée par :

$$(E_i^s[M])_C(\theta_{sim}) = (I_{i,Euler}^s[M])_C(\theta_{sim}),$$

où chaque équation est interprétée dans la sémantique $(\cdot)_{KT}$. La version abstraite est obtenue directement par l'utilisation de la sémantique abstraite $(\cdot)_{KT}^\#$ et de la fonction de partition μ , ce qui conduit à :

$$(E_i^s[M])_C^\#(\theta_{sim,\mu}) = (I_{i,Euler}^s[M])_C^\#(\theta_{sim,\mu}^\#).$$

L'avantage de l'utilisation des schémas numériques d'intégration est d'obtenir une définition simple de la sémantique du calcul des états du système d'équations $E_i^s[M]$. En effet, nous obtenons une sémantique fondée sur la sémantique de composition des équations.

Nous abordons maintenant le second aspect de la sémantique des modèles à temps continu. Le calcul garanti des états utilise la méthode de Taylor, définie à la section 3.3.3. Nous notons $(\cdot)_G$ la sémantique intermédiaire associée à l'application de cette méthode d'intégration numérique garantie. Nous avons alors la définition de la sémantique concrète suivante :

$$(E_i^s[M])_C(\theta_{math}) = (E_i[M])_G(\theta_{math}).$$

Le calcul garanti des états nécessite la connaissance de l'ensemble des équations d'un modèle à temps continu M , afin de calculer l'encadrement *a priori* de la solution et le développement en série de Taylor (voir section 3.3.3). Il faut noter que les coefficients du développement de Taylor sont décrits à l'aide de formes de Taylor. Pour plus de détails sur la sémantique concrète, le lecteur se référera à [BM08a, section 3] et pour la sémantique abstraite $(\cdot)_G^\#$, il consultera [BM08b]. La sémantique abstraite finale du calcul garanti des états est définie à partir de la sémantique abstraite intermédiaire $(\cdot)_G^\#$ et de la fonction de partition μ par :

$$(E_i^s[M])_C^\#(\theta_{math,\mu}^\#) = (E_i[M])_G^\#(\theta_{math,\mu}^\#).$$

Sémantique concrète et abstraite d'un modèle à temps continu. La définition de la sémantique des modèles à temps continu utilise une représentation non standard des valeurs, c'est-à-dire qu'elle

se fonde sur des couples de valeurs. Ceux-ci représentent les résultats de la simulation et les résultats mathématiques. Cette approche a l'avantage de décomposer les résultats de Simulink et les résultats mathématiques. Nous obtenons ainsi une sémantique qui, d'une part, mime les comportements numériques associés au moteur de simulations et qui, d'autre part, calcule les comportements mathématiques nécessaires à la définition d'un critère de correction. Cette décomposition permet également d'avoir une analyse statique générique suivant les paramètres du moteur de simulation puisque le changement d'algorithme d'intégration numérique n'interfère pas avec celui qui est garanti.

Au final, la sémantique concrète $(\cdot)_C$ d'un modèle Simulink M à temps continu, avec une méthode d'Euler, est définie par :

$$\begin{aligned} (E_i[M])_C(\theta) &= \theta'_{sim} \uplus \theta'_{math} \text{ avec} \\ \theta'_{sim} &= (I_{i,Euler}^s[M])_C((I_i^o[M])_C(\theta_{sim})) \text{ et} \\ \theta'_{math} &= (E_i[M])_G((I_i^o[M])_C(\theta_{math})). \end{aligned}$$

Les valeurs issues de la simulation sont données par θ'_{sim} et les valeurs garanties sont données par θ'_{math} .

La sémantique abstraite $(\cdot)_C^\sharp$ est fondée sur la sémantique abstraite $(\cdot)_{\mathbb{KT}}^\sharp$ et la fonction de partition μ telle que :

$$\begin{aligned} (E_i[M])_C^\sharp(\theta_\mu^\sharp) &= \theta'_{sim,\mu}^\sharp \uplus^\sharp \theta'_{math,\mu}^\sharp \text{ avec} \\ \theta'_{sim,\mu}^\sharp &= (I_{i,Euler}^s[M])_C^\sharp((I_i^o[M])_C^\sharp(\theta_{sim,\mu}^\sharp)) \text{ et} \\ \theta'_{math,\mu}^\sharp &= (E_i[M])_G^\sharp((I_i^o[M])_C^\sharp(\theta_{math,\mu}^\sharp)). \end{aligned}$$

Exemple 6.2 Prenons comme exemple le système S_1 de la figure 6.1(a). Supposons qu'à l'instant i l'environnement θ_i^\sharp soit défini par :

$$\theta_{\mu,i}^\sharp(x_1) = (\mathbf{t}_{sim}^\sharp, \mathbf{t}_{math}^\sharp).$$

L'évaluation de $E[S_1]$ dans l'environnement θ_i^\sharp suivant la sémantique $(\cdot)_C^\sharp$ produit l'environnement suivant :

$$\begin{aligned} \theta_{\mu,i}^\sharp(\ell_8) &= (\mathbf{t}_{sim}^\sharp, \mathbf{t}_{math}^\sharp); \\ \theta_{\mu,i}^\sharp(\ell_7) &= (-1 \times \mathbf{t}_{sim}^\sharp, -1 \times \mathbf{t}_{math}^\sharp); \\ \theta_{\mu,i}^\sharp(out1) &= (\mathbf{t}_{sim}^\sharp, \mathbf{t}_{math}^\sharp); \\ \theta_{\mu,i+1}^\sharp(x_1) &= (\mathbf{t}'_{sim}, \mathbf{t}'_{math}) \text{ tel que} \\ \mathbf{t}'_{sim} &= \mathbf{t}_{sim} + h \times (-1 \times \mathbf{t}_{sim}^\sharp) \text{ et } \mathbf{t}'_{math} = \mathbf{t}_{math} + \sum_{i=0}^{n-1} \frac{h^i}{i!} \frac{d^{(i)}\ell_7}{dt} + \tilde{\mathbf{t}}_{math}. \end{aligned}$$

La valeur $\tilde{\mathbf{t}}_{math}$ représente l'encadrement a priori de la solution de l'équation différentielle. L'expression

$$\sum_{i=0}^{n-1} \frac{h^i}{i!} \frac{d^{(i)}\ell_7}{dt} + \tilde{\mathbf{t}}$$

représente le développement en série de Taylor, associée à l'équation différentielle avec $\frac{d^{(i)}\ell_7}{dt}$ la i -ième dérivée de ℓ_7 par rapport au temps t . Nous obtenons alors comme sortie du système le couple de valeurs $(\mathbf{t}_{sim}^\sharp, \mathbf{t}_{math}^\sharp)$ et la valeur initiale de la prochaine itération est donnée par $\theta_{\mu,i+1}^\sharp(x_1)$.

Nous rappelons que l'ensemble des environnements concrets Θ forme un treillis complet [NNH99]. En effet, un environnement est une fonction totale des variables dans les valeurs. Si

l'ensemble des valeurs forme un treillis complet alors l'environnement forme un treillis complet. L'ordre défini sur un tel treillis est obtenu en appliquant, élément par élément, la relation d'ordre des valeurs. Autrement dit, la relation d'ordre \sqsubseteq_Θ sur Θ est définie par

$$\forall \theta_1, \theta_2 \in \Theta, \quad \theta_1 \sqsubseteq_\Theta \theta_2 \Leftrightarrow \forall v \in V[M], \quad (\mathbf{t}_{1, \text{sim}} \subseteq \mathbf{t}_{2, \text{sim}}) \wedge (\mathbf{t}_{1, \text{math}} \subseteq \mathbf{t}_{2, \text{math}}) \\ \text{tels que } (\mathbf{t}_{1, \text{sim}}, \mathbf{t}_{1, \text{math}}) = \theta_1(v) \text{ et } (\mathbf{t}_{2, \text{sim}}, \mathbf{t}_{2, \text{math}}) = \theta_2(v). \quad (6.7)$$

Nous notons \subseteq la relation d'inclusion des ensembles.

Bien que la sémantique des modèles à temps continu se fonde sur une représentation inhabituelle des valeurs, c'est-à-dire par un couple, nous pouvons énoncer un théorème de correction conforme à la théorie de de l'interprétation abstraite. Le théorème 6.2 assure la correction de la sémantique abstraite $(\cdot)_\mathbb{C}^\sharp$ des modèles à temps continu par rapport à la version concrète $(\cdot)_\mathbb{C}$. La correction de la sémantique abstraite $(\cdot)_\mathbb{C}^\sharp$ au regard de la sémantique concrète $(\cdot)_\mathbb{C}$ est obtenue par la correction de $(\cdot)_\mathbb{KT}^\sharp$ par rapport à $(\cdot)_\mathbb{KT}$ et par la correction de l'algorithme d'intégration numérique garanti. Nous notons θ l'environnement concret qui associe, à chaque variable de M , une séquence dont les éléments sont des paires de formes de Taylor. Nous notons Θ l'ensemble des environnements.

Théorème 6.2 *Soit M un modèle à temps continu sans sous-système, μ une fonction de partition et $E \subseteq \Theta$ tels que $\theta_\mu^\sharp = \alpha_\Theta(E)$ alors*

$$\bigcup_{\theta \in E} (\mathbb{E}[M])_\mathbb{C}(\theta) \sqsubseteq_\Theta \gamma_\Theta \left((\mathbb{E}[M])_\mathbb{C}^\sharp(\theta_\mu^\sharp) \right).$$

Les fonctions d'abstraction α_Θ et de concrétisation γ_Θ des environnements sont définies par :

$$\alpha_\Theta(E) = \theta_\mu^\sharp \text{ tel que } \forall v \in V[M], \quad \theta^\sharp(v) = (\alpha_\mu \circ (\alpha'_T, \alpha'_T))(E(v)) ; \\ \gamma_\Theta(\theta_\mu^\sharp) = E \text{ tel que } \forall v \in V[M], \quad E(v) = ((\gamma'_T, \gamma'_T) \circ \gamma_\mu)(\theta^\sharp(v)).$$

Les fonctions α'_T et γ'_T sont les fonctions d'abstraction et de concrétisation des éléments des séquences définies par l'équation (5.7). Ces éléments sont dans ce cas des formes de Taylor. Les fonctions (α'_T, α'_T) et (γ'_T, γ'_T) forment une correspondance de Galois entre le treillis $\langle \mathbb{N} \rightarrow (\wp(\mathcal{T}) \times \wp(\mathcal{T})), \preceq \rangle$ et le treillis $\langle D \rightarrow \mathcal{T}^\sharp \times \mathcal{T}^\sharp, \preceq \rangle$. Les fonctions α_μ et γ_μ sont les fonctions d'abstraction et de concrétisation des séquences définies au théorème 5.9.

Preuve La sémantique abstraite de $\mathbb{E}[M]$ est obtenue par l'évaluation en séquence des équations de $\mathbb{E}[M]$. Chaque équation est évaluée dans la sémantique abstraite $(\cdot)_\mathbb{KT}^\sharp$. Par le théorème 5.4 et le théorème 5.10, nous savons que la sémantique abstraite $(\cdot)_\mathbb{KT}^\sharp$ associée à une équation est correcte par rapport à $(\cdot)_\mathbb{KT}$. Nous en déduisons que la sémantique d'une suite d'équations dans cette sémantique est également correcte. La sémantique $(\cdot)_\mathbb{C}^\sharp$ est alors correcte par rapport à la version concrète $(\cdot)_\mathbb{C}$. \square

6.2.3 Capteurs et actionneurs

Dans cette section, nous définissons les fonctions **sample** et **activate**. Nous modélisons au travers de ces deux fonctions les capteurs et les actionneurs présents dans les systèmes embarqués. Dans notre analyse, ces fonctions permettent le passage de la sémantique $(\cdot)_\mathbb{C}$ des modèles à temps continu à la sémantique $(\cdot)_\mathbb{D}$ des modèles à temps discret et réciproquement. Ce passage se traduit dans le système d'équations $\mathbb{E}[M]$ d'un modèle Simulink M par la présence d'appels de fonctions. Autrement dit, il se traduit par la présence de sous-systèmes dans le modèle M .

Nous abordons une des nouveautés de l'analyse statique de modèles Simulink. Comme nous l'avons mentionné précédemment, au début de la section 6.2, les fonctions **sample** et **activate** peuvent être associées à des conversions de type (*cast*). Elles permettent de rendre explicites les changements de sémantique. De plus, elles soulignent les différences fondamentales entre les propriétés mathématiques liées aux modèles à temps continu et celles liées aux modèles à temps

discret. Nous pouvons ainsi mettre en exergue les opérations nécessaires pour passer d'un modèle à un autre. Ce qui signifie que nous pouvons introduire dans notre analyse statique des éléments qui sont généralement omis dans l'activité de validation des programmes. Ces éléments sont les capteurs et les actionneurs qui sont des parties importantes des systèmes embarqués et qui contribuent de manière significative aux approximations numériques.

Dans la suite de cette section, nous définissons successivement la fonction **activate** puis la fonction **sample**. L'ensemble de variables \vec{y} représente indifféremment l'ensemble des variables d'entrée d'un modèle ou des variables de sorties d'un modèle. Par exemple, dans l'équation de la forme $\ell = \mathbf{f}(\vec{x}_{in})$, lors de l'appel de la fonction, \vec{y} représente \vec{x}_{in} tandis que lors du retour de la fonction \vec{y} représente $\{\ell\}$. L'ensemble \vec{y} est utilisé dans la définition des fonctions **activate** et **sample**. Il met en évidence les variables qui sont affectées par le changement de sémantique.

Du discret au continu : les actionneurs

Dans cette section, nous définissons la fonction **activate** qui permet le passage de la sémantique $(\cdot)_{\mathbb{D}}$ des modèles à temps discret à la sémantique $(\cdot)_{\mathbb{C}}$ des modèles à temps continu. Elle est définie à l'équation (6.10) pour un instant i donné du temps discret, c'est-à-dire pour une itération i de la boucle de simulation. Nous considérons que chaque passage de la sémantique $(\cdot)_{\mathbb{D}}$ à la sémantique $(\cdot)_{\mathbb{C}}$, modélisé par un appel de fonction, est associé à un actionneur particulier. Nous attachons une étiquette unique b à chaque fonction **activate**. Cette étiquette permet d'identifier la fonction dans un modèle Simulink. De plus, cette étiquette permet de prendre en compte les approximations potentiellement introduites par ces fonctions. Nous notons $\mathbf{activate}_i^b$ l'application de la fonction **activate** au i -ième instant.

La fonction **activate** est associée aux actionneurs dans les systèmes embarqués. Ils permettent aux logiciels embarqués d'agir sur leur environnement d'exécution. Parmi les deux fonctions introduites dans la section 6.2.3, la fonction **activate** est celle qui n'ajoute pas de nouvelles approximations numériques. En effet, nous considérons que le passage d'un monde discret à un monde continu est effectué de manière exacte.

Nous rappelons que \mathcal{ED} est l'ensemble des nombres flottants avec erreurs différenciées et que \mathcal{T} est l'ensemble des formes de Taylor. A partir d'un environnement $\phi : V[M] \rightarrow (\mathbb{N} \rightarrow \wp(\mathcal{ED}))$, qui associe à chaque variable de M un ensemble de séquences de nombres flottants avec erreurs différenciées et un ensemble de variables \vec{y} , la fonction $\mathbf{activate}_i^b$ engendre un environnement $\theta : V[M] \rightarrow (\mathbb{N} \rightarrow (\wp(\mathcal{T}) \times \wp(\mathcal{T})))$ associant aux variables, une séquence dont les éléments sont des paires de formes de Taylor. Le premier élément de la paire représente l'ensemble de valeurs lié aux simulations tandis que le second représente l'ensemble de valeurs lié aux fonctions mathématiques.

Un nombre flottant avec erreurs différenciées est noté (f, t) . A partir d'un tel nombre, la fonction $\mathbf{activate}_i^b$ génère deux formes de Taylor décrivant les valeurs liées à la simulation et aux résultats garantis.

La partie flottante f du nombre flottant avec erreurs différenciées, associée à la variable y dans l'environnement ϕ , est donnée par $\mathcal{C}(\phi(y))$. Elle est interprétée comme une forme de Taylor et elle est associée à l'environnement de simulation θ_{sim} . Nous rappelons que la fonction \mathcal{C} (voir équation (5.3)) est définie par :

$$\mathcal{C}(f, t) = f. \quad (6.8)$$

La valeur réelle du nombre flottant avec erreurs, associée à la variable y dans l'environnement ϕ , est donnée par $\mathcal{C}_r(\phi(y))$. Elle est également interprétée comme une forme de Taylor et elle est associée à l'environnement de valeurs réelles θ_{math} . Nous rappelons que la fonction \mathcal{C}_r (voir équation (5.3)) est définie par :

$$\mathcal{C}_r(f, t) = f + \mathcal{A}(t). \quad (6.9)$$

La fonction \mathcal{A} évalue, en sommant tous les termes, la valeur réelle associée à une forme de Taylor.

Nous notons $\phi_i(y)$ le i -ième élément de la séquence associée à la variable y . La notation $\theta_{sim,i}[y \leftarrow v]$ signifie que la variable y prend la valeur v au i -ième élément de la séquence associée à la simulation. La notation $\theta_{math,i}[y \leftarrow v]$ correspond aux résultats garantis et elle se

traduit de la même manière que $\theta_{sim,i}[y \leftarrow v]$. La version concrète de la fonction **activate** est alors définie par :

$$\begin{aligned} \mathbf{activate}_i(\phi, \vec{y}) &= \theta \quad \text{avec} \\ \forall y \in \vec{y}, \quad \theta(y) &= \theta_{sim}(y) \uplus \theta_{math}(y) \quad \text{tel que} \\ \theta_{sim,i}(y) &= \langle \mathcal{C}(\phi_i(y)) \rangle_{\mathbb{T}} \quad \text{et} \\ \theta_{math,i}(y) &= \langle \mathcal{C}_r(\phi_i(y)) \rangle_{\mathbb{T}} \end{aligned} \quad (6.10)$$

La version abstraite de la fonction **activate**[#], définie à l'équation (6.11), est obtenue directement à partir des versions abstraites des fonctions \mathcal{C} et \mathcal{C}_r , et de la sémantique des formes des Taylor $\langle \cdot \rangle_{\mathbb{T}}$. Elle s'appuie également sur la fonction de partition μ . Posons n l'instant abstrait associé à l'instant i par la fonction μ , c'est-à-dire $n = \mu(i)$. Nous notons $\theta_{sim,\mu,n}^{\#}(y)$ le n -ième élément de la séquence abstraite de la variable y , associé au résultat de la simulation. La notation $\theta_{math,\mu,n}^{\#}(y)$ représente la n -ième valeur du résultat garanti de la séquence de la variable y .

$$\begin{aligned} \mathbf{activate}_i^{\#}(\phi_{\mu}^{\#}, \vec{y}) &= \theta_{\mu}^{\#} \quad \text{avec} \\ \forall y \in \vec{y}, \quad \theta_{\mu,\mu(i)}^{\#}(y) &= \theta_{sim,\mu,\mu(i)}^{\#}(y) \uplus \theta_{math,\mu,\mu(i)}^{\#}(y) \quad \text{tel que} \\ \theta_{sim,\mu,\mu(i)}^{\#}(y) &= \langle \mathcal{C}^{\#}(\phi_{\mu,\mu(i)}^{\#}(y)) \rangle_{\mathbb{T}}^{\#} \quad \text{et} \\ \theta_{math,\mu,\mu(i)}^{\#}(y) &= \langle \mathcal{C}_r^{\#}(\phi_{\mu,\mu(i)}^{\#}(y)) \rangle_{\mathbb{T}}^{\#} \end{aligned} \quad (6.11)$$

L'interprétation d'une valeur intervalle $[a, b]^{\flat}$ en une forme de Taylor suit la relation suivante :

$$[a, b]^{\flat} = \mathbf{m}([a, b]) + [1, 1]^{\flat}([a, b] - \mathbf{m}([a, b])). \quad (6.12)$$

La forme de Taylor est ainsi utilisée pour mesurer l'influence de l'approximation introduite par la fonction **activate**. Le résultat de cette fonction est alors considéré comme une nouvelle variable \flat . L'intervalle $[1, 1]^{\flat}$ représente la dérivée du premier ordre, c'est-à-dire $\frac{\partial \mathbf{activate}^{\flat}}{\partial \flat}$. Cette dérivée représente l'influence de \flat sur le résultat de la fonction **activate**.

Le théorème 6.3 assure la correction de la fonction abstraite **activate**[#] par rapport à sa version concrète **activate**.

Théorème 6.3 *Soit μ une fonction de partition. Soit $P \subseteq \Phi$ un ensemble d'environnements concrets d'un modèle à temps discret tel que $\phi_{\mu}^{\#} = \alpha_{\Phi}(P)$. Soit \vec{y} un ensemble de variables.*

$$\bigcup_{\phi \in P} \mathbf{activate}_i^{\flat}(\phi, \vec{y}) \sqsubseteq_{\Theta} \gamma_{\Theta}(\mathbf{activate}_i^{\#}(\phi_{\mu}^{\#}, \vec{y})).$$

La fonction \sqsubseteq_{Θ} est définie à l'équation (6.7). La fonction α_{Φ} est définie au théorème 6.1 et la fonction γ_{Θ} est définie au théorème 6.2.

Preuve Par la correction du domaine abstrait des formes de Taylor donnée au théorème 5.4. \square

Du continu au discret : les capteurs

Dans cette section, nous définissons la fonction **sample**. Elle permet le passage de la sémantique $\langle \cdot \rangle_{\mathbb{C}}$ des modèles à temps continu à la sémantique $\langle \cdot \rangle_{\mathbb{D}}$ des modèles à temps discret. Cette fonction transforme un couple de formes de Taylor en un nombre flottant avec erreurs différenciées.

La fonction **sample** est associée aux capteurs dans les systèmes embarqués. Ils permettent aux logiciels embarqués de mesurer les grandeurs physiques se trouvant dans leur environnement d'exécution. A la différence de la fonction **activate**, la fonction **sample** introduit des approximations numériques. En particulier, le passage d'un monde continu à un monde discret est décrit par une phase de quantification. Cette phase représente une valeur réelle en un codage en précision finie, ce qui s'accompagne nécessairement d'erreurs d'arrondi. Nous introduisons par le biais de la

fonction **sample**, la prise en compte de tous les éléments d'un système embarqué qui contribue à ajouter des imprécisions numériques. Nous obtenons également une analyse statique plus proche des conditions réelles d'exécution.

Nous notons $\theta : V[M] \rightarrow (\mathbb{N} \rightarrow (\wp(\mathcal{T}) \times \wp(\mathcal{T})))$ l'environnement qui, à chaque variable de M , associe un ensemble de séquences dont les valeurs sont un couple de formes de Taylor (résultat de la simulation et résultat mathématique). Nous notons $\phi : V[M] \rightarrow (\mathbb{N} \rightarrow \wp(\mathcal{ED}))$, l'environnement qui, à chaque variable, associe un ensemble de séquences de nombres flottants avec erreurs différentiées.

La fonction **sample** prend comme argument un environnement θ et un ensemble de variables \vec{y} . La définition de la fonction **sample** est donnée à l'équation (6.13) pour un instant i du temps discret, c'est-à-dire pour une itération i de la boucle de simulation. Nous considérons que chaque passage de la sémantique $(\cdot)_C$ à la sémantique $(\cdot)_D$, modélisé par un appel de fonction, est associé à un capteur particulier. Nous attachons une étiquette unique b à chaque fonction **sample**. Cette étiquette permet d'identifier les capteurs dans un modèle Simulink. De plus, cette étiquette permet de prendre en compte les approximations potentiellement introduites par ces fonctions. Nous notons sample_i^b l'application de la fonction **sample** au i -ième instant. À partir de θ et de \vec{y} , elle génère un environnement ϕ dont les variables sont celles de l'ensemble \vec{y} . À chaque variable y de \vec{y} est associé un nombre flottant avec erreurs différentiées, la valeur flottante est donnée par la valeur de y dans θ_{sim} et l'erreur est obtenue en calculant la différence entre le résultat mathématique $\mathcal{A}(\theta_{math}(y))$ et le résultat de la simulation $\mathcal{A}(\theta_{sim}(y))$. Nous rappelons que la fonction \mathcal{A} évalue une forme de Taylor en une valeur réelle.

De plus, nous modélisons l'opération de quantification introduite dans les capteurs. En effet, un capteur mesure une grandeur physique et en donne une représentation numérique. Cette représentation est, en général, donnée avec une précision inférieure à celle des unités de calculs des systèmes embarqués, par exemple sur 10 bits de précision. La fonction \uparrow_q est notre modélisation de la quantification, c'est-à-dire que nous considérons que le capteur utilise une arithmétique flottante sur q bits pour représenter ces mesures et donc nous arrondissons la partie flottante $\mathcal{A}(\theta_{sim}(y))$ suivant cette précision. Par conséquent, nous prenons en compte l'erreur d'arrondi $\downarrow_q(\mathcal{A}(\theta_{sim}(y)))$ dans la création du nombre flottant avec erreurs. Les fonctions \uparrow_q et \downarrow_q sont définies de la même manière que les opérations d'arrondi de l'arithmétique flottante à la section 3.1, en particulier

$$\downarrow_q(f) = f - \uparrow_q(f).$$

Nous rappelons que $\theta_{*,i}(y)$ représente le i -ième élément de la séquence associée à la variable y , où $*$ représente soit sim , le résultat de la simulation, soit $math$, le résultat mathématique. La notation $\phi_i[y \leftarrow v]$ signifie que la valeur v est affectée au i -ième élément de la séquence de la variable y . La fonction concrète de la fonction **sample** pour une itération i est définie par :

$$\begin{aligned} \text{sample}_i^b(\theta, \vec{y}) &= \phi \quad \text{avec} \\ \forall y \in \vec{y}, \quad \phi_i(y) &= (f, t^b) \quad \text{tel que} \\ f &= \uparrow_q(\mathcal{A}(\theta_{sim,i}(y))) \quad \text{et} \\ t^b &= \llbracket \mathcal{A}(\theta_{math,i}(y)) - \mathcal{A}(\theta_{sim,i}(y)) + \downarrow_q(\mathcal{A}(\theta_{sim,i}(y))) \rrbracket_{\mathbb{T}} \end{aligned} \quad (6.13)$$

Le résultat du calcul $\mathcal{A}(\theta_{math,i}(y)) - \mathcal{A}(\theta_{sim,i}(y)) + \downarrow_q(\mathcal{A}(\theta_{sim,i}(y)))$ est interprété dans la sémantique concrète des formes de Taylor. Nous associons au terme d'erreur t^b une étiquette nommée b mettant ainsi en évidence l'erreur due à la quantification du capteur b . Nous obtenons ainsi un nombre flottant avec erreurs différentiées pour chaque variable de \vec{y} .

La fonction abstraite $\text{sample}_i^{b\#}$ pour une itération i , est définie à l'équation (6.15). Elle est donnée directement par les versions abstraites $\mathcal{A}^\#$ de \mathcal{A} et $\uparrow_q^\#$ de \uparrow_q . L'abstraction $\downarrow_q^\#$ de la fonction \downarrow_q suit la définition de la fonction ulp sur les intervalles (voir l'équation (3.2)) en considérant le mode d'arrondi au plus près. La fonction $\downarrow_q^\#$ est définie par :

$$\downarrow_q^\#([a, b]) = \frac{1}{2}ulp(\max(|a|, |b|)) \times [-1, 1] \quad (6.14)$$

où l'ulp est calculée à partir de la précision q . Posons n l'instant abstrait associé à i par la fonction μ , c'est-à-dire $n = \mu(i)$. Nous rappelons que la notation $\theta_{sim,\mu,n}^\sharp$ représente le n -ième élément de la séquence abstraite associée au résultat de la simulation et que la notation $\theta_{math,\mu,n}^\sharp$ représente la même valeur mais pour le résultat garanti. De plus, $\phi_{\mu,n}^\sharp$ représente la n -ième valeur abstraite de la séquence abstraite dans l'environnement ϕ^\sharp .

$$\begin{aligned} \text{sample}_i^{\flat\sharp}(\theta_\mu^\sharp, \vec{y}) &= \phi_\mu^\sharp \quad \text{avec} \\ \forall y \in \vec{y}, \quad \phi_{\mu,\mu(i)}^\sharp(y) &= (f^\sharp, t^\sharp) \quad \text{tel que} \\ f^\sharp &= \uparrow_q^\sharp \left(\mathcal{A}^\sharp(\theta_{sim,\mu,\mu(i)}^\sharp(y)) \right) \quad \text{et} \\ t^\sharp &= \left\| \mathcal{A}^\sharp(\theta_{math,\mu,\mu(i)}^\sharp(y)) - \mathcal{A}^\sharp(\theta_{sim,\mu,\mu(i)}^\sharp(y)) + \downarrow_q^\sharp(\mathcal{A}^\sharp(\theta_{sim,\mu,\mu(i)}^\sharp(y))) \right\|_{\mathbb{T}}^\sharp \end{aligned} \quad (6.15)$$

La valeur t^\sharp est considérée comme une forme de Taylor. Cette forme est obtenue à partir de la valeur intervalle associée à $\mathcal{A}^\sharp(\theta_{math,\mu,\mu(i)}^\sharp(y)) - \mathcal{A}^\sharp(\theta_{sim,\mu,\mu(i)}^\sharp(y)) + \downarrow_q^\sharp(\mathcal{A}^\sharp(\theta_{sim,\mu,\mu(i)}^\sharp(y)))$ en suivant l'équation (6.12).

Le théorème 6.4 assure la correction de la fonction abstraite **sample**[♯] par rapport à sa version concrète **sample**.

Théorème 6.4 *Soit μ une fonction de partition. Soit $T \subseteq \Theta$ un ensemble d'environnements concrets d'un modèle à temps discret tel que $\theta_\mu^\sharp = \alpha_\Theta(T)$. Soit \vec{y} un ensemble de variables.*

$$\bigcup_{\theta \in T} \text{sample}_i^{\flat}(\theta, \vec{y}) \sqsubseteq_{\Phi} \gamma_{\Phi} \left(\text{sample}_i^{\flat\sharp}(\theta_\mu^\sharp, \vec{y}) \right).$$

La fonction \sqsubseteq_{Φ} est définie à l'équation (6.4). La fonction α_{Θ} est définie au théorème 6.2 et la fonction γ_{Φ} est définie au théorème 6.1.

Preuve Par la correction du domaine abstrait des nombres flottants avec erreurs différenciées donnée au théorème 5.7. \square

6.2.4 Sémantique des modèles hybrides

Dans cette section, nous utilisons les fonctions **sample**, **activate** ainsi que la sémantique $(\cdot)_{\mathbb{D}}$ des modèles à temps discret et la sémantique $(\cdot)_{\mathbb{C}}$ des modèles à temps continu pour définir la sémantique des modèles hybrides. Cette sémantique introduit explicitement les capteurs et les actionneurs dans le modèle Simulink. Nous sommes ainsi capables de valider les comportements numériques d'une spécification d'un système embarqué en prenant en compte tous les éléments introduisant des approximations. Nous notons $(\cdot)_{\mathbb{H}}$ la sémantique des modèles hybrides Simulink et nous la définissons pour une itération i de la boucle de simulation.

La sémantique hybride des modèles Simulink introduit une nouvelle approche de l'analyse statique de programmes. En effet, nous prenons en compte, dans une unique représentation, des éléments aux propriétés mathématiques différentes, la résolution d'équations différentielles, utilisées dans les modèles à temps continu, et l'étude de la précision numérique dans les modèles à temps discret. De plus, l'introduction des capteurs et des actionneurs sont une nouveauté dans le cadre de la validation des logiciels embarqués. Nous considérons ainsi tous les acteurs importants, du point de vue de la qualité numérique, qui entrent dans la composition d'un système embarqué. Ce qui signifie que nous validons les comportements numériques d'un logiciel embarqué en le plongeant dans un modèle d'environnement physique et en le faisant interagir avec ce dernier au travers de capteurs et d'actionneurs. Nous sommes alors en mesure d'étudier toutes les sources potentielles d'approximations numériques. La seconde originalité de notre approche réside dans l'interaction étroite entre deux sémantiques permettant de définir une sémantique de modèles hybrides Simulink. Nous introduisons, dans cette section, les interactions entre différentes sémantiques. Ce qui constitue une nouveauté du point de vue de la sémantique des langages de programmation. En

effet, il est rare de trouver un langage de programmation qui nécessite différentes interprétations de ces constructions syntaxiques. En général, ces différences sont liées uniquement aux types de bases tels que les nombres entiers ou les nombres flottants. Ces types entraînent uniquement différents modes d'évaluation des expressions arithmétiques, mais ils ne changent pas de manière importante la sémantique du langage. Dans notre cas, l'interprétation des modèles à temps continu et celle des modèles à temps discret sont très différentes. Ce qui se traduit, dans la sémantique des modèles hybrides, par une interaction inhabituelle entre deux sémantiques indépendantes.

La sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{H}}$ des modèles hybrides est définie par la combinaison de la sémantique $\llbracket \cdot \rrbracket_{\mathbb{C}}$ des modèles à temps continu et de la sémantique $\llbracket \cdot \rrbracket_{\mathbb{D}}$ des modèles à temps discret. Nous rappelons qu'un modèle M est soit à temps continu, soit à temps discret. L'aspect hybride vient des sous-systèmes de M qui peuvent être à temps continu ou à temps discret. Par exemple, le système de la figure 4.2(a) est un système hybride. La sémantique $\llbracket \cdot \rrbracket_{\mathbb{H}}$ détermine la sémantique à utiliser suivant le type du sous-système. Nous utilisons les règles de conversion suivantes :

- si le modèle M est à temps continu alors nous utilisons la sémantique $\llbracket \cdot \rrbracket_{\mathbb{C}}$ et
 - si le modèle M' est à temps continu alors nous continuons avec la sémantique $\llbracket \cdot \rrbracket_{\mathbb{C}}$;
 - si le modèle M' est à temps discret alors nous choisissons la sémantique $\llbracket \cdot \rrbracket_{\mathbb{D}}$. Ce cas de figure correspond à l'utilisation d'un capteur dans un système embarqué. Nous utilisons alors la fonction **sample** ;
- si le modèle M est à temps discret alors nous utilisons la sémantique $\llbracket \cdot \rrbracket_{\mathbb{D}}$ et
 - si le modèle M' est à temps continu alors nous optons pour la sémantique $\llbracket \cdot \rrbracket_{\mathbb{C}}$. Ce cas de figure correspond à l'utilisation d'un actionneur dans un système embarqué. Nous utilisons, dans ce cas, la fonction **activate** ;
 - si le modèle M' est à temps discret alors nous continuons avec la sémantique $\llbracket \cdot \rrbracket_{\mathbb{D}}$.

Nous supposons que tous les systèmes d'un modèle Simulink sont typés. Ce qui signifie qu'à chaque sous-système d'un modèle Simulink M est associée une information stipulant sa nature, à temps continu ou à temps discret.

Les versions concrètes et abstraites de la sémantique des modèles hybrides sont sensiblement les mêmes. Nous détaillons uniquement la version abstraite $\llbracket \cdot \rrbracket_{\mathbb{H}}^{\#}$ de la sémantique des modèles hybrides pour une itération i . Chaque sous-système possède son propre environnement dépendant de son type. Soit μ une fonction de partition. Nous notons $\sigma_{\mu}^{\#}$ l'environnement représentant soit un environnement d'un modèle à temps continu $\theta_{\mu}^{\#}$, soit un environnement d'un modèle à temps discret $\phi_{\mu}^{\#}$. La définition de la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{H}}^{\#}$ suit les sémantiques $\llbracket \cdot \rrbracket_{\mathbb{C}}^{\#}$ et $\llbracket \cdot \rrbracket_{\mathbb{D}}^{\#}$. Nous définissons la sémantique du système d'équations des sorties $E_i^o[M]$ celle du système d'équations des états $E_i^s[M]$.

L'aspect hybride des modèles Simulink se situe, dans notre formalisme, dans le système d'équations $E^o[M]$. En effet, la présence de sous-systèmes dans le modèle est traduite par des équations de la forme $\ell = \mathbf{f}(\vec{x}_{in})$. Nous définissons la sémantique du système d'équations $E_i^o[M]$ associées aux sorties du modèle M en considérant la présence d'appels de fonctions. Pour des raisons de lisibilité, nous présentons l'évaluation de la séquence d'équations $I_i^o[M]$ sous la forme d'un algorithme. En effet, la sémantique du système d'équations $E_i^o[M]$ d'un modèle hybride M est définie suivant la forme des équations et elle dépend du type de M . L'ensemble des cas à étudier est alors plus aisément mis en avant sous la forme d'un algorithme. L'algorithme 1 décrit l'évaluation de la séquence d'équations $I_i^o[M]$. La séquence évalue chaque équation une à une. Suivant le type du modèle M , à temps continu ou à temps discret, les équations sont évaluées dans la sémantique des modèles à temps continu $\llbracket \cdot \rrbracket_{\mathbb{C}}^{\#}$ ou dans la sémantique des modèles à temps discret $\llbracket \cdot \rrbracket_{\mathbb{D}}^{\#}$. Si l'équation contient un appel de fonction les règles de conversion précédemment définies sont appliquées.

La sémantique abstraite du système d'équations $E_i^o[M]$ se fonde sur l'évaluation de la séquence d'équations $I_i^o[M]$ décrite par l'algorithme 1. Cependant, la présence potentielle de boucle dans la graphe de dépendance de $E_i^o[M]$ nécessite le calcul d'un plus petit point fixe. Nous avons alors la définition suivante :

$$\llbracket E_i^o[M] \rrbracket_{\mathbb{H}}^{\#}(\sigma_{\mu}^{\#}) = \text{lfp} \left(\llbracket I_i^o[M] \rrbracket_{\mathbb{H}}^{\#}(\sigma_{\mu}^{\#}) \right). \quad (6.16)$$

La sémantique abstraite du système d'équations des états $E_i^s[M]$ du modèle M est définie en

choisissant la sémantique associée au type de M . Nous avons alors la définition suivante :

$$\langle E_i^s[M] \rangle_{\mathbb{H}}^{\#}(\sigma_{\mu}^{\#}) = \begin{cases} \langle E_i^s[M] \rangle_{\mathbb{C}}^{\#}(\sigma_{\mu}^{\#}) & \text{si } M \text{ est à temps continu} \\ \langle E_i^s[M] \rangle_{\mathbb{D}}^{\#}(\sigma_{\mu}^{\#}) & \text{si } M \text{ est à temps discret} \end{cases} \quad (6.17)$$

Au final la sémantique abstraite $\langle \cdot \rangle_{\mathbb{H}}^{\#}$ d'un modèle hybride M pour l'itération i est donnée par :

$$\langle E_i[M] \rangle_{\mathbb{H}}^{\#}(\sigma_{\mu}^{\#}) = \langle E_i^s[M] \rangle_{\mathbb{H}}^{\#} \left(\langle E_i^o[M] \rangle_{\mathbb{H}}^{\#}(\sigma_{\mu}^{\#}) \right). \quad (6.18)$$

Algorithme 1 Evaluation de la séquence d'équations $I_i^o[M]$.

Entrée(s): $I_i^o[M]$ une séquence d'équations, $\sigma_{\mu}^{\#}$ un environnement abstrait.

Sortie(s): $\sigma_{\mu}^{\#}$ environnement contenant les sorties de M .

```

pour toute équation  $e \in I_i^o[M]$  faire
  si  $M$  continu alors
    si  $e$  est de la forme  $\ell = \mathbf{f}(\vec{x}_{in})$  alors
      si  $\mathbf{f} = E[M']$  est à temps continu alors
         $\sigma^{\#} = \langle E[M'] \rangle_{\mathbb{C}}^{\#}(\sigma^{\#})$ 
      sinon {  $\mathbf{f} = E[M']$  est à temps discret }
         $\phi^{\#} = \text{sample}^{\#}(\sigma^{\#}, \vec{x}_{in})$ 
         $\sigma^{\#} = \text{activate}^{\#}(\langle E[M'] \rangle_{\mathbb{D}}^{\#}(\phi^{\#}), \ell)$ 
      fin si
    sinon {  $e$  est de la forme  $\ell = \text{exp}$  }
       $\sigma^{\#} = \langle e \rangle_{\mathbb{KT}}^{\#}(\sigma^{\#})$ 
    fin si
  sinon {  $M$  est discret }
    si  $e$  est de la forme  $\ell = \mathbf{f}(\vec{x}_{in})$  alors
      si  $\mathbf{f} = E[M']$  est à temps continu alors
         $\theta^{\#} = \text{activate}^{\#}(\sigma^{\#}, \vec{x}_{in})$ 
         $\sigma^{\#} = \text{sample}^{\#}(\langle E[M'] \rangle_{\mathbb{C}}^{\#}(\theta^{\#}), \ell)$ 
      sinon {  $E[M']$  est à temps discret }
         $\sigma^{\#} = \langle E[M'] \rangle_{\mathbb{D}}^{\#}(\sigma^{\#})$ 
      fin si
    sinon {  $e$  est de la forme  $\ell = \text{exp}$  }
       $\sigma^{\#} = \langle e \rangle_{\mathbb{KED}}^{\#}(\sigma^{\#})$ 
    fin si
  fin si
fin pour

```

La correction de la sémantique abstraite $\langle \cdot \rangle_{\mathbb{H}}^{\#}$ par rapport à la sémantique concrète $\langle \cdot \rangle_{\mathbb{H}}$ est obtenue par les théorèmes de correction de la sémantique des modèles à temps discret (voir théorème 6.1), de la sémantique des modèles à temps continu (voir théorème 6.2), de la fonction **activate** (voir théorème 6.3) et de la fonction **sample** (voir théorème 6.4).

6.3 Analyse statique

Dans cette section, nous définissons l'analyse statique de modèles hybrides Simulink, c'est-à-dire la *simulation abstraite* de modèles Simulink. Nous avons présenté à la section 6.2 la sémantique des modèles hybrides Simulink pour une itération i de la boucle de simulation. Nous décrivons maintenant la sémantique des modèles Simulink, notée $\langle \cdot \rangle_{\mathbb{A}}$, en prenant en compte l'évolution temporelle.

Cette section est organisée de la manière suivante. Dans un premier temps, nous définissons la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{A}}$ et la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{A}}^{\#}$. Dans un second temps, nous traduisons la sémantique abstraite en une boucle de simulation abstraite afin de mettre en évidence la relation étroite avec la boucle de simulation de Simulink.

Sémantique des modèles Simulink. Nous représentons l'évolution du temps discret par une équation particulière $k = k + 1$ avec $k = 0$ pour le premier instant. L'interprétation de cette équation est donnée par la sémantique $\llbracket \cdot \rrbracket_{\mathbb{D}}$ des modèles à temps discret en supposant les calculs exacts. Pour chaque instant k , nous calculons la sémantique du modèle grâce à l'application de sémantique $\llbracket \cdot \rrbracket_{\mathbb{H}}$ sur le système d'équations $E_k[M]$. Nous rappelons que l'environnement σ représente soit un environnement θ pour les modèles à temps continu, soit un environnement ϕ pour les modèles à temps discret. La sémantique des modèles hybrides, notée $\llbracket \cdot \rrbracket_{\mathbb{A}}$, pour tous les instants, est obtenue en calculant un plus petit point fixe. Ce qui signifie que la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{A}}$ est définie en utilisant deux fonctions auxiliaires par :

$$\begin{aligned} \llbracket E[M] \rrbracket_{\mathbb{A}}(\sigma) &= G(\sigma) \text{ avec} \\ G &= \lambda e. \text{ lfp } ((k = k + 1)_{\mathbb{D}}(F(e, k))) \quad \text{et} \quad F = \lambda e. \lambda k. \llbracket E_k[M] \rrbracket_{\mathbb{H}}(e). \end{aligned} \quad (6.19)$$

Les fonctions F et G sont décrites à l'aide de la notation du lambda calcul. La fonction F représente l'application de la sémantique $\llbracket \cdot \rrbracket_{\mathbb{H}}$ pour un instant i et l'environnement e donné. La fonction G décrit le calcul du plus petit point fixe induit par l'évolution temporelle à partir d'un environnement e donné. Bien que l'existence du plus petit point fixe soit donnée par le théorème de Tarski, la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{A}}$ n'est pas calculable. En effet, elle calcule les comportements d'un modèle Simulink pour un temps infini. Nous mettons ainsi en évidence que la boucle de simulation de Simulink (voir section 4.3) ne donne qu'une représentation partielle des comportements des modèles.

Nous notons μ la fonction de partition. Nous rappelons que l'environnement abstrait $\sigma_{\mu}^{\#}$ représente soit un environnement abstrait de modèles à temps continu $\theta_{\mu}^{\#}$ soit un environnement abstrait de modèles temps discret $\phi_{\mu}^{\#}$. La sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{A}}^{\#}$ est définie de la même manière que la sémantique concrète par :

$$\begin{aligned} \llbracket E[M] \rrbracket_{\mathbb{A}}^{\#}(\sigma_{\mu}^{\#}) &= G^{\#}(\sigma_{\mu}^{\#}) \text{ avec} \\ G^{\#} &= \lambda e. \text{ lfp } ((k = k + 1)_{\mathbb{D}}^{\#}(F^{\#}(e, k))) \quad \text{et} \quad F^{\#} = \lambda e. \lambda k. \llbracket E_k[M] \rrbracket_{\mathbb{H}}^{\#}(e). \end{aligned} \quad (6.20)$$

Les fonctions $F^{\#}$ et $G^{\#}$ sont décrites à l'aide de la notation du lambda calcul. La fonction $F^{\#}$ représente l'application de la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{H}}^{\#}$ pour un instant i et l'environnement abstrait e donné. La fonction $G^{\#}$ décrit le calcul du plus petit point fixe induit par l'évolution temporelle à partir d'un environnement abstrait $e^{\#}$ donné. L'abstraction des séquences de valeurs permet de remédier au problème dû à l'évolution temporelle infinie. Nous obtenons ainsi un moyen effectif de calculer l'ensemble de comportements des modèles hybrides Simulink.

La correction de la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{A}}^{\#}$ par rapport à la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{A}}$ est induite par la correction de la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{H}}^{\#}$ vis-à-vis de la sémantique concrète $\llbracket \cdot \rrbracket_{\mathbb{H}}$.

Boucle de simulation abstraite. A partir de la sémantique abstraite $\llbracket \cdot \rrbracket_{\mathbb{A}}^{\#}$, nous pouvons traduire la boucle de simulation de Simulink en une boucle de simulation abstraite. Cette dernière est décrite à l'algorithme 2. Elle est la traduction directe de la sémantique définie à l'équation (6.20). La boucle de simulation abstraite prend en argument un modèle Simulink M , une fonction de partition μ et des états initiaux σ_0 . Nous supposons, pour simplifier, que les entrées du système sont représentées dans le modèle M , c'est-à-dire qu'elles sont alors représentées par un système d'équations. Cet algorithme calcule les séquences abstraites de chaque signal du modèle M suivant la fonction de partition μ . Le calcul de point fixe est alors représenté par la boucle **répéter-jusqu'à**. $\sigma_{\mu}^{\prime\#}$ est une variable auxiliaire permettant de vérifier que nous arrivons à un point fixe. Elle est initialisée avec

l'environnement vide, qui est noté \perp . La comparaison d'environnements déterminant l'arrivée à un point fixe est donnée par l'opération \sqsubseteq_Σ . Cette opération est définie suivant le type de M soit par l'équation 6.4 si M est à temps discret, soit par l'équation 6.7 si M est à temps continu.

Algorithme 2 Boucle de simulation abstraite.

Entrée(s): M un modèle Simulink, μ une fonction de partition

Sortie(s): $\sigma_\mu^\#$ invariants

$k = 0$

$\sigma_\mu^\# = \sigma_0$

$\sigma'_\mu = \perp$

répéter

$\sigma'_\mu = \sigma_\mu^\#$

$\sigma_\mu^\# = (\llbracket E_k \llbracket M \rrbracket \rrbracket_\mathbb{H})^\#(\sigma_\mu^\#)$

$k = k + 1$

jusqu'à $\sigma_\mu \sqsubseteq_\Sigma \sigma'_\mu$

Bien que la sémantique abstraite $(\llbracket \cdot \rrbracket_\mathbb{A})^\#$ fournisse un calcul effectif des comportements d'un modèle Simulink, elle est fondée sur des domaines numériques, c'est-à-dire des treillis complets, de hauteur infinie. Ce fait induit un possible temps de calcul infini. Il est alors nécessaire d'utiliser un opérateur d'élargissement (voir la section 2.3) afin d'accélérer le calcul de point fixe. Il est possible de l'appliquer dans notre analyse en substituant l'opération d'union par celle de l'élargissement dans la sémantique des équations donnée à l'équation (5.11). Nous rappelons cette équation :

$$(\llbracket \ell(k) = e \rrbracket_{\mathbb{KV}})^\#(\sigma_\mu^\#) = \sigma_{\mu, \mu(k)}^\# \left[\ell \leftarrow \sigma_{\mu, \mu(k)}^\#(\ell) \sqcup_{\mathbb{V}}^\# \llbracket e \rrbracket_{\mathbb{V}}^\#(\sigma_{\mu|_{\mu(k)}}^\#) \right].$$

Dans notre cas, nous utilisons l'opérateur d'élargissement $\nabla_{\mathbb{I}}$ sur les intervalles, défini à l'équation (2.2) et l'opérateur d'élargissement $\nabla_{\mathcal{T}}$ sur les formes de Taylor centrées, défini à l'équation (5.1.3).

Nous voyons, dans le cas de l'opération d'élargissement, l'importance de la partition des séquences. En effet, nous évitons ainsi de définir un opérateur d'élargissement sur le temps. Nous définissons cet opérateur uniquement sur les valeurs des séquences. Par exemple, dans le cas d'une entrée périodique de période p , la définition d'une partition périodique de période p (voir équation (6.21)), nous permet de capter l'ensemble des comportements du système pour un temps infini mais dans une représentation finie. Et dans ce cas, l'opérateur d'élargissement est appliqué uniquement sur les éléments de la partition. L'extrapolation sur le temps futur est ainsi inutile.

$$\mu(n) = \begin{cases} 0 & \text{si } n \bmod p = 0 \\ 1 & \text{si } n \bmod (p+1) = 0 \\ \vdots & \vdots \\ p-1 & \text{si } n \bmod (2p-1) = 0 \end{cases}. \quad (6.21)$$

Il faut cependant remarquer que le choix de la partition est très important pour l'analyse statique de modèles Simulink. Et ce choix dépend grandement des entrées du système. Dans le cas d'une entrée périodique, le choix de la partition est évident pour capter l'ensemble des comportements d'un système analysé. Par contre, pour des entrées plus complexes, il est plus difficile de trouver une partition qui captera tous les comportements.

Exemple 6.3 Prenons le système S de la figure 6.1. Nous ne considérons les environnements qu'à la première itération, c'est-à-dire que nous ne donnons pas la séquence associée à chaque variable mais uniquement leur valeur du premier élément de la séquence. A la première itération l'évaluation de S_1 dans l'environnement $\theta_0^\#$ (contenant la valeur initiale de l'intégrateur et elle est fixée à la valeur 5) suivant la sémantique $(\llbracket \cdot \rrbracket_{\mathbb{C}})^\#$ produit la sortie :

$$\theta_{\mu,i}(\text{out1}) = ([5, 5], [5, 5])$$

Nous considérons que la valeur initiale est exacte. Le passage du système S_1 au système S_2 induit un changement de sémantique. Plus précisément, un passage de la sémantique des modèles à temps continu $(\cdot)_\mathbb{C}^\#$ à la sémantique des modèles à temps discret $(\cdot)_\mathbb{D}^\#$. Nous utilisons alors la fonction $\text{sample}^\#$. Nous obtenons alors comme valeur de l'entrée du système S_2 le nombre flottant avec erreurs différenciées suivant :

$$\begin{aligned}\phi_0^\#(\ell_1) &= \text{sample}_0^\# \left(\theta_i^\#(\text{out1}), \{\ell_1\} \right) \\ &= \uparrow_q([5, 5]), \quad [5, 5] - [5, 5] + \downarrow_q([5, 5]) \\ &= [5, 5], \quad [0, 0]\end{aligned}$$

Nous considérons que la quantification n'introduit pas de nouvelle approximation. Nous avons alors une valeur flottante qui n'est pas entachée d'erreur. L'évaluation du système S_2 avec cette entrée précédente produit la sortie suivante :

$$\phi_0^\#(\text{out1}) = ([0.01, 0.01] \times [5, 5], \downarrow([0.01, 0.01] \times [5, 5]))$$

Nous considérons que la valeur initiale de l'état du bloc *UnitDelay* du système S_2 est égal à 0. De plus, nous supposons que la constante 0.01 est représentée exactement en mémoire sinon une erreur pourrait lui être associée.

Cet exemple montre que le passage d'une sémantique à une autre est très intuitif et qu'il permet une communication entre différentes sémantiques.

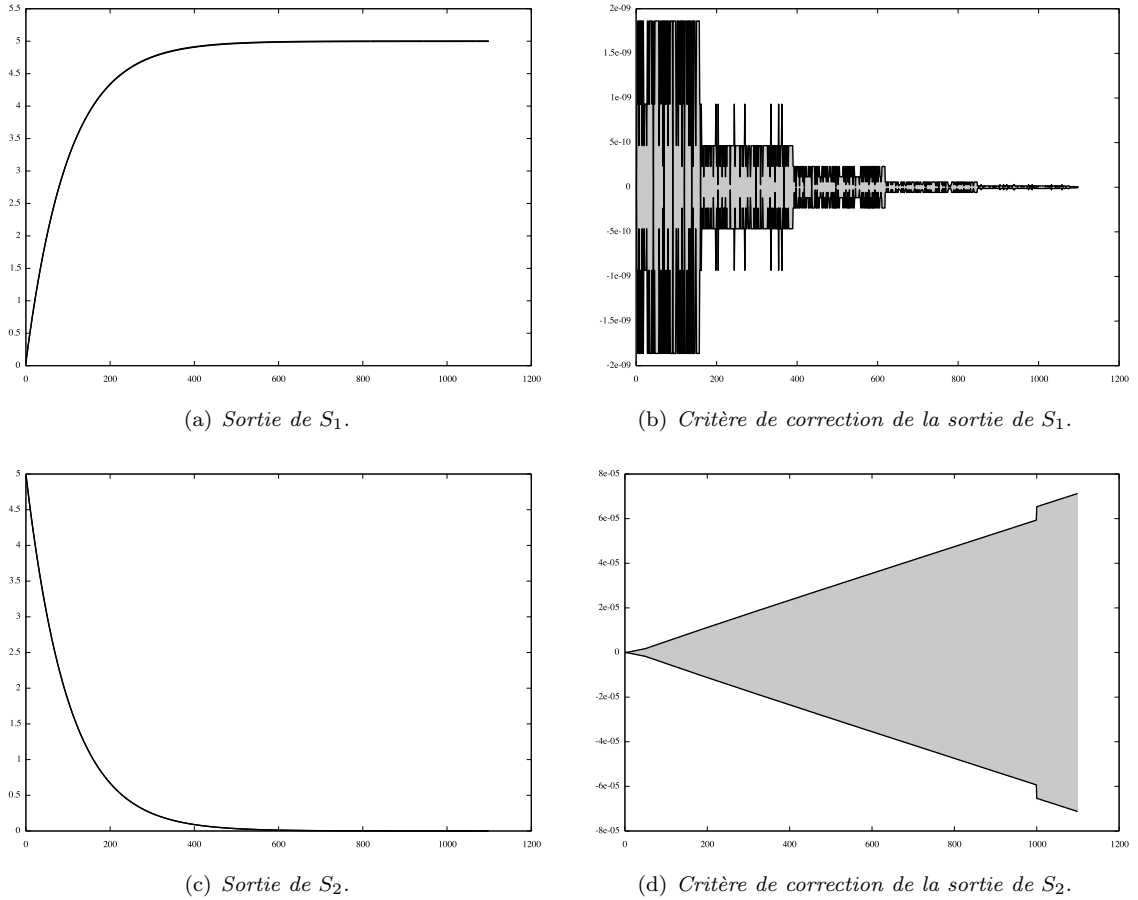


FIG. 6.3 – Résultats d'une simulation abstraite.

Afin de mettre en évidence la qualité numérique du système S , nous avons réalisé une simulation abstraite. Les résultats sont donnés à la figure 6.3. Nous utilisons un pas d'intégration $h = 0.01$. Nous avons utilisé une fonction de partition μ définie par :

$$\mu(k) = \begin{cases} k & \text{si } k \leq 1000 \\ k + 1 & \text{si } k \bmod 1001 \equiv 0 \\ \vdots & \vdots \\ k + 100 & \text{si } k \bmod 1100 \equiv 0 \end{cases}$$

En d'autres termes, nous considérons exactement l'évolution temporelle des signaux sur les 1000 premières itérations. Ceci correspond pour un pas de d'intégration h de 0.01 à 10 secondes de simulation. Puis, nous utilisons une partition périodique d'une période de 100 itérations. Nous obtenons alors des séquences abstraites contenant 1100 éléments.

La premier constat est que les sorties des systèmes S_1 (voir figure 6.3(a)) et S_2 (voir figure 6.3(c)) sont conformes aux résultats fournis par Simulink (voir figure 6.2). De plus, le critère de correction du système S_1 montre que l'intégration par une méthode d'Euler induit une erreur de méthode majorée par 2^{-9} . Ce qui signifie que nous avons une stabilité numérique. Pour le système S_2 , le critère de correction montre que les erreurs d'arrondi sont majorées par la valeur $8e^{-5}$. Ce qui montre une nouvelle fois la stabilité numérique du système S_2 suivant l'environnement d'exécution décrit par le système S_1 .

L'analyse statique définie dans les chapitres précédents a été implémentée dans un prototype d'analyseur statique de modèles Simulink. Il permet de valider expérimentalement les résultats théoriques et il montre ainsi l'intérêt de notre démarche.

Ce chapitre est organisé de la manière suivante. Nous présentons l'architecture générale de l'analyseur à la section 7.1. Nous décrivons, à la section 7.2, les résultats expérimentaux obtenus à partir de deux modèles Simulink.

7.1 Aspects logiciels de l'analyseur

Les modèles Simulink sont sauvegardés dans des fichiers au format texte dont l'extension est *mdl*. La difficulté à utiliser ces fichiers comme entrée d'un analyseur statique (ou de tout autre logiciel) est due à leur constante évolution au fil des versions de Simulink. Nous avons alors décomposé notre analyseur en un parser des fichiers *mdl* écrit en Matlab et présenté à la section 7.1.1, et en un analyseur écrit en OCaml et présenté à la section 7.1.2.

7.1.1 Parser de fichiers mdl

Les fichiers *mdl* contiennent l'ensemble des informations associées à un modèle Simulink, que cela soit sa représentation graphique ou les paramètres du simulateur. Par conséquent, ils sont constitués d'une grande quantité d'informations non pertinentes pour notre analyse statique. De plus, les évolutions successives de Simulink, ainsi que les ajouts de nouvelles fonctionnalités ou de nouvelles bibliothèques, induisent des modifications dans les fichiers *mdl*. La définition usuelle d'un parser à partir d'un analyseur lexical et d'un analyseur grammatical (par exemple flex/bison) n'est pas souhaitable dans le cas des fichiers *mdl*. En effet, les modifications du fichier, issues des évolutions de Simulink, entraîneraient une réécriture (ou au moins une adaptation) du parser.

Simulink est une extension du logiciel Matlab, donc son intégration à Matlab est complète. Et un ensemble de fonctions¹ permet d'obtenir des informations sur les modèles Simulink. Par conséquent, nous avons opté pour l'écriture d'un parser de fichier *mdl* en Matlab². Notre parser permet de convertir un modèle Simulink en un ensemble de fichiers XML et DOT, contenant uniquement les informations nécessaires pour notre analyse statique. Et surtout, le parser n'est pas soumis aux modifications des fichiers *mdl*. XML a été choisi pour sa capacité à organiser

¹Les fonctions de constructions de modèles (voir la section 3.2 du manuel de référence).

²Le développement du parser a été réalisé par Maxime Lim au cours d'un stage de quatrième année d'école d'ingénieur (Polytech'Paris-UPMC) durant l'été 2007 [Lim07].

des données et, DOT, langage de description de graphes, pour sa simplicité syntaxique et son rendu graphique. Ces deux formats sont textuels et par conséquent, ils facilitent les manipulations ultérieures.

Informations sélectionnées

Chaque bloc possède un ensemble de paramètres qui sont de deux natures. Les paramètres communs à tous les blocs, par exemple leur nom (*Name*), leur identifiant unique (*Handle*) ou encore les informations sur les blocs qui leur sont connectés (*PortConnectivity*). Les paramètres spécifiques à chaque bloc, par exemple le bloc **Gain** possède un paramètre nommée *Gain* qui donne la valeur de la constante multiplicatrice.

Un modèle Simulink a une organisation arborescente qui attribue à chaque bloc un chemin. Ce chemin décrit la relation d'inclusion des sous-systèmes dans un modèle. Il décrit alors la position d'un bloc par rapport au système principal. L'ensemble des chemins d'un modèle peut être récupéré grâce à la fonction Matlab `find_system`. En appliquant cette fonction sur le modèle de la figure 4.2(a), nous obtenons la représentation arborescente de la figure 7.1. Nous ne détaillons que le sous-système S_2 . A partir de cet arbre, il est possible d'attribuer un chemin unique vers chaque bloc. Par exemple le chemin $S \rightarrow S_2 \rightarrow \text{Out1}$ correspond à la première sortie du système S_2 se trouvant dans le système S .

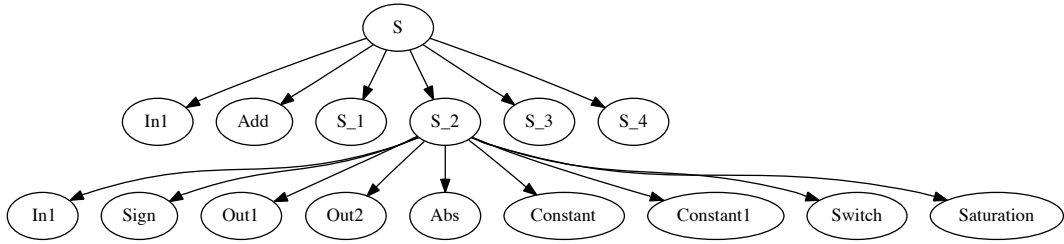


FIG. 7.1 – Représentation arborescente du système S de la figure 4.2(a).

La fonction Matlab `get_param` permet de récupérer la valeur d'un paramètre d'un bloc en connaissant son chemin dans l'arborescence. Par exemple, l'application de cette fonction avec le chemin $S \rightarrow S_1 \rightarrow \text{Constant1}$ et le paramètre *Constant* retourne la valeur -1 .

Afin de faciliter la prise en compte de blocs dans notre parser, nous définissons une structure de données en Matlab référençant les blocs que nous voulons prendre en compte. Cette structure de données liste également les paramètres associés à ces blocs. Nous notons B la liste des blocs et de leurs paramètres composant les modèles Simulink que l'analyse statique traitera.

Un modèle Simulink possède des paramètres propres qui sont, en général, liés au moteur de simulation. Par exemple, le type d'algorithme d'intégration numérique est associé au paramètre du modèle nommé **Solver** où le temps de fin de simulation est donné par la variable **StopTime**. Nous conservons la liste des paramètres de modèle et nous la notons P .

Génération des fichiers XML et DOT

Un modèle Simulink est représenté par un diagramme en blocs dont les éléments sont des opérations élémentaires (par exemple une addition ou une intégration), ou un sous-système, c'est-à-dire un élément représentant un autre diagramme en blocs (par exemple le système de la figure 4.2(a)). La représentation arborescente de la figure 7.1 fait apparaître une forme de récurrence dans la représentation des modèles. Nous exploitons cela pour récupérer des informations les

concernant. Plus particulièrement, nous allons parcourir l'arbre des chemins pour recueillir les informations sur les blocs des modèles Simulink.

La génération des fichiers XML et DOT suit l'algorithme 3. Cet algorithme applique un parcours en profondeur d'abord de l'arbre des chemins d'un modèle Simulink. Pour chaque système Simulink, grâce à la fonction `find_system` et les paramètres adéquats, nous récupérons la liste des blocs d'un niveau n de l'arbre. Pour chaque bloc b , si b est un sous-système alors nous l'ajoutons dans la liste S des sous-systèmes à visiter et, dans tous les cas, nous récupérons les valeurs de ses paramètres (`Information(B, b)`) grâce à la liste des blocs B . Dans le même temps, nous récupérons la liste de ses successeurs (`Successeur(b)`) afin de générer le graphe de dépendance. Une fois toutes les informations collectées nous générons les fichiers XML et DOT. Puis le traitement se poursuit avec les sous-systèmes contenus dans S .

Algorithme 3 *Génération des fichiers XML et DOT d'un modèle Simulink.*

Entrée(s): B la liste des blocs et paramètres gérés ; C le chemin d'un système d'un modèle M

```

 $I := \emptyset$  {liste des informations des blocs}
 $G := \emptyset$  {liste d'adjacence}
 $S := \emptyset$  {liste des sous-systèmes de  $N$ }
 $L := find\_system(C, 'SearchDepth', 1)$  {la liste des chemins des blocs contenus dans  $M$ }
pour toute bloc  $b \in L$  faire
  si  $b$  est un sous-système alors
     $S := S \cup b$ 
  fin si
  si  $b \in B$  alors
     $I(b) := Information(B, b)$ 
     $G(b) := Successeur(b)$ 
  sinon
    Erreur
  fin si
fin pour
générer_XML( $I$ )
générer_DOT( $G$ )
pour toute  $s \in S$  faire {Parcours des sous-systèmes}
  Appliquer l'algorithme 3 sur  $s$ 
fin pour

```

Le fichier XML du système S est donné à la figure 7.2, il est composé de trois parties :

- la liste des blocs et leurs paramètres (`In1` et `Add`) ;
- la liste des sous-systèmes (`S1`, `S2`, `S3` et `S4`) ;
- le nom du fichier DOT (`S.dot`).

Le graphe de dépendance des blocs de S est donné à la figure 7.3. Il a été obtenu à partir du fichier DOT suivant :

```

digraph "S" {
  "In1" -> "Add" [label="(1,1)"];
  "Add" -> "S_1" [label="(1,1)"];
  "S_3" -> "S_4" [label="(1,1)"];
  "S_1" -> "S_2" [label="(1,1)"];
  "S_4" -> "Add" [label="(1,2)"];
  "S_2" -> "S_3" [label="(1,2)"];
  "S_2" -> "S_3" [label="(2,1)"];
}

```

Les étiquettes (*label*) sur les arcs du graphe de dépendance permettent de connaître l'ordre des arguments d'un système. Par exemple, les arcs entre S_2 et S_3 identifient que la première sortie de S_2 est reliée à la seconde entrée de S_3 et que la seconde sortie de S_2 est connectée à la première

```

<?xml version="1.0" encoding="UTF-8"?>

<simulink>
  <system name="S">
    <blocks_list>
      <block name="In1" type="Inport">
        <parameter name="Path" value="S/In1"></parameter>
        <parameter name="Handle" value="4.000977e+00"></parameter>
        <parameter name="Description" value=""></parameter>
        <parameter name="Port" value="1"></parameter>
      </block>
      <block name="Add" type="Sum">
        <parameter name="Path" value="S/Add"></parameter>
        <parameter name="Handle" value="5.000977e+00"></parameter>
        <parameter name="Description" value=""></parameter>
        <parameter name="Inputs" value="+-"></parameter>
      </block>
    </blocks_list>
    <subsystems_list>
      <subsystem name="S_3" file="S_3.xml"></subsystem>
      <subsystem name="S_1" file="S_1.xml"></subsystem>
      <subsystem name="S_4" file="S_4.xml"></subsystem>
      <subsystem name="S_2" file="S_2.xml"></subsystem>
    </subsystems_list>
    <graph name="S" file="S.dot"></graph>
  </system>
</simulink>

```

FIG. 7.2 – Contenu du fichier *S.xml*.

entrée de S_3 .

Au final, notre parser génère un couple de fichier XML/DOT pour chaque système d'un modèle Simulink. De plus, il engendre un fichier XML contenant les paramètres du modèle, à partir de P , et le nom du système principal. Pour le modèle du régulateur de papillon des gaz, nous obtenons les fichiers :

$S_model.xml$	$S.dot$	$S_1.dot$	$S_2.dot$	$S_3.dot$	$S_4.dot$
$S.xml$	$S_1.xml$	$S_2.xml$	$S_3.xml$	$S_4.xml$	

7.1.2 Analyseur statique

Dans cette section, nous présentons l'aspect logiciel de notre prototype d'analyseur statique. Une représentation graphique de cette architecture est donnée à la figure 7.4. L'analyseur statique travaille à partir d'un ensemble de fichiers d'entrée, c'est-à-dire les fichiers XML et DOT fournis par notre parser et de trois fichiers de configuration. La déclaration des types de système, à temps continu ou à temps discret, est faite manuellement et elle se trouve dans un fichier `types.txt`. La présence de paramètres issus de l'environnement Matlab nécessite l'utilisation d'un fichier d'entrée (`workspace.txt`) contenant les valeurs de ces paramètres. Les entrées du modèle sont regroupées dans un troisième fichier `inputs.txt`.

Nous retrouvons dans l'architecture logicielle les différents éléments permettant de traiter les précédents fichiers, en l'occurrence les parsers associés à chacun d'entre eux. A partir des fichiers XML, de la déclaration des types et des valeurs des paramètres, nous générons le système d'équations associé au modèle Simulink. A partir du fichier DOT et donc du graphe de dépendance, nous calculons l'ordre d'évaluation des équations. Le fichier décrivant les entrées du modèle est également converti en un système d'équations.

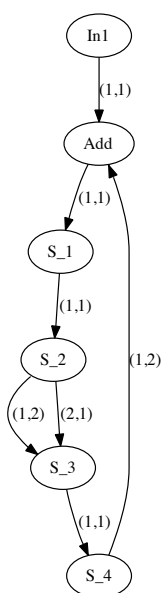
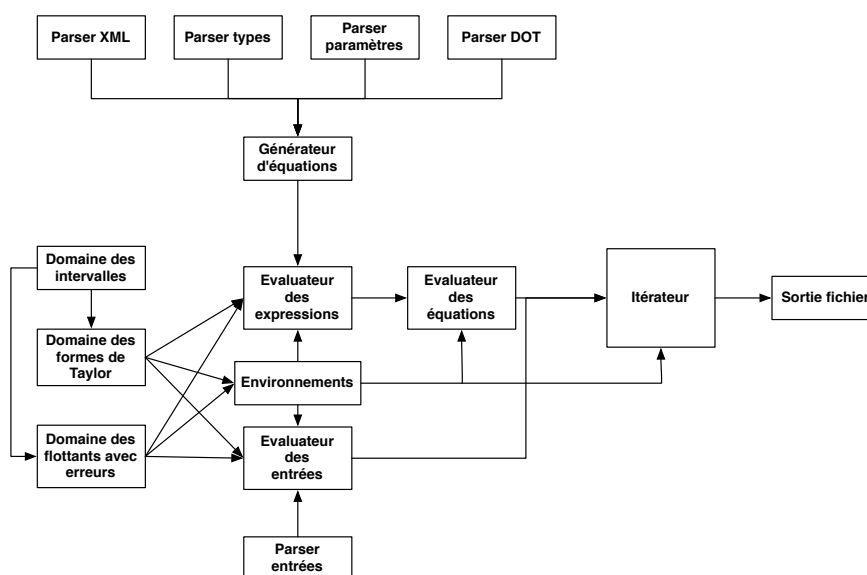
FIG. 7.3 – Graphe de dépendance du système S de la figure 4.2(a).

FIG. 7.4 – Architecture logicielle de l'analyseur statique.

Les différents domaines numériques que sont le domaine des intervalles (voir section 3.2.1), le domaine des formes de Taylor (voir section 5.1) et le domaine des flottants avec erreurs (voir section 3.2.4) sont implémentés dans l'analyseur. Ils sont les éléments de base pour définir les fonctions d'évaluation des équations ainsi que les environnements. L'itérateur, c'est-à-dire le composant logiciel permettant de calculer la sémantique abstraite (voir section 2.3), prend en paramètre la fonction de partition et l'évaluateur d'équations. Il génère un ensemble de fichiers décrivant les séquences abstraites résultat de l'analyse.

7.2 Résultats expérimentaux

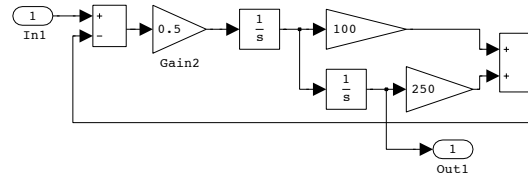
Nous présentons deux études de cas issues de l'industrie automobile. Le premier, présenté à la section 7.2.1, est un détecteur de freinage. Le second, introduit au chapitre 4, représente un contrôleur du papillon des gaz. Les résultats expérimentaux sont donnés à la section 7.2.2.

7.2.1 Détection de freinages

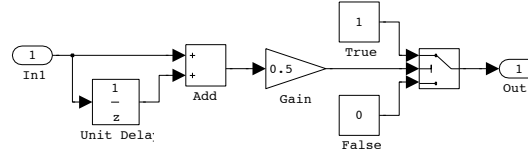
Dans cette section, nous présentons le modèle Simulink lié au système de détection de freinage. Cet exemple s'intéresse à un modèle hybride simple combinant un système mécanique et un programme. Nous considérons ainsi une analyse statique des systèmes sans rétroaction.

Modèle Simulink

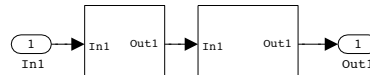
Le système de détection de freinage est composé d'une pédale et d'un détecteur. Le modèle Simulink de ce système est représenté à la figure 7.5. La représentation de la pédale se fonde sur la modélisation mathématique issue de la seconde loi de Newton (la somme des forces d'un système est égale à l'accélération multipliée par la masse) régissant un système composé d'une masse, d'un ressort et d'un amortisseur (voir figure 7.5(a)). Le détecteur de freinage est un système à temps discret qui retourne une valeur booléenne si son entrée, c'est-à-dire la force mesurée à partir de la pédale, dépasse un certain seuil (voir figure 7.5(b)). Le système complet est alors obtenu par simple composition des deux précédents systèmes (voir figure 7.5(c)).



(a) Modèle de la pédale.



(b) Modèle du détecteur.



(c) Système complet.

FIG. 7.5 – Modèle Simulink du détecteur de freinage

Expérimentation

Pour notre première expérimentation, nous considérons la fonction d'entrée $u(t)$ définie par la relation

$$u(t) = \begin{cases} [100, 200] \times t & \text{si } 0 \leq t < 1 \\ [100, 200] & \text{si } 1 \leq t < 1.5 \\ 0 & \text{sinon} \end{cases}$$

Nous fixons également la valeur du seuil du contrôleur à la valeur 0.01. Nous donnons une

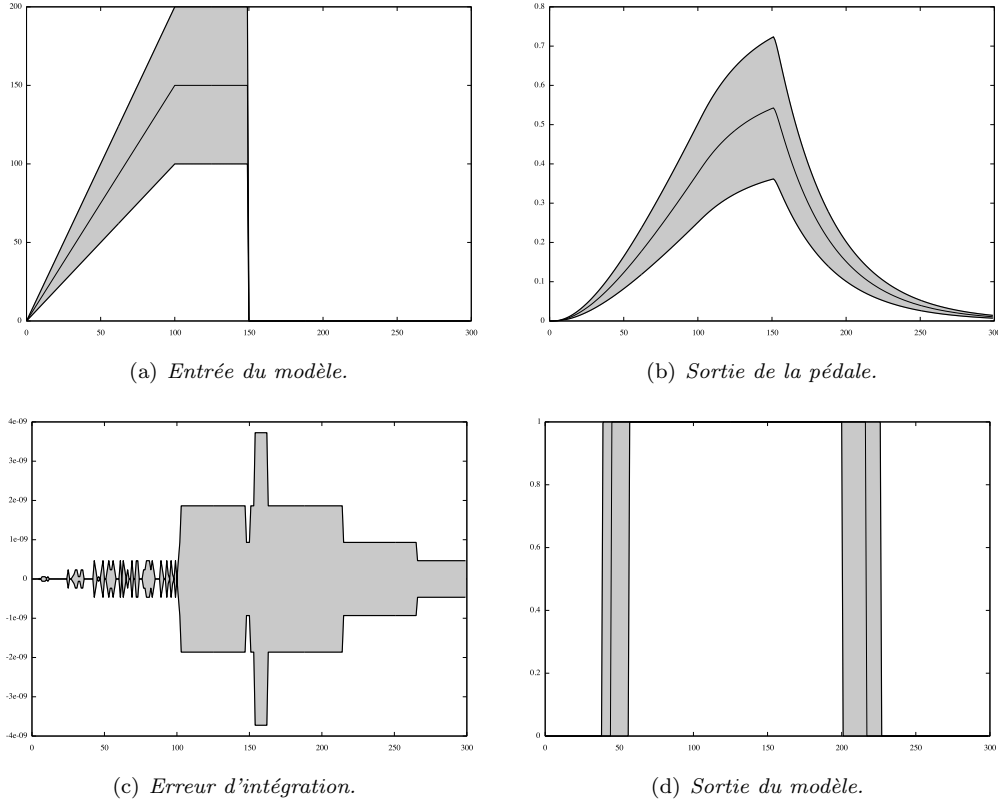


FIG. 7.6 – Résultats de l'analyse du détecteur de freinage.

représentation graphique de $u(t)$ à la figure 7.6(a). La zone grise représente l'ensemble de valeurs décrites par $u(t)$. Nous avons illustré par une ligne noire une trajectoire particulière de cet ensemble de points, qui correspond à un résultat de simulation en prenant comme valeur le milieu de l'intervalle $[100, 200]$. De plus, nous avons utilisé l'intégration numérique par la méthode d'*Euler* avec un pas d'intégration $h = 0.01$.

Pour cette expérience, nous utilisons la fonction de partition μ définie par la relation

$$\mu(k) = \begin{cases} k & \text{if } k \leq 3 \\ 4 & \text{sinon} \end{cases}.$$

Cette partition permet de considérer les 3 premières secondes de l'évolution temporelle du système et de collecter toutes les autres valeurs dans un seul élément.

La séquence de valeurs associée à la sortie de la pédale est donnée à la figure 7.6(b) et l'évolution de la distance entre l'intégration par la méthode d'Euler et l'intégration garantie est donnée à la figure 7.6(c). Nous pouvons constater que cette erreur est inférieure à $4e^{-9}$, ce qui montre une stabilité numérique de l'intégration. La sortie du détecteur est donnée à la figure 7.6(d). Nous remarquons que toutes les détections supérieures à 0.1 le sont. De plus, la trajectoire de la simulation est bien encadrée par notre analyse.

7.2.2 Contrôleur du papillon des gaz

Le second exemple est le contrôleur du papillon des gaz introduit au chapitre 4. Nous rappelons que ce modèle est composé principalement de deux éléments : un régulateur PI (sous-système S_1) et d'une modélisation du papillon des gaz (sous-système S_3). Nous fixons les paramètres du modèle de la manière suivante :

- $J = 0.000245$;
- $K_s = 0.2$;
- $K_d = 0.03$;
- $C_s = 3.0$;
- $K_i = 2.0$;
- $K_p = 0.3$;
- $T_s = 0.01$;
- $teq = 0.523599$;
- $\pi = 3.1415$.

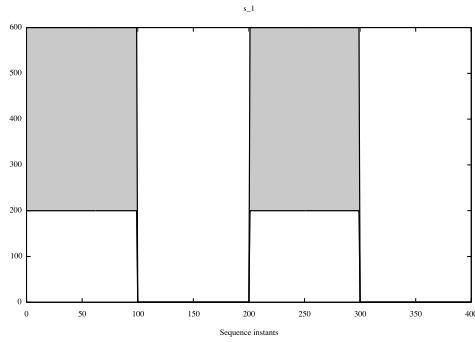
Nous définissons l'entrée du système par une fonction périodique, représentée à la figure 7.7(a). La période d'une durée de deux secondes est découpée de la façon suivante :

- la première seconde est un ensemble de fonctions constante dont les valeurs sont prises dans l'intervalle $[200, 600]$;
- la deuxième seconde la fonction est nulle.

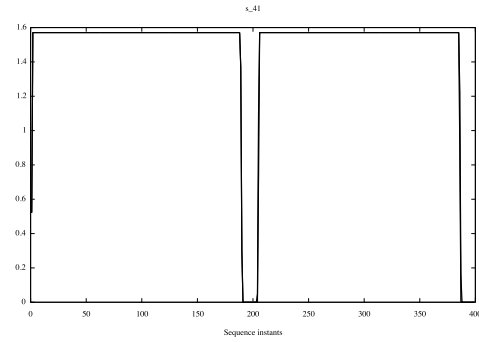
De plus, nous utilisons la fonction de partition μ suivante :

$$\mu(k) = \begin{cases} k & \text{si } k < 200 \\ k+1 & \text{si } k \equiv 0 \pmod{200} \\ k+2 & \text{si } k \equiv 0 \pmod{201} \\ \vdots & \vdots \\ k+200 & \text{si } k \equiv 0 \pmod{399} \end{cases}$$

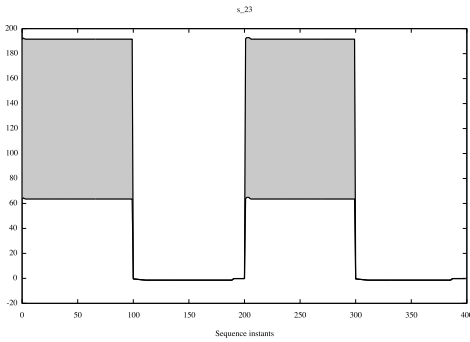
En d'autres termes, nous considérons une période puis nous utilisons une partition périodique pour toutes les autres périodes. Nous fixons le pas d'intégration $h = 0.01$.



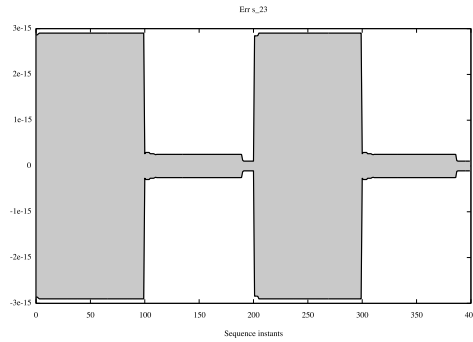
(a) Entrée du modèle.



(b) Sortie du système S_3 .



(c) Sortie du système S_1 .



(d) Erreur d'arrondi en sortie de S_1 .

FIG. 7.7 – Résultats expérimentaux du système de régulation du papillon des gaz.

La sortie du régulateur PI est donnée à la figure 7.7(c) et nous donnons à la figure 7.7(d) l'évolution des erreurs d'arrondi liées à la sortie de ce système. La sortie du système S_3 est fournie à la figure 7.7(b) dont les valeurs sont incluses dans l'intervalle $[0, \pi/2]$ avec une dynamique suivant la dynamique de l'entrée du système. Nous montrons ainsi la stabilité numérique du système en présence d'erreurs d'arrondi. En particulier, nous validons la stabilité numérique du contrôleur en fonction de la dynamique de l'environnement d'exécution.

Troisième partie

Réflexions et conclusion

Lamarre : « Mais cela ne nous donne pas la réponse, ça ne nous donne qu'une autre question. »

Fred Vargas, *Un lieu incertain*.

Nous présentons dans ce chapitre des pistes de travail potentielles qui pourront être consécutives aux travaux réalisés au cours de cette thèse. En effet, l'expressivité du langage Simulink offre un nombre important de possibilités d'évolution.

Ce chapitre est organisé comme suit. A la section 8.1, nous mettons en évidence les autres aspects du langage Simulink intéressants pour l'analyse statique des spécifications. Ensuite nous présentons, à la section 8.3, une piste pour la validation de propriétés temporelles. Puis nous abordons, à la section 8.2, au travers d'outils mathématiques spécifiques du traitement du signal, une autre analyse statique de modèles Simulink. Et pour finir à la section 8.4, nous élaborons un modèle de validation de logiciels embarqués utilisant les modèles Simulink tout au long du cycle de développement.

8.1 Extensions du langage Simulink

Les différents éléments du langage Simulink n'ont été que partiellement traités au cours des travaux réalisés durant cette thèse. En particulier, nous avons concentré nos efforts sur le noyau à flots de données du langage Simulink permettant de définir des systèmes à temps continu ou à temps discret. L'extension du langage par la prise en compte d'un nombre plus important de blocs des bibliothèques Simulink ainsi que la gestion des différents algorithmes d'intégration numérique sont une piste de recherche naturelle. Néanmoins, il existe une extension du langage qui a un impact non négligeable sur le formalisme Simulink et sur la définition d'une analyse statique.

Stateflow est une extension du langage Simulink qui offre la possibilité de définir de nouveaux blocs par des automates, c'est-à-dire des structures de contrôle événementiel. Cette extension est très utile pour définir les algorithmes de contrôle très présents dans les logiciels embarqués. Ces structures événementielles permettent de définir des blocs Simulink, c'est-à-dire des opérations agissant sur des flots de données, mais en mettant l'accent sur le contrôle plus tôt que sur le traitement des données. Cependant, la sémantique associée à ces automates est loin d'être triviale, en particulier, par la présence de mémoire partagée et d'effets de bord.

Un programme Statflow est un automate composé d'états et de transitions dont les états peuvent également être des automates¹. Il a potentiellement plusieurs entrées et plusieurs sorties. A chaque état est associé des actions, qui sont exécutées soit avant d'entrer dans l'état (*entry action*), soit à la sortie de l'état (*exit action*) ou encore pendant l'exécution d'une transition menant à cet état (*during action*). Une transition d'un automate Stateflow peut être munie d'un

¹Stateflow est une des innombrables implémentations des Statecharts[HPSS87].

ensemble d'actions et elles sont représentées par une étiquette de la forme :

$$E[C]\{A_c\}/A_t,$$

où E est un évènement, C une condition, A_c une condition d'action et A_t une action de transition. Tous ces éléments sont optionnels. Un évènement est une entité non représentée dans un programme Stateflow mais qui contrôle toute son exécution. Des évènements sont, par exemple, associés aux entrées du programme et aux sorties. A chaque fois qu'une nouvelle entrée est accessible, alors l'évènement associé est émis. Soit la transition $[x < 10]\{y = y - 2\}/E_1$, elle est valide (elle peut être choisie) si la valeur de x est inférieure à 10, si la transition est choisie alors la variable y est décrémentée de 2 et l'évènement E_1 , supposé défini au préalable, est émis.

L'algorithme de simulation d'un modèle Stateflow est relativement complexe, nous reprenons un bref aperçu de celui-ci, tiré de [SSC⁺04] et défini par les étapes suivantes :

- Recherche des états actifs : un état est actif s'il reçoit un évènement d'entrée. Cette recherche suit l'ordre imposé par la représentation graphique qui est de haut en bas et de gauche à droite afin d'éliminer un possible non-déterminisme ;
- Recherche des transitions valides : en fonction des évènements présents et des conditions sur les transitions ;
- Exécution des transitions valides : exécution en séquence de la condition d'action puis de l'action de transition. Les conséquences de ces actions peuvent générer de nouveaux évènements et donc activer de nouveaux états ;
- Fin : quand il n'y a plus d'état actif, un état est inactif quand il a exécuté toutes ses actions.

Le pouvoir expressif du langage Statflow est très important. En combinant, des évènements, des transitions gardées (munies de conditions) et des variables partagées, nous obtenons des programmes dont les exécutions sont très difficiles à décrire formellement.

Nous donnons un exemple de la difficulté liée à la sémantique de Stateflow à la figure 8.1 et basé sur deux représentations graphiques différentes d'un même programme². Le programme est composé de trois états A , B et C et de points de jonction qui ont pour rôle de modéliser des décisions, ils peuvent se traduire par une conditionnelle. Nous considérons deux types de transition. Celles qui sont toujours exécutées et elles sont de la forme $[true]e$ où e est une expression. Et celles qui ne sont jamais exécutées et elles sont de la forme $[false]e$. De plus, pour toutes les transitions, l'action a pour effet de modifier la valeur de la variable a qui est initialisée avec la valeur 0. A la figure 8.1(a), l'évaluateur tente d'accéder à l'état B en passant par les deux points de jonction. La transition de la seconde jonction étant de la forme $[false]e$, l'accès à l'état B est impossible. L'exécution conduit donc à l'état C mais sans défaire ce qu'elle a affecté à a pour aller vers l'état B . Le résultat de cette exécution est $a = 111$. A la figure 8.1(b), l'exécution accède directement à l'état C . Le résultat de cette exécution est alors $a = 101$. Cet exemple met en évidence que la représentation graphique a une influence sur la sémantique et que cette dernière peut être à l'origine de comportements non désirés.

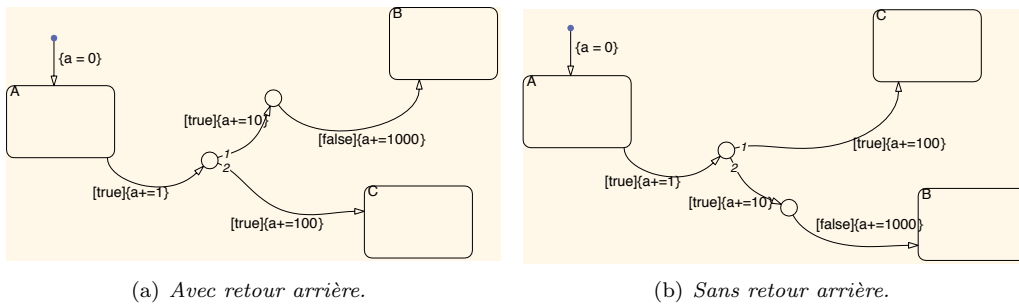


FIG. 8.1 – Deux représentations du même programme produisant des résultats différents.

²Programme tiré de [SSC⁺04].

Bien que la sémantique des modèles Stateflow soit complexe, il a été montré [SSC⁺04] qu'un sous-ensemble pouvait être représenté par un programme Lustre (voir section 2.5). Il est alors envisageable de s'appuyer sur ce sous-ensemble pour permettre une adaptation de notre formalisme pour prendre en compte des modèles Statflow. Une autre piste pour gérer ces modèles est les travaux de [CHP06] qui mélangent les programmes flots de données avec des automates. De plus, une formalisation de la sémantique de Stateflow [Ham05, HR07] pourra également être utilisée comme point de départ pour la définition d'une analyse statique des programmes Stateflow.

Une autre extension de l'analyse statique de programmes Simulink pourrait également être envisagée en considérant des programmes Lustre. Les grandes similitudes des deux langages ainsi que notre formalisme permettent une analyse statique dite hétérogène. En effet, en nous fondant sur [CCM⁺03a], qui met en évidence le lien étroit entre ces deux langages, nous pouvons définir une analyse statique de spécifications hétérogènes. Lustre/SCADE est généralement utilisé pour définir la partie logicielle des systèmes embarqués, notamment grâce à l'existence d'un générateur de code qualifiable suivant les normes de sécurité, alors que Simulink est très adapté à la modélisation d'environnements de physique. Ce constat est d'ailleurs une des motivations de [CCM⁺03b, CCM⁺03a, SSC⁺04]. Nous pouvons également adopter ce point de vue pour réaliser une telle analyse statique de spécifications de systèmes embarqués qui sont définis par de nombreux langages.

8.2 Etude fréquentielle des systèmes

Les modèles Simulink décrivent des systèmes évoluant dans le temps et les données qu'ils manipulent sont des fonctions du temps. Nous pouvons alors considérer les modèles Simulink comme des transformateurs de signaux, c'est-à-dire des filtres. La théorie du traitement du signal [Jac89, Kra92, Ben02] offre de nouveaux outils pour l'étude des modèles Simulink. Il faut remarquer que ces outils sont inclus dans la bibliothèque de Simulink dont le traitement du signal a été une des premières applications. L'approche proposée dans cette section est un début de formalisation dans la théorie de l'interprétation abstraite de ces outils et nous permet de définir une nouvelle analyse statique de modèles Simulink [CM06, CM07b].

Comme nous l'avons vu dans les chapitres précédents, nous pouvons classer les signaux en deux catégories : les signaux à temps continu et les signaux à temps discret. Il en résulte que les systèmes sont également classés en deux catégories : les systèmes à temps continu et les systèmes à temps discret. Un filtre est un système invariant du temps, c'est-à-dire que la réponse du système est indépendante de la date à laquelle s'effectue le traitement. Autrement dit, soit $r(t)$ la sortie d'un filtre à l'entrée $u(t)$ alors le signal $r(t-d)$ est associé à l'entrée $u(t-d)$ où d est une constante.

L'étude des filtres dans la théorie du traitement du signal utilise deux représentations :

- la représentation temporelle est la plus intuitive puisqu'elle décrit le signal en fonction de l'évolution du temps ;
- la représentation symbolique est une extension de la représentation fréquentielle (transformée de Fourier). Elle est caractérisée par l'utilisation de la transformée de Laplace pour les signaux à temps continu et par la transformée en Z pour les signaux à temps discret.

L'analyse statique définie dans la seconde partie de ce document s'est appuyée sur la représentation temporelle des signaux. Nous présentons maintenant les idées d'une analyse statique fondée sur la représentation symbolique. Nous restreignons notre étude aux filtres linéaires, c'est-à-dire que si $r_1(t)$ et $r_2(t)$ sont les sorties obtenues à partir des entrées $u_1(t)$ et $u_2(t)$ alors $c_1 r_1(t) + c_2 r_2(t)$ est la sortie issue de l'entrée $c_1 u_1(t) + c_2 u_2(t)$ avec c_1 et c_2 deux constantes.

L'approche fréquentielle ou symbolique est très utilisée dans l'étude des systèmes linéaires puisqu'elle facilite les calculs. En effet, dans le domaine temporel (continu ou discrets) la réponse $r(t)$ d'un filtre linéaire à une entrée $u(t)$ est obtenue par :

$$r(t) = h(t) * u(t), \quad (8.1)$$

où $h(t)$ est la réponse impulsionnelle du système et $*$ représente le produit de convolution défini

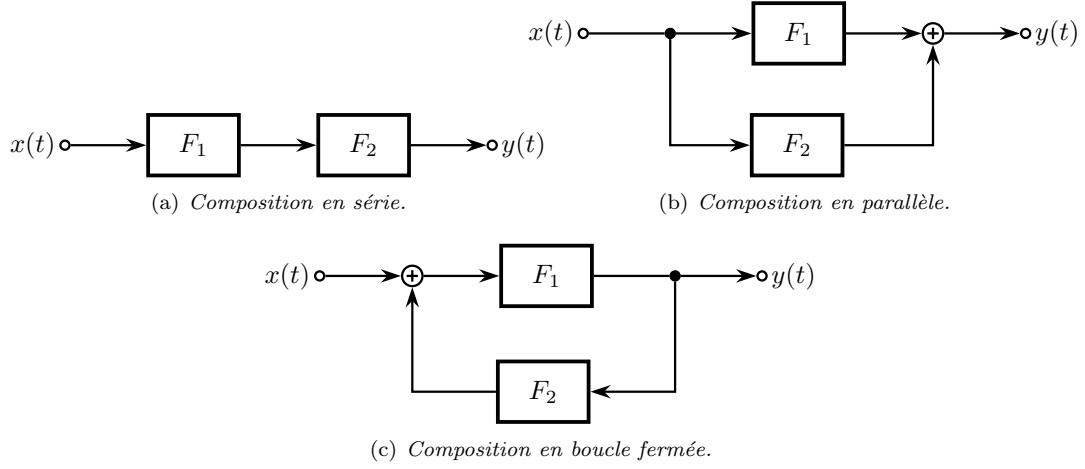


FIG. 8.2 – Règles de composition des filtres.

dans le cas des signaux à temps continu, par :

$$h(t) * u(t) = \int_{-\infty}^{+\infty} h(t-x)u(x)dx \quad (8.2)$$

et défini, dans le cas des signaux à temps discret, par :

$$h(k) * u(k) = \sum_{k=-\infty}^{+\infty} h(k-x)u(x). \quad (8.3)$$

La réponse impulsionnelle est la sortie du filtre associée à la fonction δ de Dirac, elle est définie par :

$$\delta(t) = \begin{cases} +\infty & \text{si } t = 0 \\ 0 & \text{sinon} \end{cases}. \quad (8.4)$$

D'autre part, il existe des règles de composition des filtres linéaires permettant de créer des filtres complexes à partir de filtres plus simples. La figure 8.2 donne les trois principales composition de filtres linéaires (continu ou discret) sous la forme de diagrammes en bloc. Le symbole \oplus représente aussi bien une addition qu'une soustraction. Nous notons $h_1(t)$ et $h_2(t)$ les réponses impulsionnelles des filtres F_1 et F_2 . La figure 8.2(a) est la composition en série (ou en cascade). La sortie d'un filtre est l'entrée d'un second filtre. La réponse $y(t)$ de cette composition est obtenue par un double produit de convolution $y(t) = x(t) * h_1(t) * h_2(t)$. La composition en parallèle décrite à la figure 8.2(b) représente la somme ou la soustraction des réponses de deux filtres soumis à la même entrée $x(t)$. La réponse $y(t)$ de cette composition est $x(t) * h_1(t) \pm x(t) * h_2(t)$. La composition en boucle fermée est décrite à la figure 8.2(c). Il est malheureusement plus difficile de définir la sortie $y(t)$ en fonction de l'entrée $x(t)$ et des réponses impulsionnelles des filtres.

Un filtre linéaire à temps continu F est stable, si la réponse $y(t)$ à un signal $x(t)$ borné est également bornée. Une condition nécessaire et suffisante de stabilité est que la réponse impulsionnelle $h(t)$ de F vérifie :

$$\int_{-\infty}^{\infty} |h(t)|dt < +\infty.$$

Nous constatons que la représentation temporelle permet de décrire les filtres ainsi que leurs propriétés, en particulier la stabilité, mais à partir d'opérations mathématiques difficiles à manipuler.

La représentation symbolique (ou fréquentielle) permet de simplifier considérablement les calculs en se fondant sur les propriétés de la transformée de Laplace ou de la transformée en Z .

Posons $X(s)$ la transformée (en Z ou de Laplace) d'un signal $x(t)$ et $Y(s)$ la transformée (en Z ou de Laplace) d'un signal $y(t)$. De plus, $y(t)$ est la réponse d'un filtre F à $x(t)$. Nous avons alors la relation suivante :

$$Y(s) = H(s) \times X(s), \quad (8.5)$$

où $H(s)$ est la transformée de la réponse impulsionnelle $h(t)$ de F nommée *fonction de transfert*. Le premier constat est que les transformées ont la propriété de substituer un produit de convolution en un produit "simple". Le second est lié aux définitions des transformées que nous rappelons ci-dessous. Soit $x(t)$ un signal à temps continu, sous certaines hypothèses [Ben02, chap. 2], sa transformée de Laplace $X(s)$ est

$$X(s) = \int_{-\infty}^0 e^{-st} x(t) dt + \int_0^{+\infty} e^{-st} x(t) dt. \quad (8.6)$$

Dans le cas d'un signal à temps discret $x(k)$, sous certaines hypothèses [Ben02, chap. 2], la transformée en Z $X(z)$ de $x(k)$ est donnée par la relation

$$X(z) = \sum_{n=-\infty}^0 x(n) z^{-n} + \sum_{n=0}^{+\infty} x(n) z^{-n}. \quad (8.7)$$

Dans les deux transformées, s et z sont des variables du plan complexe \mathbb{C} . Il faut remarquer qu'il existe également des transformées inverses faisant passer de la représentation symbolique à la représentation temporelle. Il faut souligner également que des tables de transformées existent, qui donnent les transformées de Laplace ou en Z d'un grand ensemble des signaux. Les transformées sont décrites, dans la plupart des cas, par des fonctions rationnelles. Par exemple, le signal de Heaviside

$$u(t) = \begin{cases} 1 & \text{si } t \geq 0 \\ 0 & \text{sinon} \end{cases} \quad \text{a pour transformée de Laplace } U(s) = \frac{1}{s}.$$

Grâce aux propriétés des transformées, par exemple la linéarité, il est possible de définir, pour toutes les compositions de filtres de la figure 8.2, les fonctions de transfert associées. Plus précisément, en notant H_1 et H_2 les fonctions de transferts des filtres F_1 et F_2 , nous avons :

- $H = H_1 \times H_2$ la fonction de transfert de la composition en série ;
- $H = H_1 \pm H_2$ la fonction de transfert de la composition en parallèle ;
- $H = \frac{H_1}{1 \pm H_1 \times H_2}$ la fonction de transfert de la composition en boucle fermée.

Le second constat est alors que les transformées fournissent une représentation compacte des signaux puisqu'elles représentent ceux-ci pour toutes les valeurs du domaine temporel.

La représentation symbolique a aussi un autre avantage concernant l'étude de la stabilité des systèmes. En effet, il existe de nombreux critères de stabilité (par exemple le critère de Routh) qui sont fondés, le plus souvent, sur la position des pôles dans le plan complexe. Les pôles sont les solutions de la fonction polynôme Q d'une fonction de transfert H telle que $H = P/Q$.

Dans le cadre de la validation de modèles Simulink, nous pouvons définir une sémantique basée sur la représentation symbolique des signaux. L'analyse statique utilisant cette sémantique permettra de calculer, pour tous les signaux du modèle, les fonctions de transferts associées. L'étude de la stabilité des modèles sera alors obtenue par un post traitement, par exemple [GL04, GL06], sur le résultat de l'analyse. L'objectif d'une analyse statique des modèles Simulink fondée sur la représentation symbolique des signaux est d'étudier la stabilité d'un ensemble de filtres. Pour expliciter cette idée, prenons comme exemple le système de la pédale dont le modèle Simulink est donné à la figure 7.5(a). Nous rappelons que ce système modélise un système mécanique composé d'une masse, d'un ressort et d'un amortisseur. Supposons que les paramètres du système que sont le coefficient d'élasticité du ressort e et le coefficient d'amortissement a varient légèrement au cours du temps, par exemple, cette variation est due à l'usure. Nous obtenons alors un modèle dont les e et a varient dans les intervalles $[e - \epsilon_e, e + \epsilon_e]$ et $[a - \epsilon_a, a + \epsilon_a]$. Les valeurs ϵ_e et ϵ_a représentent la plus grande variation qui peut affecter les paramètres e et a respectivement. Grâce à une analyse statique, nous pourrions calculer la fonction de transfert de ce système avec

des paramètres intervalles. Nous pourrions ainsi estimer la stabilité d'un ensemble de systèmes où chacun d'eux représenterait un paramétrage particulier de e et a .

8.3 Propriétés temporelles

La motivation initiale du travail effectué lors de cette thèse était d'étudier l'influence des approximations numériques sur le processus de contrôle des logiciels embarqués. Plus précisément, cette étude considère le graphe de flot de contrôle d'un modèle Simulink. A chaque nœud de ce graphe, nous étudions si le résultat mathématique (c'est-à-dire la spécification) et le résultat de la simulation coïncident. Nous appelons cette étude "analyse de la divergence du flot de contrôle".

Les traitements numériques des programmes induisent des variations plus ou moins importantes par rapport au modèle mathématique. Par exemple, l'étude mathématique d'un filtre donne un temps de réponse de m millisecondes tandis que l'implémentation de ce filtre induit un temps de réponse de $m + 0.002$, qui est issu des erreurs d'arrondi. Dans le cas d'un système de contrôle, l'utilisation d'un tel filtre va alors ajouter un délai dans la prise de décision. L'étude de la divergence de flot de contrôle est alors un moyen de mesurer une telle différence entre le modèle mathématique et le modèle Simulink.

Le calcul des invariants, c'est-à-dire des intervalles de variation d'un signal dans le modèle Simulink, par notre analyse statique est la première étape pour étudier la divergence de flots de contrôle. En effet, les invariants nous offrent toutes les informations nécessaires pour comparer les comportements liés à la simulation à ceux issus des mathématiques. Néanmoins, la difficulté d'une telle étude est une possible explosion combinatoire. L'analyse de la divergence du flot de contrôle nécessite de connaître les comportements mathématiques et ceux de la simulation. Si au niveau d'un nœud du graphe de flot de contrôle il y a une séparation des deux comportements, l'analyse doit alors poursuivre en considérant ceux-ci comme disjoints. La problématique est de trouver alors un instant où les deux comportements vont à nouveau concorder. Il faut remarquer que cet instant peut ne pas exister. De plus, la séparation des comportements entraîne une quantité d'informations importante nécessaire à la détection de la concordance des comportements. Et, cette quantité d'informations peut augmenter de façon exponentielle.

Pour le moment, la méthode qui sera utilisée pour mener à bien cette analyse n'est pas encore clairement déterminée. Cependant, la définition d'une analyse statique disjonctive semble nécessaire. La disjonction se situe au niveau des expressions conditionnelles. En effet la sémantique abstraite de ces expressions conduit à faire l'union de la branche "vraie" et de la branche "faux". Dans l'analyse statique disjonctive, nous gardons ces deux éléments séparés ce qui engendre une multiplication des valeurs.

Nous pensons également nous inspirer de [MR05, RM07] qui définissent un domaine abstrait de partitionnement de traces. Nous pouvons également utiliser les résultats de [SISG06] qui définit un critère métrique pour la fusion de valeurs disjointes permettant ainsi de limiter l'explosion combinatoire.

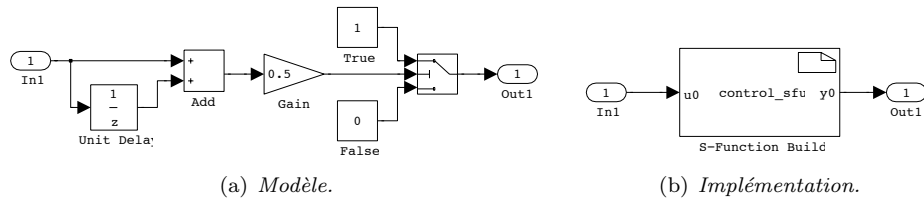
8.4 Validation multi-niveaux

Une des difficultés dans l'activité de validation est de s'assurer que les propriétés vérifiées à un certain niveau du cycle de développement, par exemple les spécifications, sont également vérifiées aux niveaux inférieurs, par exemple l'implémentation. Plus précisément le passage d'un niveau à un autre, par exemple du code source au code machine, est réalisé par un ensemble de transformations qui peuvent affecter la sémantique. Par conséquent, les propriétés validées à un niveau ne le sont peut être plus à un autre [Riv04].

Dans le cas des modèles Simulink, l'analyse statique présentée dans ce document valide les comportements numériques par rapport au modèle mathématique. Néanmoins, les spécifications ne prennent pas en compte les détails d'implémentation qui peuvent avoir un impact important sur ces propriétés. Prenons par exemple, une implémentation sur une architecture DSP (*Digital*

Signal Processor) ne possédant pas d'opération de division "matérielle", par exemple le processeur TMS320 C3X. Il est alors nécessaire d'utiliser un algorithme, par exemple basé sur une méthode Newton [Jed06], pour calculer la division de deux nombres. Du point de vue de la précision numérique, il y a un risque non négligeable que les erreurs d'arrondi soient plus importantes que celles introduites par une division "matérielle". La conséquence est un possible impact sur les comportements du système. Une nouvelle validation des propriétés doit être effectuée.

Un trait intéressant du langage Simulink est sa capacité à communiquer avec des autres langages tels que Matlab, C/C++, Ada, Fortran. Cette interopérabilité est réalisée grâce aux *S-functions*. Ce mécanisme permet la définition de blocs Simulink à l'aide d'un des langages mentionnés précédemment. Par conséquent, les *S-functions* permettent de passer du niveau de la spécification au niveau de l'implémentation. Dans un modèle Simulink, la substitution d'un ensemble de blocs par une *S-function* permet de simuler et ainsi de tester que le système implémenté a les mêmes comportements que le modèle. Par exemple, le modèle de la figure 8.3(a) peut être remplacé par le modèle de la figure 8.3(b). Le bloc **S-Function Builder** de la figure 8.3(b) représente une *S-function* automatiquement générée à partir du code source, en langage C, donné à la figure 8.3(c).



```
double moyenne (double a, double b) {
    double deux = 2.0;
    double somme = x + y;
    double resultat = somme / deux;

    return resultat;
}

int control (double in) {
    static double udelay = 0.0;
    double m = 0.0;
    int resultat = 0;

    m = moyenne (in, udelay);
    resultat = m >= 0.01 ? 1 : 0;

    return resultat;
}
```

(c) Code source de l'implémentation.

FIG. 8.3 – Modèle Simulink et une implémentation possible en C.

L'utilisation des *S-functions* permet alors d'appliquer une analyse statique des implémentations des logiciels embarqués tout en bénéficiant de la modélisation de l'environnement physique. Il en résulte une analyse statique plus précise que celle faisant des hypothèses sur l'environnement physique. L'un des grands avantages de cette méthodologie est une réutilisation importante des modèles des environnements physiques. Ces modèles demandent, en général, une grande expertise et un temps de conception important pour être fidèles à l'environnement physique.

La difficulté de valider une propriété sur différentes représentations d'un programme est liée à l'identification des éléments similaires dans les deux représentations. Nous prenons comme exemple la fonction *moyenne* donnée à la figure 8.3(c). Nous supposons qu'une analyse statique nous a

```

        .globl _moyenne
_moyenne:
        mflr r0
        fadd f2,f1,f2
        bcl 20,31,"L00000000001$pb"
"L00000000001$pb":
        mflr r10
        mtlr r0
        addis r2,r10,ha16(LC0-"L00000000001$pb")
        lfd f1,lo16(LC0-"L00000000001$pb")(r2)
        fmul f1,f2,f1
        blr
        .subsections_via_symbols

```

FIG. 8.4 – Traduction en assembleur PPC (Power PC) de la fonction moyenne.

fourni les invariants P_c à chaque point de contrôle c . Prenons maintenant le programme assembleur, donné à la figure 8.4, issu de la compilation du programme C. En faisant une analyse statique, nous obtenons des invariants $Q_{c'}$ pour chaque point de contrôle c' . La difficulté est alors de faire correspondre pour chaque point de contrôle c les P_c avec les $Q_{c'}$. Par exemple, la division par 2 dans le code source a été remplacée par un décalage binaire et une multiplication dans le code assembleur. Cet exemple met en évidence la difficulté de suivre les transformations de code, qui ont alors un impact sur les propriétés calculées. Dans le cas de la génération automatique de code à partir des modèles Simulink, le problème est le même³. Il est accentué dans le cas d'une génération "à la main" puisque le code n'a pas nécessairement suivi la structure du modèle.

Dans le cas de la validation des logiciels embarqués, en utilisant les modèles Simulink, nous pouvons relâcher la contrainte de faire correspondre les invariants à chaque point de contrôle. En effet, nous pouvons uniquement observer si les sorties de l'implémentation du logiciel embarqué, c'est-à-dire ses décisions, sont conformes à celles obtenues à partir du modèle Simulink. Par exemple, nous pouvons uniquement faire correspondre les sorties du modèle de la figure 8.3(a) et de l'implémentation de la figure 8.3(b). Nous obtenons ainsi une validation sur plusieurs niveaux de développement qui se focalise uniquement sur les fonctions des systèmes. La différence avec l'approche commune de la validation multi-niveaux est la validation du processus de transformation. Par exemple, si nous validons un compilateur, c'est-à-dire qu'il génère un code dont la sémantique est conforme à la sémantique du code source, nous garantissons ainsi la préservation de propriétés. Cependant, cette activité est très complexe. Nous proposons une version plus pragmatique qui consiste uniquement à s'intéresser aux résultats en s'abstrayant des moyens mis en œuvre pour y arriver. Une implémentation et une spécification Simulink seront validées si elles produisent des sorties équivalentes aux mêmes entrées.

³Le coût important du module de génération de code de Simulink nous empêche de donner un exemple de telles transformations

Le remplacement des éléments mécaniques par des composants électromécaniques induit des techniques nouvelles de réalisation des systèmes embarqués. La conception assistée par ordinateur et en particulier, la conception fondée sur les modèles permet de contenir la complexité croissante de ces systèmes. L'outil commercial Matlab/Simulink est un des précurseurs dans ce domaine. Grâce à la simulation numérique, les concepteurs peuvent estimer les comportements des modèles et détecter très tôt dans le cycle de développement des aberrations. Néanmoins, dans le cas de systèmes embarqués critiques de contrôle, c'est-à-dire des systèmes dont les défaillances peuvent avoir des conséquences catastrophiques, la simulation numérique n'est pas suffisante car elle est comparable à l'activité de tests des logiciels. Le test est généralement incomplet puisque tester un programme pour l'ensemble de ses entrées est infaisable en pratique.

Le principal atout du langage Simulink est d'offrir, pour la spécification des systèmes embarqués, un cadre unifié pour représenter les parties logicielles ainsi que les environnements d'exécution. Dans le domaine de la vérification des logiciels embarqués, il est nécessaire de prendre en compte tous les éléments des systèmes embarqués pour être le plus fidèle possible aux conditions réelles de fonctionnement. La méthode de validation doit alors prendre en compte une hétérogénéité mathématique dans ces systèmes. En général, l'environnement physique est composé d'éléments qui évoluent continûment dans le temps (par exemple la température d'une pièce, la vitesse d'une automobile, etc.) alors que le logiciel élément d'un composant électronique fonctionne à une certaine cadence.

L'analyse statique, que nous avons définie au cours de cette thèse, est capable de prendre en compte l'hétérogénéité mathématique des modèles hybrides Simulink. Elle permet de valider les propriétés numériques de ceux-ci grâce à l'utilisation de domaines numériques abstraits conçus spécifiquement dans cet objectif. De plus, elle prend en compte tous les éléments composant les systèmes embarqués, c'est-à-dire l'environnement, le logiciel mais également les capteurs et les actionneurs.

Le domaine des formes de Taylor, défini à la section 5.1, est une extension de l'arithmétique d'intervalles. Il permet de prendre en compte les relations entre les variables du modèle et ainsi il produit des résultats plus précis. Ce domaine permet d'adapter les algorithmes d'intégration numérique à l'arithmétique d'intervalles. Ces algorithmes sont l'élément central de la sémantique des modèles à temps continu de Simulink. De plus, l'utilisation d'algorithmes d'intégration numérique garantis permet de calculer une sur-approximation des comportements des modèles à temps continu, qui est sûre de contenir le résultat mathématique. Ces deux résultats permettent d'obtenir un critère de correction évaluant l'influence de la discrétisation temporelle (c'est-à-dire l'échantillonnage) sur le modèle.

Le domaine des nombres flottants avec erreurs différentiées, défini à la section 5.2, permet de mesurer la qualité numérique d'un calcul en précision finie. Il mesure la distance entre le résultat mathématique et le résultat en arithmétique flottante d'un même calcul. La combinaison du domaine de nombres flottants avec erreurs avec le domaine des formes de Taylor permet d'accroître la précision de l'estimation des erreurs d'arrondi. De plus, le domaine des nombres flottants avec erreurs différentiées offre un critère de correction des modèles à temps discret de Simulink. En effet, la mesure de la précision numérique donne une indication importante sur le choix des algorithmes numériques utilisés dans les logiciels embarqués.

Le domaine des séquences, défini à la section 5.3, rend possible la représentation de manière finie des ensembles de simulations d'une durée potentiellement infinie. L'abstraction des séquences est fondée sur une fonction de partition des ensembles. Grâce à cette fonction, nous avons la possibilité de nous adapter facilement à différents modèles Simulink et ainsi de définir des analyses statiques précises. En effet, le nombre de parties d'une partition induit une notion de précision représentée par la taille des partitions.

Un autre trait important de l'analyse statique des modèles Simulink est la mise en avant, au chapitre 6, des éléments d'un système embarqué, qui sont généralement ignorés lors de l'activité de validation. Plus précisément, les capteurs et les actionneurs, qui sont des éléments importants de ces systèmes, sont modélisés dans notre sémantique des modèles hybrides Simulink. Nous sommes alors capable d'étudier les propriétés numériques des spécifications des systèmes embarqués en prenant en compte tous les éléments qui les composent.

Au final, nous obtenons un critère de correction des comportements numériques des modèles Simulink, c'est-à-dire la distance entre le résultat mathématique et le résultat de la simulation. Ce critère permet de valider les comportements d'un modèle numérique vis-à-vis de sa spécification (c'est-à-dire sa description mathématique) en prenant en compte tous les éléments composant un système embarqué de contrôle. Nous sommes ainsi capable de détecter, dans les premières étapes du cycle de développement, une inconsistance dans le modèle, en particulier, liée aux imprécisions numériques. En automatique, il existe des outils d'étude des systèmes qui permettent de garantir des propriétés comme la stabilité. Néanmoins, l'utilisation d'algorithmes numériques et d'une arithmétique en précision finie engendrent des erreurs qui peuvent avoir une influence sur ces propriétés et ces erreurs ne sont pas, en général, prises en compte dans ces outils d'étude.

Nous avons également mis en évidence au chapitre 8, un certain nombre de travaux futurs autour de la validation du langage de spécification Simulink. En particulier l'extension du langage Simulink par des automates finis est une piste importante en vue de l'application de notre analyse statique sur des cas industriels. En effet, la modélisation des processus de contrôle par des automates est très courante dans l'industrie. Les composantes des logiciels embarqués sont réparties en deux groupes : le traitement des données (langage Simulink) et le processus de contrôle (langage Stateflow). Nous avons principalement traité le premier des deux groupes lors de cette thèse.

La particularité des modèles Simulink est leur étroite relation avec les éléments de l'automatique ou de la théorie du traitement du signal. Comme nous l'avons vu à la section 8.2, nous pouvons appliquer d'autres outils mathématiques issus de ces théories afin d'étudier les modèles Simulink. L'analyse fréquentielle est l'outil incontournable lié à l'étude des systèmes en théorie du traitement du signal. Il est très adapté à l'étude des systèmes linéaires. Nous pouvons grâce à cet outil étudier la stabilité des modèles Simulink.

L'analyse statique de modèles Simulink, que nous avons définie dans cette thèse, conduit à la validation de propriétés temporelles inhérentes à la nature de ces modèles. Il est évident que l'étude de systèmes évoluant dans le temps s'intéresse aux comportements temporels de ceux-ci. L'analyse des propriétés numériques des modèles Simulink permet d'étudier l'influence des approximations sur le processus de décision. Une analyse statique capable d'étudier cette propriété serait très souhaitable pour augmenter la confiance dans les logiciels embarqués.

L'analyse statique de spécifications n'est pas suffisante pour garantir le "bon" fonctionnement des logiciels embarqués. Un ensemble de transformations manuelles ou automatiques introduit de nouvelles approximations ou des changements sémantiques. Le résultat de ces transformations, c'est-à-dire des implémentations de logiciels embarqués, peut dévier légèrement des comportements définis par la spécification. Il est alors nécessaire de valider l'ensemble des étapes de conception afin

de garantir la préservation des propriétés. La difficulté est alors de communiquer les informations de validation d'un niveau à un autre du cycle de développement. Nous pensons que l'outil Simulink permet de faciliter cette communication.

Nous avons développé une analyse statique de modèles Simulink, c'est-à-dire une analyse statique des spécifications de systèmes embarqués de contrôle. Nous avons ainsi montré l'intérêt d'appliquer cette méthode dans les hauts niveaux des cycles de développement offrant un moyen de valider les logiciels embarqués en prenant en compte leur environnement d'exécution. Nous obtenons ainsi des garanties supplémentaires sur la sécurité ainsi que des résultats plus proches du fonctionnement réel de ces logiciels.

Bibliographie

- [ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata : An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems I*, number 736 in LNCS, 1993.
- [ALSU06] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers : Principles, Techniques, and Tools*. Addison Wesley, second edition, 2006.
- [App98] Andrew W. Appel. *Modern Compiler Implementation : In ML*. Cambridge University Press, 1998.
- [ASK04] Aditya Agrawal, Gyula Simon, and Gabor Karsai. Semantic Translation of Simulink/Stateflow models to Hybrid Automata using Graph Transformations. In *Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT'04)*, number 109 in ENTCS, 2004.
- [BBC⁺97] Jean-Claude Bajard, Olivier Beaumont, Jean-Marie Chesneaux, Marc Daumas, Jocelyne Erhel, Dominique Michelucci, Jean-Michel Muller, Bernard Philippe, Nathalie Revol, Jean-Louis Roch, and Jean Vignes. *Qualité des Calculs sur Ordinateurs. Vers des arithmétiques plus fiables ?* Masson, 1997.
- [BCC⁺03] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A Static Analyzer for Large Safety-Critical Software. In *Programming Language Design and Implementation (PLDI'03)*, ACM, pages 196–207. ACM Press, 2003.
- [BCE⁺03] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and R. de Simone. The Synchronous Languages 12 Years Later. In *IEEE*, volume 91, 2003.
- [BCKM94] Christian H. Bischof, Alan Carle, Peyvand M. Khademi, and Andrew Mauer. The ADIFOR 2.0 System for the Automatic Differentiation of Fortran 77 Programs. Technical Report MCS-P481-1194, 1994.
- [Ben02] Messaoud Benidir. *Théorie et Traitement du Signal : Représentation des signaux et des systèmes*, volume 1. Dunod, 2002.
- [Ber05] Julien Bertrane. Static Analysis by Abstract Interpretation of the Quasi-synchronous Composition of Synchronous Programs. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'05)*, volume 3385 of LNCS, pages 97–112, 2005.
- [Ber06] Julien Bertrane. Proving the Properties of Communicating Imperfectly-Clocked Synchronous Systems. In *International Static Analysis Symposium (SAS'06)*, volume 4134 of LNCS, 2006.

- [BG92] Gérard Berry and Georges Gonthier. The ESTEREL Synchronous Programming Language : Design, Semantics, Implentation. *Science of Computer Programming*, 19(2) :87–152, 1992.
- [BHN02] Christian H. Bischof, Paul D. Hovland, and Boyana Norris. Implementation of Automatic Differentiation Tools. In *Partial Evaluation and Semantics-Based Program Manipulation (PEPM'02)*. ACM Press, 2002.
- [Bir67] Garrett Birkhoff. *Lattice Theory*. American Mathematical Society, 1967.
- [BJT99] Frédéric Besson, Thomas P. Jensen, and Jean-Pierre Talpin. Polyhedral Analysis for Synchronous Languages. In *International Symposium on Static Analysis (SAS'99)*. Springer-Verlag, 1999.
- [BM06] Olivier Bouissou and Matthieu Martel. GRKLib : a Guaranteed Runge-Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics*. IEEE, 2006.
- [BM08a] Olivier Bouissou and Matthieu Martel. A Hybrid Denotational Semantics for Hybrid Systems. In *European Symposium on Programming (ESOP'07)*, volume 4960 of *LNCS*. Springer, 2008.
- [BM08b] Olivier Bouissou and Matthieu Martel. Abstract Interpretation of the Physical Inputs of Embedded Programs. In *Verification, Model Checking and Abstract Interpretation (VMCAI'08)*, *LNCS*. Springer, 2008.
- [Boa96] Ariane 5 Inquiry Board. Ariane 5 – flight 501 failure. Technical report, European Space Agency, 1996.
- [BRMO97] Christian H. Bischof, Lucas Roh, and Andrew J. Mauer-Oats. ADIC : an Extensible Automatic Differentiation Tool for ANSI-C. *Software Practice and Experience*, 27(12) :1427–1456, 1997.
- [CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation : a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Principles of Programming Languages (POPL'77)*. ACM Press, 1977.
- [CC79] Patrick Cousot and Radhia Cousot. Constructive Versions of Tarski's Fixed Point Theorems. *Pacific Journal of Mathematics*, 81(1) :43–57, 1979.
- [CC92] Patrick Cousot and Radhia Cousot. Abstract Interpretation Frameworks. *Journal of Logic and Computation*, 2(4), 1992.
- [CCF⁺05] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. The ASTRÉE Analyser. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*. Springer Verlag, 2005.
- [CCM⁺03a] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, and Stavros Tripakis. Translating Discrete-Time Simulink to Lustre. In *ACM Transaction on Embedded Computing Systems*, 2003.
- [CCM⁺03b] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, Stavros Tripakis, and Peter Niebert. From Simulink to SCADE/Lustre to TTA : a Layered Approach for Distributed Embedded Applications. In *Languages, Compilers and Tools for Embedded Systems (LCTECS'03)*, 2003.
- [CH78] Patrick Cousot and Nicolas Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Principles of Programming Languages (POPL'78)*. ACM Press, 1978.
- [Che95] Jean-Marie Chesneaux. *L'arithmétique Stochastique et le Logiciel CADNA*. Habilitation à diriger des recherches, Université Pierre et Marie Curie (Paris 6), 1995.
- [CHP06] Jean-Louis Colaço, Grégoire Hamon, and Marc Pouzet. Mixing Signals and Modes in Synchronous Data-flow Systems. In *International Conference of Embedded Software (EMSOFT'06)*. ACM Press, 2006.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithm*. MIT Press, 2001.

- [CM06] Alexandre Chapoutot and Matthieu Martel. Abstract Frequency Analysis of Synchronous Systems. Poster session and short paper, 2006.
- [CM07a] Alexandre Chapoutot and Matthieu Martel. Différentiation automatique et formes de Taylor en analyse statique de programmes numériques. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'07)*, 2007.
- [CM07b] Alexandre Chapoutot and Matthieu Martel. Frequency Analysis of Data-flow Programs by Abstract Interpretation. Technical Report DRT/LIST/DTSI/SOL/07-169, Commissariat à l'Energie Atomique (CEA), 2007.
- [CM08a] Alexandre Chapoutot and Matthieu Martel. Différentiation automatique et formes de Taylor en analyse statique de programmes numériques. *Technique et Science Informatiques (TSI) numéro spécial AFADL'07*, 2008.
- [CM08b] Alexandre Chapoutot and Matthieu Martel. Static Analysis of Simulink Programs. In *Model-driven High-level Programming of Embedded Systems (SLA++P'08)*, ENTCS, 2008.
- [Con04] Charles Consel. Generative Programming from a Domain-Specific Language Viewpoint. In *Unconventional Programming Paradigms (UPP'04)*, 2004.
- [Cou81] Patrick Cousot. Semantic Foundations of Program . In *Program Flow Analysis : Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., 1981.
- [Cou05] Patrick Cousot. Integrating Physical Systems in the Static Analysis of Embedded Control Software. In *Asian Symposium on Programming Languages and Systems (APLAS'05)*. Springer Verlag, 2005.
- [Cou07] Patrick Cousot. Proving the Absence of Run-Time Errors in Safety-Critical Avionics Code. In *International Conference of Embedded Software (EMSOFT'07)*. ACM press, 2007.
- [CP96] Paul Caspi and Marc Pouzet. Synchronous Kahn Networks. In *International Conference on Functional Programming (ICFP'96)*, ACM Press, 1996.
- [CP01] Pascal Cuoq and Marc Pouzet. Modular Causality in a Synchronous Stream Language. In *European Symposium on Programming Languages and Systems (ESOP'01)*, pages 237–251. Springer-Verlag, 2001.
- [CPHP87] Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John A. Plaice. LUSTRE : a Declarative Language for Real-Time Programming. In *Principles of Programming Languages (POPL'87)*. ACM Press, 1987.
- [Dij68] Edsger W. Dijkstra. Letters to the editor : go to statement considered harmful. *Communications of the ACM*, 11(3) :147–148, 1968.
- [GAO92] GAO. Patriot Missile Defense : Software Problem Led to System Failure at Dhahran, Saudi Arabia. Technical report, U.S. General Accounting Office, 1992.
- [GBGM91] Thierry Gauthier, Michel Le Borgne, Paul Le Guernec, and Claude Le Maire. Programming Real Time Applications with SIGNAL. In *IEEE*, volume 79, 1991.
- [GD06] Alain Le Guennec and Bernard Dion. Bridging UML and Safety-Critical Software Development Environments. In *Embedded Real Time Software (ERTS'06)*, 2006.
- [GGB08] Abdoulaye Gamatié, Thierry Gautier, and Loïc Bensard. An Interval-Based Solution for Static Analysis in the SIGNAL Language. In *Engineering of Computer-Based Systems (ECBS'08)*, 2008.
- [GL04] Steph Graillat and Philippe Langlois. Pseudozer Set Decides on Polynomial Stability. In *Mathematical Theory of Networks and Systems (MTNS'04)*, 2004.
- [GL06] Steph Graillat and Philippe Langlois. Pseudozero Set of Interval Polynomials. In *Applied Computing (SAC'06)*. ACM Press, 2006.
- [Gol91] David Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, 23(1) :5–48, 1991.

- [GP06] Eric Goubault and Sylvie Putot. Static Analysis of Numerical Algorithms. In *International Static Analysis Symposium (SAS'06)*, LNCS. Springer Verlag, 2006.
- [Gri00] Andreas Griewank. *Evaluating Derivatives : Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, 2000.
- [Hal93] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.
- [Hal94] Nicolas Halbwachs. About Synchronous Programming and Abstract Interpretation. In *International Symposium on Static Analysis (SAS'94)*, volume 864 of *LNCS*. Springer-Verlag, 1994.
- [Ham05] Grégoire Hamon. A Denotational Semantics for Stateflow. In *International Conference of Embedded Software (EMSOFT'05)*. ACM, 2005.
- [HAP05] Laurent Hascoët and Mauricio Araya-Polo. The Adjoint Data-Flow Analyses : Formalization, Properties, and Applications. In *Automatic Differentiation : Applications, Theory, and Tools (AD'04)*, LNCS. Springer, 2005.
- [Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2nd edition, 2002.
- [HJM⁺02] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, George C. Necula, Grégoire Sutre, and Westley Weimer. Temporal-Safety Proofs for Systems Code. In *International Conference on Computer Aided Verification (CAV'02)*. Springer-Verlag, 2002.
- [HJMS03] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Software verification with Blast. In *International Workshop on Model Checking of Software (SPIN)*, volume 2648. LNCS, Springer-Verlag, 2003.
- [HKSP03] Thomas A. Henzinger, Christoph Kirsch, Marco Sanvido, and Wolfgang Pree. From Control Models to Real-Time Code Using Giotto. *IEEE Control Systems Magazine*, 23(1) :50–64, 2003.
- [HNW93] Ernst Hairer, Syvert P. Norsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I : Nonstiff Problems*, volume 8 of *Computational Mathematics*. Springer-Verlag, 2nd edition, 1993.
- [HP05] Diederich Hinrichsen and Anthony J. Pritchard. *Mathematical Systems Theory I : Modelling, State Space Analysis, Stability and Robustness*. Number 48 in Texts in Applied Mathematics. Springer, 2005.
- [HPSS87] David Harel, Amir Pnueli, Jeanette P. Schmidt, and Rivi Sherman. On the Formal Semantics of Statecharts. In *Logic in Computer Science (LICS'87)*. IEEE Computer Society Press, 1987.
- [HR99] Nicolas Halbwachs and Pascal Raymond. Validation of Synchronous Reactive Systems : from Formal Verification to Automatic Testing. In *Asian Computing Science Conference (ASIAN'99)*, volume 1742 of *LNCS*. Springer Verlag, 1999.
- [HR07] Grégoire Hamon and John Rushby. An Operational Semantics for Stateflow. *Journal on Software Tools for Technology Transfer (STTT)*, 9(5–6) :447–456, 2007.
- [IEC01] IEC Working Group. *IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems (7 parts)*. International Electrotechnical Commission, 2001.
- [IEE85] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE, New York, 1985.
- [Jac89] Leland B. Jackson. *Digital Filter and Signal Processing*. Kluwer Academic Publishers, 1989.
- [Jea00] Bertrand Jeannot. *Partitionnement dynamique dans l'analyse de relations linéaires et applications à la vérification de programmes synchrones*. PhD thesis, Institut National Polytechnique de Grenoble, 2000.

- [Jed06] Franck Jedrzejewski. *Introduction aux méthodes numériques*. Springer, 2nd edition, 2006.
- [Jen95] Thomas P. Jensen. Clock Analysis of Synchronous Dataflow Programs. In *Partial Evaluation and Semantic-Based Program Manipulation (PEPM'95)*. ACM Press, 1995.
- [JHR99] Bertrand Jeannet, Nicolas Halbwachs, and Pascal Raymond. Dynamic Partitioning in Analyses of Numerical Properties. In *International Static Analysis Symposium (SAS'99)*. Springer Verlag, 1999.
- [JKDW01] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis*. Springer, 2001.
- [Kah74] Gilles Kahn. The Semantics of a Simple Language for Parallel Programming. In *International Federation of Information Processing (IFIP'74)*, 1974.
- [Kra92] Peter Kraniuskas. *Transforms in Signals and Systems*. Addison-Wesley, 1992.
- [Lan99] Philippe Langlois. Automatic Linear Correction of Rounding Errors. Technical report, INRIA, 1999.
- [Lim07] Maxime Lim. Parser de fichiers MDL pour l'analyse statique. Rapport de stage, 2007.
- [Mar05] Matthieu Martel. An Overview of Semantics for the Validation of Numerical Programs. In *Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *LNCS*. Springer Verlag, 2005.
- [Mar06] Matthieu Martel. Semantics of Roundoff Error Propagation in Finite Precision Computations. *Journal of Higher Order and Symbolic Computation*, 19(1), 2006.
- [MH06] Russel Miles and Kim Hamilton. *Learning UML 2*. O'Reilly, 2006.
- [Min04] Antoine Miné. Relational Abstract Domains for the Detection of Floating-Point Run-Time Errors. In *European Symposium on Programming (ESOP'04)*, volume 2986 of *LNCS*, pages 3–17. Springer, 2004.
- [Min06] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1) :31–100, 2006.
- [Mon07] David Monniaux. The Pitfalls of Verifying Floating-Point Computations. *Transactions on programming languages and systems (TOPLAS)*, 30(3), 2007.
- [Moo79] Ramon E. Moore. *Methods and Applications of Interval Arithmetic*. Studies in Applied Mathematics. SIAM, 1979.
- [MR05] Laurent Mauborgne and Xavier Rival. Trace partitioning in abstract interpretation based static analyzers. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 5–20. Springer-Verlag, 2005.
- [Mye79] Glenford J. Myers. *The Art of Software Testing*. John Wiley & Sons, 1979.
- [Neu01] Arnold Neumaier. *Introduction to Numerical Analysis*. Cambridge University Press, 1st edition, 2001.
- [NJC99] Ned S. Nedialkov, Kenneth R. Jackson, and George F. Corliss. Validated Solutions of Initial Value Problems for Ordinary Differential Equations. *Applied Mathematics and Computation*, 105(1) :21–68, 1999.
- [NNH99] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 1999.
- [PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, 1992.
- [Ral81] Louis B. Rall. *Automatic Differentiation : Techniques and Applications*. Springer, 1981.

- [Rev01] Nathalie Revol. Introduction to Interval Arithmetic. Technical Report RR2001-41, LIP, École Normale Supérieure de Lyon and INRIA, 2001.
- [Riv04] Xavier Rival. Invariant Translation-Based Certification of Assembly Code. *International Journal on Software and Tools for Technology Transfer*, 6(1) :15–37, 2004.
- [RM07] Xavier Rival and Laurent Mauborgne. The Trace Partitioning Abstract Domain. *Transactions on Programming Languages and Systems (TOPLAS)*, 29(5), 2007.
- [Sch02] Bernd S. W. Schroder. *Ordered Sets : An Introduction*. Springer, 2002.
- [SISG06] Sriram Sankaranarayanan, Franjo Ivancic, Ilya Shlyakhter, and Aarti Gupta. Static Analysis in Disjunctive Numerical Domains. In *Static Analysis Symposium (SAS '06)*, volume 4134 of *LNCS*, pages 3–17, 2006.
- [SM04] Tim Schattkowsky and Wolfgang Müller. Model-Based Design of Embedded Systems. In *Object-Oriented Real-Time Distributed Computing (ISORC'04)*, 2004.
- [Som06] Ian Sommerville. *Software Engineering*. Addison-Wesley, 8th edition, 2006.
- [Son98] Edouardo D. Sontag. *Mathematical Control Theory : Deterministic Finite Dimensional Systems*. Number 6 in Texts in Applied Mathematics. Springer, Second edition, 1998.
- [SSC⁺04] Norman Scaife, Christos Sofronis, Paul Caspi, Stavros Tripakis, and Florence Marainchi. Defining and Translating a "safe" subset of Simulink/Stateflow into LUSTRE. In *International Conference of Embedded Software (EMSOFT'04)*, ACM, pages 259–268. ACM Press, 2004.
- [Tiw02] Ashish Tiwari. Formal Semantics and Analysis Methods for Simulink Stateflow models. Technical report, SRI International, 2002.
- [Vig96] Jean Vignes. A Survey of the CESTAC Method. In *Real Numbers and Computer Conference (RNC'96)*, 1996.
- [Win93] Glynn Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

Abstract Simulation : a Static Analysis of Simulink Models

Alexandre Chapoutot

Laboratoire Modélisation et Analyse de Systèmes en Interaction,
CEA - Centre de Saclay, F-91191 Gif-sur-Yvette Cedex

Abstract In regard to the growth of complexity of embedded systems, software tools are needed at design-time. Simulink¹ and Lustre/Scade² are the main industrial tools used in this context. Despite of the numerous features added in both tools, such as simulation, test or code generation, Simulink is more often used due to its important system design expressiveness. It allows us to model and simulate continuous-time and discrete-time systems, as well as a mix of both. For embedded systems, Simulink offers a convenient way to model and specify both the embedded software and the physical environment. The application of formal methods on such specifications is an important challenge for the validation of embedded software. Moreover, applying formal methods sooner in the cycle of development is also an essential industrial challenge in order to reduce the cost of bug fixing.

In this thesis, we define a static analysis by abstract interpretation of Simulink models. This static analysis is called *Abstract Simulation*. The aim of *Abstract Simulation* is to provide a correctness criterion for the executions of Simulink models because they are often used for the validation of systems. However, such simulations are closer to test-based verifications than to formal proofs and, consequently, they do not permit to validate, in regards to the specification, a system. *Abstract Simulation* provides a correctness criterion for numerical behaviors of the Simulink models in the sense that they mimic what happens in the real world.

We assume that the mathematical model encoded as a Simulink model is correct (the physical system is correctly modeled). We aim at automatically and conjointly compute an over-approximation of the mathematical behaviors and the simulation behaviors for all the possible inputs of the model. We can thus estimate the whole imprecision introduced by the simulation, i.e. numerical errors as truncation errors and round-off errors as well as sensor errors like quantization and sampling. The correctness criterion of continuous-time models is given by the distance between numerical integration algorithm (Simulink *solver*) and guaranteed numerical integration algorithm based on Taylor method. The correctness criterion of discrete-time models is given by using the abstract numerical domain of floating-point numbers with errors. Furthermore, *Abstract Simulation* uses two abstract numerical domains, such that the domain of Taylor forms and the domain of floating-point numbers with differentiated errors. The former is a composition of the domain of floating-point numbers with errors and the automatic differentiation techniques. This combination permits to compute a better over-approximation of the rounding errors. *Abstract Simulation* is also based on an abstraction of sequences using set partitioning. These domains allow us to estimate errors introduced by numerical algorithms and by computations during simulations. As a result, our method makes it possible to validate numerical behaviors of embedded systems modeled in Simulink.

¹Trademarks of Mathworks

²Trademarks of Esterel Technologies

Simulation abstraite : une analyse statique de modèles Simulink

Alexandre Chapoutot

Laboratoire Modélisation et Analyse de Systèmes en Interaction,
CEA - Centre de Saclay, F-91191 Gif-sur-Yvette Cedex

Résumé La conception de systèmes embarqués nécessite de plus en plus l'utilisation d'outils logiciels afin de contenir la complexité croissante de ceux-ci. Les deux principaux outils industriels dans ce domaine sont Simulink³ et Lustre/Scade⁴. Ces deux outils possèdent de nombreuses fonctionnalités comme un moteur de simulations, des générateurs de tests ou de code. Cependant, Simulink est, dans la majorité des cas, utilisé pour la conception de systèmes embarqués et ceci parce qu'il a une expressivité plus importante. Il est capable de modéliser et de simuler des systèmes à temps continu, à temps discret et un mélange des deux, c'est-à-dire des systèmes hybrides. Pour la conception des systèmes embarqués, Simulink permet de modéliser l'environnement physique et le logiciel embarqué dans un même formalisme. L'application des méthodes formelles sur de telles spécifications est un défi industriel et scientifique important pour la validation des logiciels embarqués. De plus, l'utilisation de méthodes formelles, au plus tôt dans le cycle de développement, est un challenge essentiel dans l'industrie afin de réduire les coûts liés à la correction de bogues.

Dans cette thèse, nous définissons une analyse statique par interprétation abstraite de modèles Simulink. Nous appelons cette analyse *simulation abstraite*. L'objectif de la *simulation abstraite* est de fournir un critère de correction des comportements numériques des exécutions des modèles Simulink. Ces simulations sont souvent utilisées pour valider les systèmes modélisés, mais elles sont plus proches de l'activité de tests que celle de la preuve. En conséquence, elles ne permettent pas de valider vis-à-vis des spécifications un système modélisé avec Simulink. La *simulation abstraite* fournit un critère de correction dans le sens que les comportements des modèles Simulink représentent au mieux les comportements du monde réel.

Nous supposons que le modèle mathématique, représenté par un modèle Simulink, est correcte vis-à-vis du monde réel. Notre objectif est de calculer automatiquement et conjointement une sur-approximation des comportements mathématiques et des comportements issus de la simulation numérique pour une plage d'entrées possibles. Nous sommes ainsi capable d'estimer l'ensemble des imprécisions introduit par la simulation numérique, c'est-à-dire les erreurs d'arrondi ou les erreurs de troncature liées, par exemple, aux capteurs. Le critère de correction des modèles à temps continu est obtenu en évaluant la distance séparant les résultats des méthodes d'intégration numérique, utilisées par le moteur de simulations, des résultats obtenus par une méthode d'intégration numérique garantie. Le critère de correction des modèles à temps discret est donné par l'utilisation du domaine numérique abstrait des nombres flottants avec erreurs différenciées. Ce nouveau domaine numérique est issu de la combinaison du domaine des flottants avec erreurs et la méthode de différentiation automatique permettant d'avoir une meilleure abstraction des erreurs. Nous définissons également une abstraction d'un domaine des séquences utilisant les partitions d'un ensemble. Nous sommes ainsi en mesure de représenter des simulations infinies d'une manière finie. L'ensemble de ces domaines permet alors d'estimer les erreurs introduites par les traitements numériques présents lors des simulations. Nous obtenons alors une méthode de validation des comportements numériques des systèmes embarqués modélisés en Simulink.

³Marque déposée de Mathworks

⁴Marque déposée de Esterel Technologies