UNIVERSITÉ PARIS IX
UFR SCIENCES DES ORGANISATIONS

Mémoire présenté en vue de l'obtention
de l'Habilitation à Diriger des Recherches

Eléments pour la validation de systèmes numériques intégrés

Matthieu MARTEL

11 juillet 2006

Jury :

Patrick COUSOT (Rapporteur), *Professeur à l'Ecole Normale Supérieure*
Marc DAUMAS (Rapporteur), *Chargé de recherche habilité au CNRS (LIRMM)*
Roberto GIACOBAZZI (Rapporteur), *Professeur à l'Università degli Studi di Verona*
Eric GOUBAULT, *Directeur de recherche au CEA (LIST)*
Daniel KROB (Président), *Directeur de recherche au CNRS (LIX)*
Vangelis PASCHOS (Coordinateur), *Professeur à l'Université Paris IX*

Table des matières

Chapitre 1

Introduction

1.1 Problématique

Il est des domaines où les problématiques industrielles peuvent inspirer des thématiques de recherche plutôt théoriques, procurant au chercheur un double sentiment de satisfaction concernant l'intérêt scientifique et l'utilité pratique de ses travaux. Sous d'austères dénominations telles que systèmes embarqués, systèmes à logiciel prépondérant, ou encore systèmes critiques, se cachent autant de problèmes mathématiquement ardus que d'enjeux industriels et sociétaux : la terminaison d'un programme, l'accessibilité d'un état dans un automate hybride sont des problèmes indécidables en général tandis que le dysfonctionnement potentiel d'un régulateur de vitesse sur une automobile ou l'échec d'une mission spatiale font la une de l'actualité.

1.1.1 Problématique industrielle : validation de systèmes embarqués, intégration et couplage de systèmes

Des systèmes de transport (automobiles, avions) aux grands équipements industriels (centrales électriques, machines-outils), l'informatique permet d'assurer le contrôle de processus de transformation de grandeurs physiques. L'expression "système à logiciel prépondérant", souvent employée pour qualifier ces systèmes, souligne l'importance du processus de régulation et, par conséquent, de l'informatique en leur sein. Par exemple, dans une automobile, de nombreux calculateurs[1] servent à contrôler une kyrielle de processus interdépendants (injection, adhérence au sol, etc.). On considère que le comportement d'une voiture dépend d'environ 4000 paramètres.

La conception de tels systèmes industriels s'appuie sur des processus de développement particuliers, dits processus d'ingénierie système [Mei98, Mei02], parmi lesquels le "cycle en V", représenté dans la figure 1.1, est le plus connu. Typiquement, un système global est décomposé récursivement jusqu'à l'obtention d'objets suffisamment élémentaires pour être réalisé par un certain corps de métier ; il s'agit de la phase d'ingénierie correspondant à la partie gauche de la figure 1.1. Lorsque la taille et l'hétérogénéité d'un système global deviennent des obstacles majeurs à sa conception, on parle aussi de système industriel complexe.

Si la décomposition en sous-systèmes rend abordable la réalisation de ces derniers, elle engendre une difficulté majeure au moment de leurs couplages, lorsqu'il s'agit de constituer le système global. Cette phase d'intégration, qui correspond à la partie droite de la figure 1.1, nécessite

[1] Le terme "calculateur", employé dans un contexte industriel, désigne généralement à la fois un programme informatique et l'ordinateur qui l'exécute.

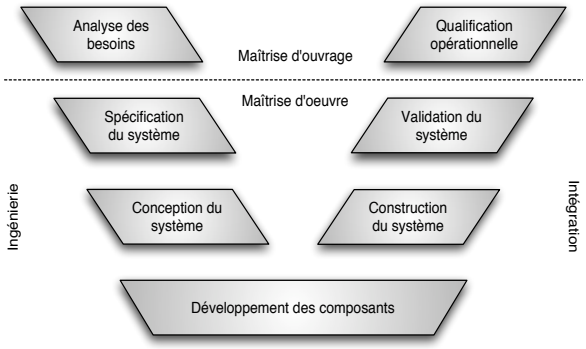

FIG. 1.1 – *Le cycle de développement d'un système.*

de vérifier que les interactions entre des composants hétérogènes et conçus séparément produisent effectivement le comportement global souhaité. Le fait que les sous-systèmes ainsi couplés soient de natures différentes (objets physiques, électroniques, informatiques) et reposent sur des modèles mathématiques éloignés augmente les difficultés rencontrées lors de la validation du système intégré. Dans le domaine du logiciel, l'explosion de la fusée Ariane 5 lors de son premier vol, due à un dysfonctionnement d'un programme initialement conçu pour Ariane 4 est un problème typique d'intégration. Les principales classes de couplages inter-systèmes apparaissent dès lors que l'on identifie les trois grandes classes de systèmes homogènes présents dans un système industriel complexe :

1. *Les systèmes technologiques :* cette catégorie regroupe les systèmes transformant principalement des grandeurs physiques, comme le système d'injection d'un moteur de voiture. Ici, la fonction de transfert est essentiellement une fonction continue telle que la solution d'une équation aux dérivées partielles.
2. *Les systèmes informatiques :* cette catégorie regroupe les systèmes à logiciel prépondérant, dont les entrées sont essentiellement des paramètres discrets et les sorties le résultat d'un traitement informatique.
3. *Les systèmes humains :* les opérateurs sont en effet considérés comme des éléments à part entière d'un système industriel complexe. Même si l'étude précise de leurs comportements relève plutôt des sciences cognitives et s'éloigne de notre champ d'étude, leurs interactions avec les autres catégories de systèmes doivent être prises en compte lors de la phase d'intégration (qualification opérationnelle).

Cette typologie fait apparaître les grandes catégories de couplages inter-systèmes, qui correspondent aux interactions possibles entre systèmes homogènes : on retrouve par exemple les couplages entre systèmes physiques et informatiques, couramment regroupés sous le terme de systèmes hybrides, les couplages entre systèmes informatiques hétérogènes (notamment les problèmes de l'informatique distribuée hétérogène), ou encore la problématique des interfaces homme-système lors des couplages entre systèmes technologiques ou informatiques et systèmes humains.

Dans la suite de ce document, nous nous intéresserons plus particulièrement à deux types de couplages : d'une part ceux entre systèmes informatiques et physiques et d'autre part ceux entre systèmes informatiques et électroniques (processeurs, mémoire, etc.). En effet, au sein des systèmes industriels décrits précédemment, les logiciels embarqués sont des sous-systèmes particuliers utilisés pour réguler le comportement des autres sous-systèmes. Ils influent donc directement

sur les processus physiques et les fonctions globales du système. Par exemple le sous-système correspondant au logiciel de régulation d'un turbo-réacteur influe directement sur les fonctions globales du système "avion". Parce qu'ils ont pour rôle de contrôler les autres sous-systèmes, les logiciels embarqués sont critiques pour le bon fonctionnement de l'ensemble d'un système. Parce qu'ils interagissent fortement avec les autres sous-systèmes, les logiciels embarqués ne peuvent pas être entièrement validés au niveau unitaire, c'est-à-dire sans tenir compte de leurs couplages avec les autres composants du système global.

La validation d'un logiciel embarqué intégré dans son environnement d'exécution, autrement dit en tenant compte de ses interactions avec le système physique qu'il contrôle et l'architecture matérielle sur laquelle il s'exécute, est un thème de recherche tout aussi vaste qu'encore largement inexploré, scientifiquement riche et industriellement important. Ce document a pour objet d'apporter quelques éléments de réponse, récapitulant ainsi les travaux de recherche que j'ai menés dans ce domaine ces dernières années, ainsi que de proposer quelques directions méritant d'être approfondies dans le futur.

1.1.2 Thématique de recherche : validation de calculs en nombres flottants

La validation de logiciels embarqués effectuant des traitements numériques est un thème de recherche situé à l'intersection des principaux problèmes d'intégration système. D'une part, les calculs réalisés par ces programmes permettent de contrôler les processus physiques effectués par les systèmes et, d'autre part, ces calculs (on pourrait dire la sémantique des programmes) dépendent de la façon dont sont représentés les nombres en machine et des algorithmes utilisés dans les processeurs pour implémenter les opérations élémentaires. Ainsi, la validation de tels logiciels fait immédiatement apparaître deux des principales classes de couplages inter-systèmes : ceux avec l'environnement physique dans lequel le système embarqué évolue et ceux avec l'architecture matérielle sur laquelle les programmes sont exécutés. Pour illustrer cela, considérons par exemple le calcul

$$s = \sum_{i=0}^{i<25} \left((10^i + 1) - 10^i\right)$$

réalisé par le programme en langage C suivant :

```
int i;
double x= 10.0, y = 1.0, s=0.0, t;

for (i=0; i<25; i++) {
  t = pow(x,i);
  s = s+((t+y)-t);
}
```

Dans les réels, la valeur de s est égale à 25. En exécutant ce programme, on obtient $s = 20$ sur un ordinateur muni d'un processeur Pentium et $s = 16$ sur un ordinateur muni d'un processeur PowerPC (en utilisant le même compilateur gcc, versions 4.0.2 et 4.0.1 respectivement). L'écart entre ces deux derniers résultats, au demeurant inexacts, est dû à une différence dans la gestion des nombres au sein des unités flottantes[2], pourtant toutes deux conformes à la norme IEEE 754 sur

[2]Sur un processeur Pentium, les résultats intermédiaires des calculs sont conservés sur un grand nombre de bits tant qu'ils restent stockés dans des registres alors qu'ils sont arrondis après chaque opération sur un processeur PowerPC. La norme IEEE 754 n'est pas interprétée de la même manière par les deux fabricants.

laquelle nous reviendrons. Cet exemple permet de mettre en évidence deux faits : les traitements numériques ne peuvent être validés indépendamment des modèles mathématiques et physiques auxquels ils se rattachent ; de plus, ils ne peuvent être validés sans tenir compte de l'architecture matérielle sur laquelle ils sont exécutés.

Tout comme dans l'exemple précédent, les traitements numériques réalisés sur ordinateur sont le plus souvent effectués en utilisant des nombres flottants [Gol91, Knu97, DM97] conformes à la norme IEEE 754 [ANS85, ISO89]. Or, ces derniers sont nécessairement entachés d'erreurs, le résultat de chaque opération devant être approché pour être représenté en mémoire sur un nombre fini de bits. De ce fait, leur arithmétique diffère sensiblement de celle des nombres réels avec lesquels nous sommes habitués à raisonner. Par exemple, en nombres flottants, de nombreuses opérations sont non-associatives, non-distributives ou encore non-inversibles. Des équations parmi les plus élémentaires n'ont pas les mêmes solutions dans les deux arithmétiques : comme le remarquent C. Michel *et al.* [MRL01], en double précision dans la norme IEEE 754,

$$x + 16.0 = 16.0 \iff x \in [-8.88178419700125232e^{-16}, 1.77635683940025046e^{-15}]. \quad (1.1)$$

Les concepteurs de systèmes étant habitués à raisonner dans les nombres réels, on qualifiera d'erreur introduite par les nombres flottants toute différence de comportement entre une implantation et un modèle de référence dans lequel les calculs seraient effectués en utilisant des nombres réels. De ce point de vue, les erreurs dues aux nombres flottants peuvent être classées en deux grandes catégories :

1. Les erreurs dites *objectives* : sont regroupées sous ce terme les erreurs observables lors d'une exécution non-instrumentée d'un programme : dépassements de capacité (underflow et overflow) et opérations non-définies (division par zéro, racine carrée d'un nombre négatif). Lorsqu'elles surviennent de façon inattendue, ces erreurs interrompent généralement l'exécution normale d'un programme (modification d'un drapeau dans le processeur, levée d'exception). Ces erreurs peuvent néanmoins être difficiles à détecter et inexistantes dans le modèle utilisant des nombres réels, comme dans l'extrait de programme suivant proposé par A. Deutch, de Polyspace Technologies[3] :
   ```
   if (x!=0) x = 1/(x*x);
   ```
   Ici, une division par zéro peut survenir si ce calcul est effectué en nombres flottants (si x est proche de zéro, x^2 peut être nul) mais pas s'il est effectué en nombres réels.
2. Les erreurs dites *subjectives* : sont dénommées ainsi les erreurs correspondant à des pertes de précision numérique (les erreurs d'arrondi). Ces erreurs, omniprésentes, ne sont pas observables lors d'une exécution non-instrumentée d'un programme. On les qualifie de subjectives dans la mesure où leur gravité dépend du contexte dans lequel les résultats du calcul sont utilisés : ainsi, une erreur de 10% par rapport au résultat exact peut être négligeable dans certains cas et avoir des conséquences critiques dans d'autres.

Les erreurs d'arrondi sont en général acceptables dans la mesure où le résultat produit par la machine est proche de celui que l'on aurait obtenu en utilisant des nombres réels. Cependant, elles peuvent aussi être amplifiées par propagation, dénuant de sens le résultat d'un calcul. Ces problèmes sont bien connus des numériciens qui disposent de solutions adaptées à leur domaine, le plus souvent fondées sur le calcul des erreurs commises pour des jeux de données représentatifs. Par exemple, dans la librairie d'algèbre linéaire LAPACK 3.0, les routines de test servant à vérifier

[3] Polyspace : `http ://www.polyspace.com`.

la précision des algorithmes numériques occupent 308 770 lignes de code sur un total de 766 953, soit environ 40% de l'ensemble [Hig97].

Le problème de la validation des traitements numériques effectués par des logiciels embarqués se pose avec d'autant plus d'acuité que des calculs en nombres flottants gouvernent aujourd'hui des systèmes critiques. Le recours aux nombres flottants est rendu nécessaire par la complexité croissante des traitements à réaliser. En effet, des logiciels qui dans un passé récent manipulaient seulement des entiers (ou des nombres en virgule fixe), voire seulement des booléens, utilisent de plus en plus souvent les nombres flottants pour piloter des systèmes industriels complexes, et parfois pour anticiper leur évolution. Par exemple, les programmes de contrôle-commande les plus critiques de la gamme Airbus et, en particulier, ceux de l'A380 effectuent des calculs en général peu complexes, si on les compare aux calculs scientifiques non embarqués, mais qui sont d'une extrême criticité ; cela va du filtrage dans des asservissements ou des chaînes d'acquisition de données capteurs, à l'élaboration d'ordres issus des lois de pilotage, pour ne citer que quelques applications.

Des dysfonctionnements dus à l'utilisation des nombres flottants dans des systèmes critiques ont déjà été répertoriés. Par exemple, en 1991, pendant la première guerre du Golfe, une accumulation d'erreurs isolément négligeables a biaisé la trajectoire d'un missile Patriot, provoquant un accident grave. Plus précisément, une variable flottante censée contenir la valeur d'une horloge locale était incrémentée de 1/10-ième tous les dixièmes de seconde [MD92]. Or, 1/10-ième n'est pas, et ce n'est pas intuitif, représenté de façon exacte en développement binaire fini (par exemple avec une implémentation de la norme IEEE 754 sur un processeur classique).

L'emploi des nombres flottants dans les systèmes embarqués correspond à une nouvelle évolution dans le développement de ces derniers, qu'il est nécessaire d'accompagner par des méthodes de validation de codes critiques afin de détecter les erreurs même rares introduites par ces nouveaux traitements.

1.2 Mon parcours

1.2.1 En thèse

Mes premiers travaux concernant la sûreté et l'analyse statique de programmes remontent à mon stage de DEA, effectué en 1996 au Laboratoire de l'Informatique du Parallélisme à l'Ecole Normale Supérieure de Lyon sous la direction de Marc Gengler. Ceux-ci portaient sur l'évaluation partielle et l'analyse statique de langages fonctionnels typés d'ordre supérieur [Mar96].

L'évaluation partielle est une technique d'optimisation de programmes reposant sur des règles sémantiques qui permettent de garantir l'équivalence dans un certain contexte entre le programme initial et le programme optimisé [CD93, JGS93]. Plus précisément, un programme est évalué en fonction d'une partie seulement de ses données pour obtenir un nouveau programme spécialisé par rapport à ces dernières. L'évaluateur partiel garantit que le programme spécialisé auquel on fournit la partie des données absente lors de la transformation produit le même résultat que le programme initial exécuté avec le même ensemble de données.

Généralement, les programmes fournis à un évaluateur partiel sont annotés de façon à indiquer quelles parties du code peuvent être évaluées statiquement et lesquelles doivent être résidualisées, c'est-à-dire reconstruites dans le programme spécialisé. Ces annotations sont produites automatiquement par une analyse statique du code source, appelée analyse des temps de liaison [CC94, HS91, Lau91].

Un des intérêts de cette approche est que, étant donné un programme pour lequel certaines garanties de sûreté ont été apportées, l'évaluation partielle permet d'obtenir automatiquement des versions spécialisées de celui-ci, optimisées pour certains contextes d'exécution et vérifiant les mêmes propriétés de sûreté que le programme initial.

Familiarisé avec ce domaine lors de mon stage de DEA, je décidais de poursuivre dans cette voie pour mon doctorat [Mar00], sous la direction de Marc Gengler et Luc Bougé. En 1996-1997, peu de travaux avaient été menés sur l'évaluation partielle de programmes parallèles et je choisissais d'étudier plus particulièrement ce thème, dans le cadre favorable qu'offrait le Laboratoire de l'Informatique du Parallélisme à l'Ecole Normale Supérieure de Lyon.

Au début de ma thèse, à partir de septembre 1996, ma première approche a consisté à retrouver, pour des systèmes parallèles, des résultats théoriques classiques de l'évaluation partielle de programmes séquentiels. Les travaux de Torben Mogensen sur le λ-calcul pur [Mog92, Mog94] offraient pour cela un cadre de travail concis que je décidais de reprendre pour un langage formel parallèle : le π-calcul [Mil93, Mil99]. Mes travaux, publiés en 1997 dans les actes du symposium "Partial Evaluation and Semantics-Based Partial Evaluation" [GM97], montrent comment un système de processus peut être optimisé par évaluation partielle d'une partie de ses communications : celles pour lesquelles les canaux de communication et les contenus des messages sont connus a priori. Cet article définit aussi des conditions nécessaires pour qu'une communication puisse être réduite tout en conservant une certaine équivalence entre le programme initial et le programme résiduel : la bisimulation barbelée faible (weak barbed bisimulation).

Une autre aspect de ce travail dans le π-calcul est lié à la possibilité de générer automatiquement, par auto-application, un générateur de compilateurs à partir d'un évaluateur partiel (ce que l'on appelle les projections de Futamura [Fut71]). Ce générateur prend en entrée un interpréteur décrivant la sémantique d'un langage et produit un compilateur pour ce langage, vers le langage cible de l'évaluateur partiel. La simple correction de ce dernier est suffisante pour garantir celle du générateur et des compilateurs obtenus. Assurer la correction d'un compilateur est en général une tâche difficile, rarement accomplie, et générer un code final correct à partir d'un programme écrit dans un langage de haut niveau et dont on a vérifié la sûreté est une étape importante souvent mise entre parenthèses. Dans mon article à PEPM'97, un méta-évaluateur ainsi qu'un évaluateur partiel auto-applicable pour le π-calcul sont définis, ouvrant la voie à une généralisation de cette approche pour des systèmes de processus communicants.

A la suite de cette étude théorique, il m'apparaissait important de montrer l'utilité pratique de mes travaux. Pour cela, je décidais de m'intéresser à un langage moins formel et plus proche des véritables langages de programmation que le π-calcul. Mon choix se portait sur Concurrent-ML [NN99, Rep99] dont le modèle de concurrence est proche de celui du π-calcul. J'ai eu à cette période l'opportunité d'aller travailler quelques temps à l'Oregon Graduate Institute (Etats Unis), avec Tim Sheard, et je profitai de cette occasion pour me familiariser avec l'analyse des temps de liaison et l'évaluation partielle de ML [MTHM97, TS97]. De juin à septembre 1997, j'implémentais, sous la direction de Tim Sheard, la première version du vérificateur de types du langage Meta-ML et participais à la rédaction du manuel d'utilisateur [SBM00].

A partir de septembre 1997, je rejoignais le Laboratoire d'Informatique de Marseille, à l'Université Aix-Marseille II, pour poursuivre ma thèse sous la direction de Marc Gengler qui venait d'obtenir un poste de Professeur dans cette université. Je travaillais tout d'abord sur une généralisation à Concurrent-ML de l'approche retenue par Tim Sheard pour Méta-ML (la programmation étagée), publiée dans les actes des Rencontres Francophones du Parallélisme (RENPAR) en 1998 [MG98].

Ayant défini les principes de l'évaluation partielle de systèmes de processus communicants, un problème crucial restait à résoudre : la détection automatique des parties statiques d'un programme. Comme cela a déjà été mentionné, cette détection s'effectue par une analyse statique du code source appelée analyse des temps de liaison. Déterminer précisément les communications statiques d'un programme écrit en Concurrent-ML nécessite une connaissance fine de son flot de contrôle, ce qui englobe, une connaissance de la topologie des communications : il est en effet nécessaire de connaître le plus précisément possible, pour chaque envoi de message, la liste possible des destinataires. Les analyses existantes pour Concurrent-ML [SNN97] ou le π-calcul [BDNN98] étaient insuffisantes dans la mesure où elles associaient une propriété unique à chaque canal de communication, indépendamment de son contexte d'utilisation. Ceci ne permet pas en général d'annoter efficacement un programme en vue de son évaluation partielle car, dans ce cadre, il est fréquent que pour un même canal, certains messages soient connus alors que d'autres ne le sont pas. Dans ce cas, les analyses mentionnées précédemment n'offrent pas d'autre alternative que de considérer que le canal en question transmet toujours des messages dynamiques (c'est-à-dire des messages dont le contenu n'est pas connu à la compilation).

Entre 1998 et 2000, je consacrais une partie importante de ma thèse à la définition d'une analyse de flot de contrôle pour Concurrent-ML reposant sur un calcul précis de la topologie des communications d'un programme (ces travaux se trouvaient interrompus de septembre 1998 à juillet 1999, période durant laquelle j'effectuais mon service militaire). Cette analyse détecte en outre, pour chaque envoi de message, les destinataires potentiels en tenant compte des précédences entre les différentes communications. Tout d'abord, elle ordonne (partiellement) les communications réalisées par chaque processus en construisant pour chacun d'eux un automate d'états finis en fonction de leur code. Ensuite, elle calcule une approximation conservatrice des interactions entre processus en se fondant sur un automate réduit dont la taille est polynomiale en celle du programme à analyser. Ces travaux ont fait l'objet d'une communication dans les actes de "SPIN Model Checking and Software Verification", en l'an 2000 [MG00b].

Outre l'évaluation partielle, les analyses de flot de contrôle pour des systèmes de processus communicants sont utiles pour prouver des propriétés de sûreté et de sécurité. J'ai illustré ce type d'utilisation dans un article publié dans les actes des Rencontres Francophones du Parallélisme (RENPAR) en l'an 2000 [MG00a], notamment en montrant comment vérifier le bon fonctionnement d'un mécanisme d'allocation de circuits virtuels similaire à celui utilisé dans ATM. D'autres exemples sont présentés dans ma thèse [Mar00]. La précision apportée par le calcul de la topologie des communications de l'application s'avère nécessaire pour obtenir ces résultats. Depuis ces travaux, des analyses encore plus précises, mais aussi d'une complexité supérieure, ont été proposées notamment par J. Feret [Fer05a].

Je concluais mes travaux sur l'évaluation partielle de systèmes de processus mobiles en proposant une analyse des temps de liaison pour Concurrent-ML, couplée à un évaluateur partiel et à l'analyse de flot de contrôle décrite précédemment. Ces travaux ont été publiés dans les actes de la conférence "EUROPAR", en 2001 [MG01]. Dans cet article ainsi que dans ma thèse, j'illustre comment l'ensemble de ces techniques permet, par exemple, d'optimiser un protocole de communication pour un certain contexte d'utilisation. Prouver la correction d'un protocole de communication est souvent une tâche complexe et développer des versions de celui-ci optimisées par rapport à certaines conditions d'utilisation demande une révision complète des preuves, ce qui est en général prohibitif. De telles versions optimisées peuvent être obtenues par évaluation partielle, comme cela a été fait par G. Muller, E.-N. Volanschi et R. Marlet pour les parties séquentielles du protocole RPC (Remote Procedure Call [MVM97]). Mes travaux permettent des optimisations

supplémentaires, en supprimant certains envois de messages. Par exemple, un protocole transmettant des messages par paquets peut être optimisé lorsque la taille des messages (et par conséquent le nombre de paquets) est connue a priori, même si le contenu des messages demeure inconnu au moment de l'évaluation partielle.

En novembre 2000, je soutenais ma thèse sur le thème de l'évaluation partielle et de l'analyse statique de processus mobiles, obtenant la mention très honorable avec félicitations du jury.

1.2.2 Les débuts de Fluctuat

En décembre 2000, je rejoignais le centre de Saclay du Commissariat à l'Energie Atomique pour travailler dans le Laboratoire de Sûreté des Logiciels (LSL). Ce laboratoire, entièrement dédié aux méthodes de vérification et de validation de programmes (test [MA00], preuve [BPR+02], interprétation abstraite [GGP+01]), m'offrait l'opportunité attrayante de poursuivre des travaux en analyse statique et me permettait, particularité attractive à mes yeux, de développer des recherche à la fois théoriques et appliquées, en collaboration avec des équipes universitaires de la région parisienne spécialisées dans l'analyse statique ainsi qu'avec de grands groupes industriels développant des logiciels embarqués hautement critiques (dans les domaines de l'avionique et du nucléaire notamment).

A mon arrivée au CEA, j'étais associé à un projet du cinquième programme cadre de recherche et de développement de la Communauté Européenne[4] dans le cadre duquel E. Goubault venait de proposer un sujet nouveau en analyse statique et qui allait devenir mon principal thème de recherche : la validation de traitements numériques en nombres flottants. Dans les secteurs de l'aéronautique, de l'automobile, du nucléaire, etc., de nombreux logiciels embarqués critiques commencent à effectuer des calculs en nombres flottants dont la précision doit être garantie pour des questions de sûreté de fonctionnement. Comme cela a déjà été mentionné dans la section 1.1.2, il s'agit là d'une évolution de ce type d'applications qui n'effectuaient auparavant que des traitements numériques beaucoup plus modestes, implémentés en virgule fixe, voire seulement des traitements booléens. Ces évolutions des applications embarquées font de la validation de la précision des calculs effectués par des logiciels critiques un thème de recherche nouveau, riche sur le plan théorique, et répondant à un besoin industriel important.

Sur un plan scientifique, E. Goubault venait de proposer une approche novatrice, consistant à calculer, par analyse statique par interprétation abstraite [CC77, CC92], la propagation des erreurs d'arrondi commises au cours d'un calcul, ce qui permet de déterminer les opérations responsables des principales pertes de précision [Gou01]. Ces informations sont précieuses pour comprendre les raisons d'une imprécision sur le résultat d'un calcul et pour aider le programmeur à la corriger. L'aide à la correction est un problème de première importance : le simple fait de savoir qu'une valeur numérique est imprécise est une information très insuffisante car il est souvent difficile pour le programmeur d'en comprendre les raisons. Par exemple, il est utile de connaître l'écart entre un calcul en nombres flottants et un calcul idéal en nombres réels et, en cas de divergence, de savoir dans quelles fonctions ou à quelles lignes de code se trouvent les opérations responsables des principales imprécisions.

Mes premiers travaux sur le thème de la précision numérique, publiés dans les actes de l'"European Symposium on Programming" (ESOP) en 2002 [Mar02a], ont porté sur la définition d'une

[4] Le projet IST-1999-20527 "Daedalus", coordonné par P. Cousot et associant Airbus France, l'Ecole Normale Supérieure, AbsInt, le CEA, l'Ecole Polytechnique, l'Université de Copenhage, Polyspace, l'Université de Tel-Aviv, l'Université de Sarbruck et l'Université de Trier.

famille de sémantiques améliorant celle sur laquelle E. Goubault se fondait pour définir son analyse statique.

Pour donner au lecteur une intuition du principe de fonctionnement de ces sémantiques, considérons l'exemple suivant : soient $a_{\mathbb{F}} = 621.3$ et $b_{\mathbb{F}} = 1.287$ deux valeurs que l'on supposera appartenir à un ensemble simplifié de nombres flottants composés de quatre chiffres écrits en base dix. Supposons que des erreurs initiales sont attachées à $a_{\mathbb{F}}$ et $b_{\mathbb{F}}$. On écrit $a = 621.3\vec{\varepsilon} + 0.05\vec{\varepsilon}_1$ et $b = 1.287\vec{\varepsilon} + 0.0005\vec{\varepsilon}_2$ pour indiquer que l'erreur sur $a_{\mathbb{F}}$ (resp. $b_{\mathbb{F}}$) vaut 0.05 (resp. 0.0005). $\vec{\varepsilon}$ est une variable formelle attachée aux termes flottants de a et b et $\vec{\varepsilon}_1$ et $\vec{\varepsilon}_2$ sont des variables formelles indicées par les points de contrôle auxquels a et b sont définis dans le code. Intéressons-nous au produit $a_{\mathbb{F}} \times b_{\mathbb{F}}$ dont la valeur exacte est $a_{\mathbb{F}} \times b_{\mathbb{F}} = 799.6131$. Dans notre système le résultat de ce calcul est :

$$a \times b = 799.6131\vec{\varepsilon}\vec{\varepsilon} + 0.06435\vec{\varepsilon}\vec{\varepsilon}_1 + 0.31065\vec{\varepsilon}\vec{\varepsilon}_2 + 0.000025\vec{\varepsilon}_1\vec{\varepsilon}_2$$

que l'on réécrit :

$$a \times b = 799.6131\vec{\varepsilon} + 0.06435\vec{\varepsilon}_1 + 0.31065\vec{\varepsilon}_2 + 0.000025\vec{\varepsilon}_{12}$$

L'étape de réécriture permet de ne garder qu'une seule variable formelle par coefficient et obéit aux règles suivantes : les indices de $\vec{\varepsilon}$, $\vec{\varepsilon}_1$ et $\vec{\varepsilon}_2$ sont considérés comme des mots sur l'alphabet des points de contrôle et le produit de deux variables formelles produit une nouvelle variable formelle indicée par la concaténation des mots des deux opérandes. La variable $\vec{\varepsilon}$ a pour indice le mot vide, un mot de longueur n correspond à une erreur d'ordre n. Les mots composés des mêmes lettres sont supposés identiques. La différence entre $a_{\mathbb{F}} \times b_{\mathbb{F}}$ et 621.35×1.2875 vaut 0.375025 et cette erreur provient du fait que l'erreur initiale sur a (resp. b) a été multipliée par b (resp. a) et qu'une erreur d'ordre deux a été introduite par le produit des deux termes d'erreur initiaux. Ainsi, à la fin du calcul, la contribution à l'erreur globale de l'erreur initiale sur a (resp. b) vaut 0.06435 (resp. 0.31065). Elle correspond au coefficient de la variable formelle $\vec{\varepsilon}_1$ (resp. $\vec{\varepsilon}_2$).

Finalement, le nombre 799.6131 n'est pas représentable dans notre ensemble simplifié de nombres flottants. Suivant les normes IEEE 754 [ANS85] et IEC 559 [ISO89] et la norme IEEE 854 [IEE87], en mode d'arrondi au plus près, cette valeur sera approchée par 799.6. La multiplication elle-même a donc introduit une nouvelle erreur, dont la valeur est 0.0131 et que l'on attache à une nouvelle variable formelle $\vec{\varepsilon}_\times$. Nous obtenons le résultat final :

$$a \times b = 799.6\vec{\varepsilon} + 0.06435\vec{\varepsilon}_1 + 0.31065\vec{\varepsilon}_2 + 0.000025\vec{\varepsilon}_{12} + 0.0131\vec{\varepsilon}_\times \tag{1.2}$$

A première vue, on aurait pu penser que l'imprécision sur ce calcul était principalement due à l'erreur initiale sur a qui est cent fois plus grande que celle sur b. Cependant, l'équation 1.2 montre que l'erreur sur le résultat provient principalement de l'erreur initiale sur b. C'est cette dernière qu'il est nécessaire de réduire pour améliorer sensiblement la précision du résultat final.

Mon article publié à l'"European Symposium on Programming" [Mar02a], introduit la sémantique précédente, qui repose sur l'utilisation du langage des points de contrôle et sur les règles de réécriture associées. Il améliore les travaux antérieurs en intégrant les erreurs d'ordre supérieur, négligées auparavant, et, d'autre part, montre comment certains termes d'erreurs peuvent être regroupés, selon leur ordre ou pour calculer globalement la propagation des erreurs introduites par des blocs d'opérations cohérents (fonctions, lignes de programme, etc.). Ne négliger aucun terme d'erreur (en l'occurrence les erreurs d'ordre supérieur) est un point important dans un contexte de

sûreté de fonctionnement, et que la plupart des autres méthodes, même dynamiques (la différentiation automatique [Gri00] ou l'arithmétique stochastique [Che95] par exemple) ne garantissent pas en général. Pouvoir regrouper certains termes d'erreur permet de réduire le temps de calcul et l'espace mémoire requis pour l'analyse, élément nécessaire à la validation de codes de grande taille.

Toujours dans le cadre du projet Européen "Daedalus", avec E. Goubault et S. Putot, nous initions à partir de 2001 le développement d'un outil d'analyse statique par interprétation abstraite, baptisé Fluctuat, qui utilise les sémantiques précédemment mentionnées pour valider la précision des traitements numériques effectués par des programmes écrits en langage C. Une description de la première version de ce logiciel a fait l'objet d'une communication publiée aussi dans les actes de l'"European Symposium on Programming" (ESOP) en 2002 [GMP02]. Je contribuais notamment à la définition de l'interface graphique, répondant ainsi à un problème important à mes yeux d'un point de vue applicatif : présenter des résultats formels a priori relativement difficiles à interpréter de façon suffisamment intuitive pour être utilisables par un développeur non spécialiste des sémantiques sous-jacentes. Cette étape m'apparaissait indispensable pour une utilisation industrielle de ces analyses. Le principe retenu pour représenter graphiquement des séries de termes d'erreur consiste à dessiner un histogramme dont l'axe des d'abscisses indique les lignes de code du programme analysé et dont celui des ordonnées donne la contribution des erreurs introduites par les différentes lignes de code, à l'erreur globale sur le résultat d'un calcul. Cela permet d'identifier aisément les principales sources d'erreurs survenues au cours d'un calcul. La figure 1.2 illustre ce principe pour l'exemple de l'équation 1.2.

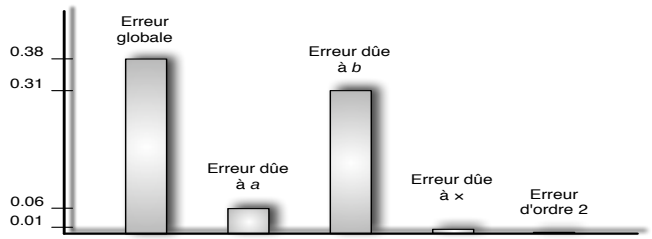

FIG. 1.2 – *Représentation sous forme d'histogramme des résultats présentés à l'équation 1.2.*

Initialement développé dans le cadre de "Daedalus", l'analyseur Fluctuat retenait l'attention d'Airbus, partenaire de ce projet, qui décidait de l'évaluer sur des logiciels embarqués d'avionique tels que le système de commande de vol de l'A340 et celui de l'A380, alors en cours de développement. Airbus estimait ces premières utilisations assez concluantes pour continuer, depuis début 2003 et la fin du projet "Daedalus", à soutenir le développement de cet outil via des contrats de recherches directs, financés par la Direction des Programmes de l'Aviation Civile (DPAC). Cette collaboration est toujours en cours et s'est poursuivie de façon ininterrompue depuis lors, sous la forme de plusieurs projets au montage desquels je participais activement.

La sémantique décrite ci-dessus, le logiciel Fluctuat ainsi que certaines des extensions décrites ci-après m'ont valu plusieurs invitations à des séminaires de laboratoires, courant 2002, à l'Ecole Normale Supérieure (séminaire d'interprétation abstraite), à l'INRIA Rocquencourt (Séminaire du groupe A3) et à l'Ecole Normale Supérieure de Lyon (séminaire du LIP). Une version longue de l'article d'ESOP sur les sémantiques pour la précision numérique, comprenant aussi quelques extensions, a été acceptée pour publication en 2004 dans le "Journal of Higher Order and Symbolic

Computation" [Mar06].

1.2.3 Fluctuat : évolutions

A partir de 2002, mes travaux de recherche ont été en grande partie destinés à améliorer les analyses décrites précédemment et qui constituent le principe fondamental de fonctionnement de Fluctuat.

Je me concentrais tout d'abord sur la définition d'une analyse relationnelle adaptée aux calculs numériques effectués dans des boucles. Dans un programme, le corps d'une boucle peut être vu comme une fonction mathématique f et itérer cette boucle n fois revient à calculer la n-ième itérée $f^{(n)}$ de f. Les fonctions itérées constituent un élément central de la théorie du chaos qui propose des outils mathématiques pour l'étude de ces systèmes [ASY96, Bar93, Bea91]. Notamment, les exposants de Lyapunov [ASY96, Moo92], permettent de déterminer la stabilité d'une fonction itérée (au sens où les itérées de f ne divergent pas pour deux valeurs initiales voisines). Mes travaux sur ce sujet montrent comment déterminer, par un calcul d'exposants de Lyapunov abstraits, la stabilité des calculs en nombre flottants mais aussi des termes d'erreur associés, pour un calcul effectué dans une boucle. Ils ont fait l'objet d'une communication dans les actes du "Static Analysis Symposium", en 2002 [Mar02b].

L'étude de la propagation des erreurs dans les calculs effectués dans une boucle pose de nombreux problèmes qui amenaient les premières versions de Fluctuat à sur-approximer un très grand nombre de termes d'erreur. Même pour des exemples extrêmement simples, l'approche naïve ne fonctionne pas. Par exemple, au cours de l'analyse du calcul itéré (stable) de

$$x = x \times a, \quad \text{avec} \quad 0 < a < 1 \tag{1.3}$$

l'erreur associée à x peut diverger, si l'on n'y prend pas garde. Ces problèmes ont été précisément décrits dans une communication présentée en 2003 par S. Putot au séminaire de Dagstuhl sur le thème "Numerical Software with Result Verification" et a fait l'objet d'une communication dans les actes de ces rencontres, par S. Putot, E. Goubault et moi-même [PGM04].

Pour obtenir des résultats précis pour une boucle dont le corps correspond, par exemple, à l'équation 1.3, l'analyseur doit déplier un certain nombre k de fois son corps, ce qui revient à ajouter des points de contrôle dans le programme (on multiplie par k le nombre de points de contrôle présents dans la boucle). k n'étant pas connu a priori, sa valeur peut être fixée par un paramètre de l'analyseur (toutes les boucles seront dépliées k fois), ce qui peut être insuffisant dans certains cas et introduire des calculs inutiles dans d'autres. Pour apporter de meilleures solutions à ce problème, j'étudiais des techniques de dépliage et regroupement des points de contrôle en cours d'analyse, ce que l'on appelle le partitionnement dynamique des points de contrôle [Bou92, HT98]. Les techniques de partitionnement dynamique nécessitent de pouvoir identifier suffisamment précisément les différents points d'une trace d'exécution, et j'utilisais, dans un premier temps, la notion de "timestamp", introduite dans le cadre des analyses d'alias [Ven02]. Cela permet de déplier un nombre maximal de fois une boucle, tout en réduisant ce dépliage s'il s'avère inutile pour obtenir des résultats précis. Cette approche a été publiée en 2003 dans les actes de "Source Code Analysis et Manipulation" [Mar03] et s'appuie sur des travaux initiés durant l'été 2002, période à laquelle j'avais l'opportunité d'aller séjourner au Massachussetts Institute of Technology pour collaborer avec M. Rinard, spécialisé notamment dans les analyses d'alias et qui me permettait de mieux me familiariser avec ce domaine. Une amélioration de cette méthode, en utilisant des mots de contrôle

[Ami05] au lieu des timestamps fait actuellement l'objet d'une réflexion commune, avec A. Cohen et P. Amiranoff.

Toujours dans le but d'améliorer la convergence des analyses dans Fluctuat, nous décidions aussi, avec E. Goubault et S. Putot, d'expérimenter des techniques fondées sur les itérations sur les politiques (par opposition aux itérations sur les valeurs) et issues de la théorie du contrôle optimal [How60, Put94]. De mars à août 2003, je co-encadrais, avec E. Goubault, S. Putot et S. Gaubert, le stage d'option de l'Ecole Polytechnique de A. Costan sur ce thème. Ces travaux, très prometteurs, ont fait l'objet d'une communication commune aux personnes mentionnées ci-dessus et publiée dans les actes de la conférence "Computer Aided Verification" en 2005 [CGG+05].

Intéressé par Fluctuat, Airbus exprimait aussi, courant 2002, un nouveau besoin : sa nécessité, pour le développement de l'A380, de valider la précision numérique de calculs présents dans des programmes directement écrits en assembleur. Ces programmes apparaissent dans des calculateurs auxiliaires qui effectuent essentiellement de l'acquisition et du traitement numérique de signaux avant de transmettre leurs résultats au programme principal de commande de vol. Il s'agit de données critiques dans la mesure où elles servent d'entrée au système principal. L'assembleur utilisé est celui du TMS 320, un processeur dédié au traitement digital de signaux (DSP) de Texas Instruments [Tex97, Tex98].

Dans le cadre d'un projet de la Direction des Programmes de l'Aviation Civile, en collaboration avec Airbus, je commençais début 2003 à implémenter un nouvel analyseur statique, fonctionnant par interprétation abstraite, pour l'assembleur du TMS 320. Du point de vue utilisateur, il s'agissait de développer une nouvelle version de Fluctuat calculant les mêmes propriétés pour l'assembleur que pour le langage C. D'un point de vue technique, si le changement de langage n'introduit pas de nouvelle difficulté majeure concernant la précision numérique, l'analyse statique de programmes assembleur constituait un sujet nouveau. Les quelques travaux menés dans cette direction concernaient les calculs de pires temps d'exécution [FHL+01, LFW02], la compilation d'invariants à partir de langages de plus haut niveau [Riv04], ou la détection d'erreurs à l'exécution dans des codes annotés [XRB00, XRB01]. Tous ciblent du code binaire, non relogeable, ce qui diffère du code source assembleur, relogeable, notamment pour le calcul du graphe de flot de contrôle [The01].

La demande d'Airbus m'amenait à définir une nouvelle technique pour l'interprétation abstraite de codes source écrits en assembleur, capable notamment de traiter correctement les branchements dynamiques (branchements à des adresses calculées) et comportant un domaine abstrait original pour la gestion de la pile. Cette analyse a été implémentée dans un second outil, une version de Fluctuat pour le TMS 320 que j'ai entièrement programmée et qui permet aujourd'hui de traiter des applications entières d'Airbus. Cette analyse et le logiciel correspondant sont décrits dans une publication parue en 2004 dans les actes de "Program Analysis for Software Tools and Engineering" [Mar04].

Il est intéressant de souligner que ces travaux sur l'analyse de programmes assembleur s'inscrivent naturellement dans la problématique générale décrite à la section 1.1.1 : la validation de systèmes intégrés. En effet, il s'agit là d'un cas typique où un système logiciel ne peut pas être validé sans tenir compte de ses couplages avec le matériel : ici le processeur.

En 2003 et 2004, je participais aussi à une Action Spécifique du CNRS sur le thème de la validation numérique pour le calcul embarqué (GDR ARP, CNRS-STIC) regroupant plusieurs équipes françaises spécialisées en arithmétique des ordinateurs (notamment les équipes de J.M. Muller au Laboratoire de l'Informatique du Parallélisme de l'ENS Lyon, de J.M. Chesneaux au Laboratoire

d'Informatique de Paris 6 et de P. Langlois à l'Université de Perpignan). Ces équipes ont pour point commun d'avoir proposé des outils pour la validation de la précision numérique de calculs effectués en nombres flottants (CADNA [Che95], MPFI [GPR03] et CENA [Lan01]). Les discussions de ce groupe m'ont inspiré de nouveaux travaux portant sur la comparaison formelle et expérimentale de ces outils. Il peut en effet paraître difficile, au premier abord, de comparer des résultats obtenus, par exemple, par une méthode d'intervalles à ceux obtenus par différentiation automatique (car les propriétés calculées ne sont pas les mêmes). Mon étude a fait l'objet d'une publication dans les actes de la conférence "Verification, Model Checking and Abstract Interpretation", en 2005 [Mar05a]. Dans la continuité de cet article, A. Chapoutot, lors de son stage de seconde année de master effectué sous ma direction de mars à septembre 2005, a proposé une nouvelle analyse couplant les sémantiques de Fluctuat et la différentiation automatique [Cha05]. Une fois implémentée dans Fluctuat, cette analyse doit permettre d'améliorer sensiblement la précision de l'analyseur en réduisant fortement les phénomènes d'effet enveloppant [DM97] introduits par les intervalles. J'ai aussi eu l'occasion de présenter ces travaux dans une des réunions du projet V3F auquel je suis associé. V3F est une action concertée incitative (ACI) du CNRS sur le thème de la "validation et vérification de logiciels avec calculs à virgule flottante". Cette action regroupe notamment les équipes de B. Legeard à l'Université de Franche-Comté, de M. Rueher du laboratoire d'Informatique Signaux et Systèmes de Sophia Antipolis (I3S), des équipes de l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), notamment A. Gotlieb et B. Jeannet et du laboratoire de Sûreté des Logiciels du CEA (notamment B. Marre). Débuté en 2003 pour une durée de trois ans, le but principal de ce projet est de développer des techniques de résolution de contraintes spécifiques aux nombres flottants.

Suite à l'Action Spécifique autour de la validation numérique évoquée au cours du paragraphe précédent, il m'apparaissait utile de collaborer plus étroitement avec des équipes de recherche spécialisées en arithmétique des ordinateurs, dans le but de mieux exploiter les propriétés fines de cette dernière au niveau de l'analyse statique. Courant 2005, j'entrais en contact avec M. Daumas, du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, et P. Langlois, de l'Université de Perpignan Via Domitia, afin d'initier des échanges et des travaux de recherche communs. En particulier, M. Daumas, avec G. Melquiond, avait initié des travaux sur la preuve formelle de propriétés concernant l'arithmétique flottante [DM04, DMM05, dDLM05] et développé le logiciel GAPPA. P. Langlois a pour sa part développé des techniques de validation et de correction de la précision de calculs flottant par différentiation automatique [Lan01]. Nos premiers échanges ont permis de mieux formaliser un problème d'intégration discrète de valeurs fournies par un signal continu : dans certains cas, les approches exactes telles que celle utilisée par Fluctuat et qui consistent à majorer finement mais sûrement les pertes de précision numérique survenant au cours d'un calcul, ne permettent pas d'apporter des éléments de réponse concluant à des problèmes très répandus en pratique. Par exemple, lorsqu'un intégrateur discret est utilisé pour effectuer la somme des valeurs que l'on obtient en échantillonnant un signal continu, il est possible que, dans le pire des cas, la somme discrète ne soit pas bornée même si l'intégrale du signal continu l'est (on peut par exemple supposer que $f(t) = \sin(t)$ dans l'équation suivante) :

$$\left| \int_0^T f(t)dt \right| < M_1 \not\Longrightarrow \left| \sum_{i=1}^{i=k} \text{round}(f(\frac{i \times T}{k})) \right| < M_2 \tag{1.4}$$

Cela peut amener des outils de validation à conclure que le résultat du calcul précédent ne sera pas borné, si les erreurs d'arrondi ne se compensent jamais, ce qui paraît très improbable en pratique

(à moins qu'un problème de conception ne force cette situation à se produire[5]). Cependant, il n'existe pas à ce jour de réponse entièrement satisfaisante à ce problème, comme cela a été décrit dans un article rédigé par M. Daumas, P. Langlois et moi-même et soumis en février 2006 au journal Technique et Science Informatique [BDLM06].

1.2.4 La chaire X-Thales sur les systèmes industriels complexes

Titulaire d'un poste de chargé d'enseignement à temps partiel à l'Ecole Polytechnique (voir la section 1.4), je me voyais proposer par le département d'informatique, en juin 2004, de travailler avec D. Krob à la mise en place de la chaire X-Thales nouvellement créée pour étudier les systèmes industriels complexes (voir section 1.1.1). Cette chaire comporte des activités d'enseignement (décrites à la section 1.4) mais aussi de recherche décrites ci-dessous.

A l'automne 2004, je travaillais, avec D. Krob à la définition des grands axes de recherche de la chaire. De notre point de vue, un système peut être vu comme une fonction de transfert dépendant d'un ensemble de paramètres physiques et informatiques. Les paramètres physiques décrivent typiquement l'environnement dans lequel évolue le système. Ils correspondent à des valeurs continues dont l'évolution est souvent modélisable à l'aide d'équations aux dérivées partielles. Les paramètres informatiques sont des variables discrètes qui traduisent typiquement les choix effectués par l'opérateur pour piloter le système. Les valeurs calculées par la fonction de transfert traduisent la réponse de celui-ci aux sollicitations de l'environnement et de l'opérateur. Elles peuvent elles-même rétroagir sur les entrées.

Ensuite, un système de systèmes peut être modélisé par un ensemble de fonctions de transfert couplées entre elles. Lors de l'analyse des propriétés globales d'un système, l'étude des couplages entre sous-systèmes devient cruciale et difficile, notamment lorsque ceux-ci reposent sur des modèles mathématiques de natures différentes, comme décrit à la section 1.1.1.

Cette vision, consistant à voir un système dans sa globalité comme un ensemble de fonctions interconnectées entre elles, nous amène à identifier quatre grandes thématiques de recherche :

1. *Systémique :* la systémique correspond à la première phase de la conception d'un système : compréhension du système vu globalement, décomposition, spécification, modélisation.
2. *Etude des sous-systèmes :* on retrouve ici des thématiques disciplinaires importantes pour la conception des principaux sous-systèmes d'un systèmes complexe, par exemple la thématique des logiciels embarqués ou celle de la théorie du contrôle.
3. *Couplages inter-systèmes :* cette problématique survient au moment de l'intégration des composants d'un système, lorsque les sous-systèmes développés séparément sont connectés. On retrouve par exemple ici la thématique des systèmes hybrides et celle des interfaces homme-système.
4. *Etude des fonctions globales d'un système :* cette problématique, liée à la dernière étape de la conception d'un système, après la phase d'intégration, est centrée sur l'étude des propriétés globales du système, par exemple la tolérance aux pannes.

[5] On peut assimiler le dysfonctionnement du missile Patriot à une telle erreur de conception (voir la section 1.1.2), la même constante mal représentée étant additionnée un grand nombre de fois. Pour éviter ce type de problèmes, on peut, a minima, recommander aux concepteurs de logiciels embarqués de ne pas accumuler des constantes, d'utiliser le mode d'arrondi au plus près afin de compenser au mieux les arrondis et d'effectuer des remises à zéro régulières des accumulateurs.

En nous fondant sur cette analyse, nous décidions, en novembre 2004 de créer un séminaire mensuel, à l'Ecole Polytechnique, pour initier des réflexions sur ces différents axes. Géré par D. Krob, E. Goubault et moi-même, ce séminaire accueillait entre autres, durant l'année scolaire 2004-2005 G. Berry (systèmes synchrones), T. Henzinger (automates hybrides), S. Gaubert (systèmes dynamiques monotones) et P. Rouchon (contrôle de systèmes complexes). Un groupe de travail regroupant notamment D. Krob, E. Goubault, M. Pouzet et moi-même se réunit régulièrement, depuis le printemps 2005 pour travailler sur la modélisation mathématique de systèmes complexes.

Ces réflexions sur les systèmes complexes me conduisaient à réfléchir sur de nouveaux développements en lien avec la validation, la précision numérique et les systèmes hybrides. Comme cela a été mentionné à la section 1.1.2, les problèmes de précision numérique se trouvent au cœur des problèmes d'intégration système.

Les analyses pour la précision numérique utilisées dans Fluctuat (voir section 1.2.2) permettent de détecter des divergences entre calculs en nombres réels et en nombres flottants sous l'hypothèse que les entrées du programme sont comprises entre deux bornes fixes. On dira par exemple que la valeur fournie par un certain capteur est toujours comprise entre -1 et 1. Or, ces entrées dépendent de l'environnement physique du système que l'on étudie et évoluent aussi suivant des lois physiques. Par exemple, les valeurs retournées au cours du temps par un capteur pourront suivre une fonction solution d'un système d'équations aux dérivées partielles. Le fait de ne pas tenir compte de ces phénomènes pendant l'analyse d'un programme amène à sur-estimer les possibilités d'erreurs, parfois de manière importante.

Je souhaite développer de nouvelles techniques d'analyse statique de systèmes hybrides discrets-continus, permettant de valider le comportement d'un programme (système discret) évoluant dans un environnement physique réaliste (système continu), plus précisément par interprétation abstraite de systèmes dont le comportement dynamique est défini à l'aide de systèmes hybrides. Mes travaux préliminaire sur ce sujet ont fait l'objet d'une première communication à "Numerical and Symbolic Abstract Domains" [Mar05b]. Ils ont aussi fait l'objet du stage de seconde année de master de O. Bouissou [Bou05], effectué sous ma direction de mars à septembre 2005. Ce sujet continue d'être étudié en thèse par O. Bouissou, sous la direction d'E. Goubault et de moi-même. Nos premiers résultats ont été soumis en novembre 2005 au journal "Higher Order and Symbolic Computation" dans un article co-signé par O. Bouissou et moi-même [BM06b].

Je souhaite aussi développer une autre direction de recherche inspirée par les problématiques d'intégration système, de précision numérique et de validation : l'analyse des sous-systèmes physiques d'un système complexe. Il existe actuellement plusieurs techniques et outils utilisables dans un contexte industriel pour valider automatiquement et dans tous les contextes d'utilisation des systèmes purement logiciels. Par exemple, des outils tels que Astrée [CCF+05], ou Fluctuat permettent de garantir l'absence de certains types d'erreurs dans des codes embarqués écrits en langage C. Cependant, aucune méthode de validation n'a été développée pour extraire automatiquement des propriétés de bon fonctionnement, valables dans tous les contextes d'utilisation (autrement dit des invariants), pour les composants physiques d'un système complexe. Ceux-ci sont en général modélisés et spécifiés dans des formalismes qui permettent de simuler (de manière imparfaite) leur comportement. Ces simulations peuvent ainsi être vues comme de simples exécutions de cas de tests, aucune aide n'étant fournie par ailleurs pour choisir des cas pertinents par rapport au modèle.

Au moment de l'intégration, la validation d'un système industriel complexe nécessite une connaissance fine du comportement de ses sous-systèmes logiciels et physiques. Mon but est de

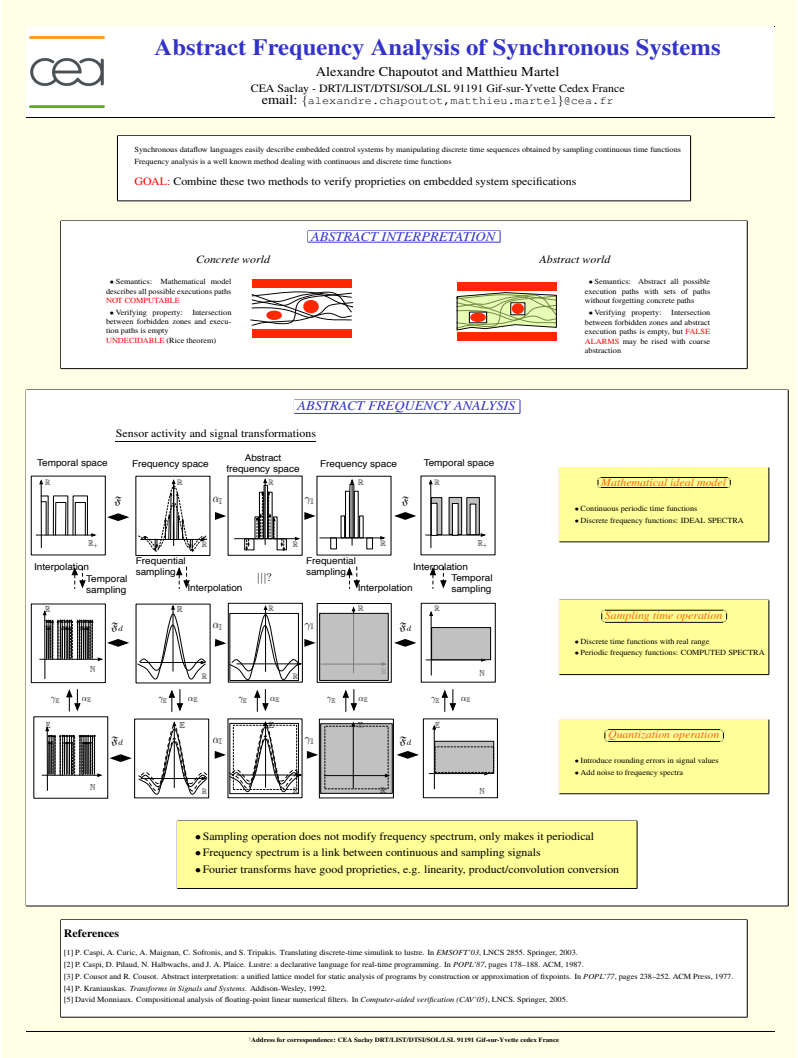

FIG. 1.3 – *Poster référencé par [CM06].*

développer des techniques permettant d'extraire automatiquement des propriétés valides dans tous les contextes (invariants) à partir de modèles décrivant les composants physiques d'un système. Ce sujet fait l'objet, depuis l'automne 2005 de la thèse d'A. Chapoutot dirigée par E. Goubault et moi-même. Les premiers résultats obtenus dans cette direction ont fait l'objet d'un poster présenté à la conférence Languages, Compilers, and Tools for Embedded Systems (LCTES) en 2006 [CM06] et repris à la figure 1.3.

1.3 Travaux en cours et perspectives à court et moyen terme

Pour une plus ample description des principales problématiques actuelles et à venir liées à l'analyse statique pour la précision numérique, le lecteur pourra se référer à un article co-signé par E. Goubault, S. Putot et moi-même, publié dans les actes de l'"European Congress on Embedded Real Time Software" en 2006 [GMP06].

Concernant l'analyseur Fluctuat et ses évolutions, signalons que ce projet fait actuellement l'objet d'autres collaborations industrielles à travers des contrats de recherche avec des acteurs industriels : Airbus, mais aussi l'Institut de Radioprotection et de Sûreté Nucléaire (IRSN), Hispano-Suiza (intégrée au groupe industriel SAFRAN) et Siemens VDO en ce qui concerne les projets directs ; les pôles de compétitivité Systématic (région Ile-de-France), et Aéronautique Systèmes Embarqués (région Midi-Pyrénées) ainsi qu'une partie de leurs partenaires industriels. D'autres liens forts ont été tissés par exemple avec Renault et EADS Space Transportation qui devraient à court ou moyen terme déboucher sur de nouveaux projets. Je participe activement, en collaboration avec E. Goubault et S. Putot, au montage et à la réalisation de ces projets. En nous permettant de confronter notre approche à des applications industrielles de grande taille, ces contrats motivent souvent de nouveaux développements théoriques pour pallier les difficultés rencontrées lors de leur étude.

L'analyse statique de la précision des traitements numériques reste un thème de recherche récent, dans lequel de nombreux travaux restent à faire. Les travaux sur les systèmes hybrides ainsi que ceux sur l'extraction d'invariants de modèles mathématiques, mentionnés à la section 1.2.4, illustrent cette situation. Ils font actuellement l'objet de thèses que je co-encadre.

Le fait de considérer que les entrées d'un programme peuvent être modélisées par des fonctions continues, typiquement les solutions d'un système d'équations différentielles, m'a récemment conduit à travailler sur la résolution sûre de tels systèmes : pour abstraire correctement un système continu, il est en effet indispensable de connaître des sous-approximations et des sur-approximations garanties en tout point de celui-ci. Des premiers travaux prometteurs, menés en collaboration avec O. Bouissou et concernant une variante garantie de la méthode de Runge-Kutta sont actuellement en cours de finalisation. En outre, la méthode proposée permet de produire deux fonctions $\underline{F}(t)$ et $\overline{F}(t)$ telles que si $F(t)$ est la solution d'une équation différentielle, alors pour tout $t \in \mathbb{R}^+$, $\underline{F}(t) \leq F(t) \leq \overline{F}(t)$ et $|\overline{F}(t) - \underline{F}(t)| \leq \epsilon$ pour un ϵ choisi a priori par l'utilisateur. Un article sur ce sujet est en cours de rédaction. Ces travaux ont par ailleurs été présentés par O. Bouissou aux journées ARINEWS des 28 et 29 novembre 2005 et dans un séminaire du Laboratoire Signaux et Systèmes de l'Ecole Supérieure d'Electricité en mars 2006. Plus généralement, le développement de méthodes à précision garantie pour des problèmes d'analyse numérique classiques constitue un axe de recherche important et prometteur.

Une autre direction concerne la définition de domaines relationnels, plus précis que les domaines non-relationnels utilisés traditionnellement. De tels domaines ont été proposés pour la détection d'erreurs à l'exécution [Fer05b, Fer05c, Min04, Min05]. Une autre approche, très prometteuse pour les domaines utilisés par Fluctuat a aussi été proposée par E. Goubault et S. Putot [GP05]. De nombreux travaux restent à être effectués dans cette direction. Leur intérêt semble évident lorsqu'il apparaît que pour certains problèmes (voir l'équation (1.4)) les solutions exactes ne peuvent apporter de solution entièrement satisfaisante (dans le pire cas, une erreur importante peut effectivement survenir bien que ce cas paraisse très improbable).

Dans mon article à VMCAI'05 [Mar05a], je propose plusieurs pistes consistant à coupler des sémantiques existantes afin de calculer des propriétés de plus grand intérêt (séries d'erreurs et différentiation automatique, séries d'erreurs et arithmétique stochastique). Ces travaux repris par A. Chapoutot lors de son stage de seconde année de Master [Cha05] mériteraient d'être encore poursuivis, notamment en utilisant les travaux de D. Monniaux sur l'interprétation abstraite de programmes probabilistes [Mon01a, Mon01b] (comme cela été déjà proposé par E. Goubault en 2001 [Gou01]).

Les travaux sur les stratégies d'itération (partitionnement dynamique, itération sur les poli-

tiques) constituent aussi des directions de recherche prometteuses dans lesquelles de nombreux travaux restent à effectuer.

Concernant les systèmes industriels complexes et plus particulièrement la modélisation de systèmes hybrides, des résultats préliminaires prometteurs ont été obtenus avec D. Krob et S. Bliuze. La modélisation rigoureuse de ce type de systèmes est un axe de recherche riche, mathématiquement difficile et largement inexploré. Les méthodes de validation correspondantes, qui n'ont pas encore été étudiées, présenteront un très grand intérêt scientifique et applicatif.

Comme l'illustrent les paragraphes précédents, de nombreux travaux s'inscrivant dans la continuité directe de ceux que j'ai effectués ces dernières années font actuellement l'objet de recherches actives. D'autres, nombreux aussi, restent encore à être effectués. Des perspectives à plus long terme seront évoquées à la fin de ce document mais nous pouvons déjà constater que les thèmes de recherche décrits dans les pages précédentes méritent de nombreux approfondissements.

1.4 Mes activités d'enseignement

Depuis le début de mon doctorat, en septembre 1996, j'ai enseigné tous les ans quasiment sans interruption. J'étais tout d'abord moniteur de l'enseignement supérieur à Lyon, pendant ma première année de thèse où j'enseignais à l'Université Lyon 1 et à l'Ecole Normale Supérieure. Après mon départ à Marseille je poursuivais mon monitorat à l'Ecole Supérieure d'Ingénieurs de Luminy (ESIL). Qualifié aux fonctions de maître de conférence en 2001 (qualification n^{o}02227119002), j'effectuais des vacations à l'Ecole Polytechnique de 2001 à 2003. En 2003, j'obtenais un poste de chargé d'enseignement à temps partiel à l'Ecole Polytechnique.

Depuis 2003, j'interviens aussi chaque année dans un module d'interprétation abstraite, dirigé par R. Cousot, du Master Parisien de Recherche en Informatique (ex DEA "Programmation : Sémantique, preuves et langages" cohabilité par l'Université Paris 7, l'Ecole Normale Supérieure de Cachan, l'Ecole Normale Supérieure et l'Ecole Polytechnique). Ces cours ont pour particularité de me permettre de présenter mes travaux de recherche sur l'analyse statique pour la précision numérique.

Enfin, j'ai obtenu, à la rentrée scolaire 2005, un poste de maître de conférence à l'Institut National des Sciences et Techniques du Nucléaire (INSTN), école d'ingénieurs rattachée au CEA. Mes fonctions consistent plus particulièrement à assurer la coordination du master d'ingénierie des systèmes industriels complexes, décrit ci-après, et le CEA, l'INSTN cohabilitant cette formation dans laquelle enseignent de nombreux chercheurs du CEA.

Comme cela a été mentionné à la section 1.2.4, j'ai été associé par l'Ecole Polytechnique, en juin 2004, à la mise en place des activités de recherche et d'enseignement de la Chaire X-Thales dirigée par D. Krob. En ce qui concerne l'enseignement, nous avons proposé un nouveau master d'ingénierie des systèmes industriels complexes qui a été habilité en juillet 2005 et qui a ouvert ses portes en septembre 2005. Une description de ce cursus a fait l'objet d'un article, par D. Krob et moi-même, publié dans les actes de la conférence annuelle de l'Association Française d'Ingénierie Système [KM06].

Ayant été associé à ce dossier en juin 2004, j'ai participé presque depuis le début à la définition des programmes, à la sélection des cours et des enseignants, aux discussions visant à intégrer d'autres établissements (l'Institut National des Sciences et Technologies du Nucléaire, une école d'ingénieurs dépendant du CEA et l'Université Paris 11), au montage du dossier d'habilitation, à la promotion auprès des élèves (organisation de conférences pour les élèves, site web, journées

d'orientation et d'information organisées à l'Ecole Polytechnique, etc.), à l'organisation pédagogique de la première rentrée (emplois du temps, coordination avec d'autres départements de l'Ecole Polytechnique associés au Master tels que le département de Mathématiques Appliquées ou de Langues, coordination avec les autres établissements académiques associés, organisation des interventions de certains industriels et notamment l' "Université Thales", organisation de journées de pré-rentrée pour les élèves, etc.).

De plus, depuis l'ouverture de ce master, je m'investis tout autant dans la coordination globale de cette formation (interactions avec les enseignants, suivi des élèves, suivi de projets de synthèse, jurys, etc.) que dans l'enseignement proprement dit : j'ai en effet proposé de dispenser deux cours de la filière "Systèmes de transport" sur le thème des systèmes hybrides et sur celui de la sûreté des logiciels.

Mes projets futurs en matière d'enseignement s'inscrivent naturellement dans la continuité des activités précédemment mentionnées, en lien étroit avec mes activités de recherche. Ils visent à contribuer très activement à faire vivre ce nouveau Master au sein du département d'informatique de l'Ecole Polytechnique, à contribuer à son développement au fil des années, à veiller à ce qu'il demeure, dans l'esprit du projet initial, une formation scientifique de haut niveau (partageant par exemple de nombreux cours avec des masters de recherche tels que le Master Parisien de Recherche en Informatique) et une formation ouvrant aux élèves des perspectives de carrières enthousiasmantes au sein de grands groupes industriels via des enseignements préparant au travail de chef de projets techniques complexes et via une promotion active du Master auprès des grandes entreprises des transports, de la micro-électronique, etc.

Pour finir, voici un récapitulatif des enseignements que j'ai effectués :

- *année scolaire 2005-2006 :*
  - Master Parisien de Recherche en Informatique, partie du cours "Interprétation Abstraite" sur la précision numérique, 3h.
  - Cours "Système Hybrides", seconde année du Master d'ingénierie des systèmes industriels complexes de l'Ecole Polytechnique, l'INSTN et l'Université Paris 11, 12h.
  - TD d'Informatique parallèle et Distribuée, INF554, Majeure d'Informatique 2, Troisième année de l'Ecole Polytechnique, 10h.
- *année scolaire 2004-2005 :*
  - Master Parisien de Recherche en Informatique, partie du cours "Interprétation Abstraite" sur la précision numérique, 6h.
  - TD d'Informatique parallèle et Distribuée, INF554, Majeure d'Informatique 2, Troisième année de l'Ecole Polytechnique, 20h.
- *année scolaire 2003-2004 :*
  - Partie de cours du DEA de Programmation, filière "Sémantique et Interprétation Abstraite", cours d' "Analyse Statique de Propriétés Numériques, de Sécurité et de Mobilité, 6h.
  - TD d'Informatique parallèle et Distribuée, INF554, Majeure d'Informatique 2, Troisième année de l'Ecole Polytechnique, 20h.
  - TD de Fondements de l'Informatique, INF431, Seconde Année de l'Ecole Polytechnique, 40h.
- *année scolaire 2002-2003 :*
  - Partie de cours du DEA de Programmation, filière "Sémantique et Interprétation Abstraite", cours de "Sémantique Géométrique, Numérique et Probabiliste". Présentation des

sémantiques numériques, 6h.
  - TD d'Informatique parallèle et Distribuée, INF554, Majeure d'Informatique 2, Troisième année de l'Ecole Polytechnique, 20h.
  - TD de Fondements de l'Informatique, INF431, Seconde Année de l'Ecole Polytechnique, 40h.
- *année scolaire 2001-2002 :*
  - TD d'Informatique parallèle et Distribuée, INF554, Majeure d'Informatique 2, Troisième année de l'Ecole Polytechnique, 20h.
- *année scolaire 1999-2000 :*
  - première année d'Ecole d'Ingénieurs, Ecole Supérieure d'Ingénieurs de Luminy, Logiciel de Base (assembleur, génération de code, interruptions), 32h de TD.
  - première année d'Ecole d'Ingénieurs, Ecole Supérieure d'Ingénieurs de Luminy, Encadrement des projets de Logiciel de Base (Réalisation d'un assembleur-éditeur de liens), 24h de TP.
  - Seconde année d'Ecole d'Ingénieurs, Ecole Supérieure d'Ingénieurs de Luminy, Architecture avancée (pipelines, caches, etc.), 16h de TD. Ce cours est la suite du cours de Logiciel de Base et introduit les notions avancées concernant l'architecture des processeurs modernes.
- *année scolaire 1997-1998 :*
  - première année d'Ecole d'Ingénieurs, Ecole Supérieure d'Ingénieurs de Luminy, Logiciel de Base (assembleur, génération de code, interruptions), 32h de TD. Utilisation d'un simulateur pour processeur MIPS, Gestion des appels de procédures, appels récursifs au niveau de l'assembleur.
  - première année d'Ecole d'Ingénieurs, Ecole Supérieure d'Ingénieurs de Luminy, Encadrement des projets de Logiciel de Base (Réalisation d'un assembleur-éditeur de liens), 24h de TP.
  - première année d'Ecole d'Ingénieurs, Ecole Supérieure d'Ingénieurs de Luminy, Théorie des Graphes (notions de base et algorithmes), 16h de TD.
- *année scolaire 1996-1997 :*
  - Deug MASS, Université Lyon I, Initiation à l'algorithmique et à la programmation en Turbo Pascal, 16h de TD, 24h de TP.
  - première année de Magistère de biologie moléculaire et cellulaire, ENS-Lyon, Bio-informatique, 32h de TD (Algorithmes pour le traitement de séquences d'ADN (8 séances), Traitement et Synthèse d'Images (4 séances), Optimisation (4 séances).

1.5 Curriculum vitae

Matthieu Martel

ADRESSE : CEA - Centre de Saclay
LIST-DTSI-SOL
CEA F91191 Gif-Sur-Yvette
TÉLÉPHONE : 01 69 08 71 83
MAIL : matthieu.martel@cea.fr
NÉ LE : 23 Septembre 1973

SITUATION ACTUELLE

Statut	:	• Ingénieur-Chercheur au Commissariat à l'Energie Atomique, depuis décembre 2000, • Chargé d'enseignements à temps incomplet à l'Ecole Polytechnique, depuis septembre 2003.
Laboratoire	:	Laboratoire de Sûreté des Logiciels, CEA, Direction de la Recherche Technologique.
Thèmes de recherche	:	Analyse statique, Précision numérique, Systèmes industriels complexes.

THÈSE

Titre	:	*Analyse Statique et Evaluation Partielle de Systèmes de Processus Mobiles.*
Soutenance	:	Le 29 Novembre 2000, à l'Université d'Aix-Marseille II, Ecole Doctorale de Mathématiques et Informatique. Mention : Très honorable avec félicitations du jury.
Jury	:	- Françoise Bellegarde, Prof., Université de Franche-Comté, Rapporteur, - Giovanni Coray, Prof., Ecole Polytechnique Fédérale de Lausanne, Rapporteur, - Marc Gengler, Prof., Université d'Aix-Marseille II, Directeur de thèse, - Michael Goldsmith, Prof., Oxford University, Examinateur, - Traian Muntean, Prof., Université d'Aix-Marseille II, Président du jury.
Financement	:	Allocation de recherche (MESR) et Monitorat.

SCOLARITÉ

1999-2000	:	$3^{ième}$ année de thèse, *Laboratoire d'Informatique de Marseille (LIM), Université de la Méditerranée*, sous la direction de Marc Gengler.
1998-1999	:	Scientifique du Contingent.
1997-1998	:	2^{de} année de thèse, *Laboratoire d'Informatique de Marseille (LIM), Université de la Méditerranée*, sous la direction de Marc Gengler.
1996-1997	:	$1^{ière}$ année de thèse, *Laboratoire de l'Informatique du Parallélisme (LIP), Ecole Normale Supérieure de Lyon*, sous la direction de Marc Gengler et Luc Bougé.
1995-1996	:	DEA d'Informatique, $3^{ième}$ année de Magistère, *Ecole Normale Supérieure de Lyon.*
1994-1995	:	Maîtrise d'Informatique, 2^{de} année de Magistère, *Ecole Normale Supérieure de Lyon.*
1993-1994	:	Licence d'Informatique, $1^{ière}$ année de Magistère, *Ecole Normale Supérieure de Lyon.*

PUBLICATIONS

Revues internationales

1. M. Martel, *Semantics of Roundoff Error Propagation in Finite Precision Computations*, Journal of Higher-Order and Symbolic Computation, 19:7-30, 2006.
2. O. Bouissou et M. Martel, *Static Analysis by Abstract Interpretation of Hybrid Systems*, soumis au Journal of Higher-Order and Symbolic Computation, 2005.

Conférences internationales

3. O. Bouissou et M. Martel, *A Runge-Kutta method for computing guaranteed solutions of ODEs*, 12th GAMM - IMACS International Symposion on Scientific Computing, Computer Arithmetic and Validated Numerics, 2006.
4. E. Goubault, M. Martel et S. Putot, *Some future challenges in the validation of control systems*, European Congress on Embedded Real Time Software (ERTS), 2006.
5. A. Costan, E. Goubault, S. Gaubert, M. Martel et S. Putot, *A policy iteration algorithm for computing fixed points in static analysis of programs*, Int. Conf. on Computer Aided Verification (CAV), LNCS 3576, pages 462-475, 2005.
6. M. Martel, *An Overview of Semantics for the Validation of Numerical Programs*, Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI), LNCS 3385, pages 59-77, 2005.
7. M. Martel, *Validation of Assembler Programs for DSPs: A Static Analyzer*, ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE), ACM-Press, pages 8-13, 2004.
8. S. Putot, E. Goubault et M. Martel, *Static Analysis Based Validation of Floating-Point Computations*, follow-up of the seminary on Numerical Software with Result Verification of Dagstuhl, Germany, LNCS 2991, pages 306-313, 2003.
9. M. Martel, *Improving the Static Analysis of Loops by Dynamic Partitioning Techniques*, Third IEEE International Workshop on Source Code Analysis and Manipulation (SCAM), IEEE Press, pages 13-21, 2003.
10. M. Martel, *Static Analysis of the Numerical Stability of Loops*, Static Analysis Symposium (SAS), LNCS 2477, pages 133-150, 2002.
11. M. Martel, *Roundoff Error Propagation in Finite Precision Computations: a Semantic Approach*, European Symposium On Programming (ESOP), LNCS 2305, pages 194-208, 2002.
12. E. Goubault, M. Martel et S. Putot, *Asserting the Precision of Floating-Point Computations: a Simple Abstract Interpreter*, European Symposium On Programming (ESOP), LNCS 2305, pages 209-212, 2002.
13. M. Martel et M. Gengler, *Partial Evaluation of Concurrent Programs* (EUROPAR), LNCS 2150, pages 504-513, 2001.
14. M. Martel et M. Gengler, *Communication Topology Analysis for Concurrent Programs*, 7th International SPIN Workshop on Model Checking of Software (SPIN), LNCS 1885, pages 265-286, 2000.
15. M. Martel et M. Gengler, *Self-applicable Partial Evaluation for the Pi-calculus*, ACM SIGPLAN Symposium on Partial Evaluation and Program Manipulation (PEPM), pages 36-46, 1997.

Workshops

16. M. Martel, *Towards an Abstraction of the Physical Environment of Embedded Systems*, Int. Workshop. on Numerical and Symbolic Abstract Domains (NSAD), 2005.

Conférences nationales

17. D. Krob et M. Martel, *Le master professionnel "Ingénierie des systèmes industriels complexes" : une formation d'architecte système délivrée par l'Ecole Polytechnique, l'Institut National des Sciences et Techniques Nucléaires et l'Université Paris Sud 11*, Soumis à la 4-ième Conférence AFIS (Association Française d'Ingénierie Système), 2006.
18. M. Martel et M. Gengler, *Analyse Statique pour la Sûreté des Applications Distribués*, Rencontres Francophones du Parallélisme (Renpar), pages 217-222, 2000.
19. M. Martel et M. Gengler, *Des Etages en Concurrent ML*, Rencontres Francophones du Parallélisme (Renpar), pages 247-250, 1998.

Rapports

20. E. Goubault, M. Martel et S. Putot, *Fluctuat user's guide*, Rapport technique CEA, ref. DTSI/SLA/02-497, 2004.
21. T. Sheard, Z. Benaissa et M. Martel, *Introduction to Multi-Stage Programming Using MetaML*, Rapport Technique, Pacific Software Research Center Pacific Software Research Center, Oregon Graduate Institute, Septembre 2000.
22. M. Martel, *Analyse Statique et Evaluation Partielle de Systèmes de Processus Mobiles*, Thèse de Doctorat, Laboratoire d'Informatique de Marseille, Novembre 2000.

Posters

23. A Chapoutot et M. Martel et *Abstract Frequency Analysis of Synchronous Systems*, ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems.

Projets - Collaborations Industrielles

- Participation au projet *Preuves Approchées*, depuis 2002. Ce projet est financé par la Direction des Programmes de l'Aviation Civile. Collaboration industrielle avec Airbus sur le thème de la sûreté des calculs numériques embarqués.
- Participation à l'Action Spécifique *Validation Numérique pour le Calcul Embarqué*, GDR ARP du CNRS-STIC, 2003-2004.
- Participation à l'Action Concertée Incitative (ACI-Sécurité) du CNRS : *V3F - Validation et Vérification en présence de calculs à Virgule Flottante* 2003-2005.
- Participation au projet européen IST-1999-20527 *Daedalus* en 2001 et 2002. Collaboration industrielle avec Airbus sur le thème de la sûreté des calculs numériques embarqués.

Développement de Logiciels

- Développement complet du logiciel FC3X, outil de validation par interprétation abstraite de codes numériques écrits en assembleurs pour des DSPs (Digital Signal Processors). Développé en OCAML. Ce logiciel est actuellement testé par Airbus qui envisage de l'utiliser industriellement. Voir la référence [7].
- Participation au développement du logiciel Fluctuat, outil de validation par interprétation abstraite des calculs en nombres flottants présents dans des programmes C. Développé en C/C++ avec Eric Goubault et Sylvie Putot. Ce logiciel est actuellement testé par Airbus qui envisage de l'utiliser industriellement. Voir les références [8] et [12].
- Participation au développement de MetaSML, version réflexive du langage ML. MetaSML est développé par Tim Sheard à Oregon Graduate Institute. Implémentation en SML de la première version du système de typage lors d'un séjour à OGI en 1997. Voir la référence [21].

Séminaires de Laboratoires

- Séminaire INRIA-Alchemy, INRIA-Futurs, Novembre 2003.
- Séminaire du LIP, Ecole Normale Supérieure de Lyon, Mai 2002.
- Séminaire Commun A3-CRI-PRISM-LRI, INRIA Rocquencourt, Mai 2002.
- Séminaire d'Interprétation Abstraite, Ecole Normale Supérieure, Janvier 2002.
- Séminaire du Laboratoire d'Informatique Théorique, Ecole Polytechnique Fédérale de Lausanne, Octobre 2000.

Collaborations Internationales

- Collaboration avec Martin Rinard et Patrick Lam, Laboratory of Computer Science, Massachussets Institute of Technology (chercheur invité en juillet et août 2002).
- Collaboration Tim Sheard , Pacsoft Group, Oregon Graduate Institute (chercheur invité de juin à août 1997).

ENCADREMENT - ANIMATION SCIENTIFIQUE

Depuis Octobre 2005	:	Encadrement des thèses d'Olivier Bouissou et Alexandre Chapoutot dans la continuité de leurs stages de DEA (sous la responsabilité d'Eric Goubault, DR CEA).
Depuis Septembre 2004	:	Participation, notamment avec D. Krob, à l'élaboration du projet de recherche de la chaire X-Thales (article synthétique en cours de rédaction).
Depuis Septembre 2003	:	Organisation, avec N. Williams du séminaire de laboratoire "Sûreté des logiciels" au CEA. Séminaire généralement hebdomadaire.
Depuis Octobre 2004	:	Organisation, avec D. Krob et E. Goubault du séminaire de la chaire "Systèmes industriels complexes" à l'Ecole Polytechnique. Séminaire mensuel.
Mars - Septembre 2005	:	Encadrement de O. Bouissou, stage de seconde année de Master (MPRI). Poursuite en thèse acquise (bourse CFR octroyée par le CEA). Sujet : *analyse statique de systèmes hybrides discrets-continus.*
Mars - Septembre 2005	:	Encadrement de A. Chapoutot, stage de seconde année de Master (Paris 6, parcours Logiciels Sûrs). Poursuite en thèse acquise. Sujet : *analyse statique pour la précision numérique.*
Janvier - Septembre 2003	:	Co-encadrement de A. Aretina , postdoc. Sujet : *analyse statique pour la précision numérique.*
Avril - Août 2003	:	Co-encadrement de A. Costan, stage d'option de l'Ecole Polytechnique. Sujet : *stratégies d'itération sur les politiques pour le calcul de points fixes en analyse statique.*

FONCTIONS D'ENSEIGNEMENT

Depuis Septembre 2005	:	Maître de conférence à l'Institut National des Sciences et Techniques du Nucléaire (décret non encore paru en janvier 2006) .
Depuis Septembre 2004	:	Chargé d'enseignement à temps incomplet à l'Ecole Polytechnique.
Depuis Juin 2004	:	Participation, avec D. Krob, au montage du Master "Ingénierie des Systèmes Industriels Complexes", cohabilité par l'Ecole Polytechnique, l'Institut National des Sciences et Techniques du Nucléaire et l'Université Paris 11.
Septembre 2003 - Juin 2004	:	Chargé d'Enseignements à temps incomplet à Ecole Polytechnique (durée : 10 mois).
2001	:	Qualification aux fonctions de maître de conférence n°02227119002.
1996 - 1999	:	Moniteur de l'Enseignement Supérieur.

Chapitre 2

Analyses dynamiques pour la précision numérique

2.1 Introduction

Méthodes d'intervalles, arithmétique stochastique, différentiation automatique, etc. : plusieurs approches ont été développées pour estimer et pour améliorer la précision des traitements numériques effectués dans des programmes. Historiquement imaginées pour améliorer la qualité de codes numériques intensifs, elles sont aujourd'hui utilisées telles quelles ou après avoir été adaptées dans le domaine de la validation d'applications critiques.

Parmi les nombreux travaux existant dans ce domaine, on ne trouve que peu d'études comparatives traitant des méthodes de natures différentes [Mar05a]. Cela est partiellement dû au fait que les propriétés calculées par ces techniques sont difficiles à comparer. En effet, par exemple, comment comparer les résultats d'un calcul par intervalles à ceux obtenus par différentiation automatique ?

Dans ce chapitre, après quelques rappels concernant la norme IEEE 754, nous essayons de clarifier les liens existant entre les méthodes les plus répandues d'estimation de la précision numérique d'un calcul. Pour cela nous utilisons une sémantique formelle simple, dont les valeurs sont composées de flottants et de termes exacts d'erreur globale. Cette sémantique est assez expressive pour être formellement comparée aux autres méthodes. Il s'agit aussi d'une instance particulière de la famille de sémantiques définies au chapitre suivant [Mar02a, Mar06]. Ensuite, nous définissons des sémantiques formelles pour les méthodes d'intervalles, de différentiation automatique et pour l'arithmétique stochastique. Ces méthodes sont en général présentées dans un cadre moins formel. Cette présentation nous permet de comparer formellement les propriétés calculées par chacune d'entre elles.

De notre point de vue, une méthode est adaptée à la validation d'applications critiques si elle permet à l'utilisateur de détecter des erreurs survenant lors de l'exécution de programmes embarqués, c.à.d. sans instrumentation du code. De ce point de vue, les méthodes améliorant la précision des calculs posent des problèmes car elles ne reflètent pas le comportement du programme lors de son utilisation finale puisque le code source est modifié par instrumentation.

En fin de chapitre nous discutons de la possibilité d'adapter ces méthodes à l'analyse statique. Signalons aussi que des résultats expérimentaux étayant ceux présentés dans ce document ont été publiés dans [Mar05a]. Pour conclure, précisons enfin que nous nous limitons ici aux méthodes permettant de traiter les problèmes de précision numérique. D'autres travaux relatifs

aux nombres flottants ne sont pas considérés, comme les techniques de preuve de propriétés portant sur les nombres flottants (par exemple [BD03, DRT01, Har99]), ou les techniques de résolutions de contraintes sur les flottants utiles à la génération de cas de tests [MRL01]. Nous excluons aussi les méthodes d'amélioration de la précision comme les arithmétiques multi-précision [HLFZ01, Pri91] ou exactes [PEE97]. Ces arithmétiques augmentent la précision des calculs mais ne sont pas directement utiles pour estimer l'erreur résiduelle.

2.2 La norme IEEE 754

Introduite au début des années 1980 pour harmoniser la représentation des nombres flottants et le comportement des opérations élémentaires dans les processeurs, la norme IEEE 754 [ANS85, ISO89, Gol91] est maintenant suivie par la plupart des constructeurs. Elle fournit une sémantique précise pour les opérations arithmétiques présentes dans des langages de haut niveau. Suivant cette norme, un nombre flottant x en base β est défini par :

$$x = s \cdot (d_0.d_1 \dots d_{p-1}) \cdot \beta^e = s \cdot m \cdot \beta^{e-p+1} \tag{2.1}$$

où

- $s \in \{-1, 1\}$ est le signe de x,
- $m = d_0.d_1 \dots d_{p-1}$ est la mantisse dont les chiffres d_i vérifient $0 \leq d_i < \beta$ pour $0 \leq i \leq p-1$,
- p est la précision c.à.d. le nombre de chiffres de la mantisse,
- e est l'exposant, $e_{min} \leq e \leq e_{max}$.

Un nombre flottant x est dit normalisé lorsque $d_0 \neq 0$. La normalisation évite les représentations multiples d'un même nombre, en interdisant par exemple de noter $0.1e^{-1}$ au lieu de $1.0e^{-2}$. La norme IEEE 754 autorise plusieurs valeurs pour p, e_{min} et e_{max} : le format simple précision est défini par $\beta = 2$, $p = 23$, $e_{min} = -126$ et $e_{max} = +127$ tandis que, en double précision, nous avons $\beta = 2$, $p = 52$, $e_{min} = -1022$ et $e_{max} = +1023$. $\beta = 2$ est la seule base autorisée mais des variantes autorisent par exemple la base 10, comme la norme IEEE 854. Des formats autres que les simples et les doubles, dits étendus, sont aussi définis par la norme. Ils dépendent de l'implémentation et nous ne détaillerons pas ici.

La norme IEEE 754 définit aussi les nombres dénormalisés qui sont des nombres flottants tels que $d_0 = d_1 = \ldots = d_k = 0$, $k < p-1$ et $e = e_{min}$ ce qui permet d'avoir un dégradation progressive de la précision autour de zéro [Gol91]. Enfin, la norme IEEE 754 définit des valeurs spéciales :

- NAN (Not a Number) pour le résultat des opérations non valides,
- les valeurs $\pm\infty$ pour les dépassements de capacité,
- les valeurs $+0$ et -0 (zéros signés) qui sont des nombres dénormalisés.

Dans la suite de ce document, nous noterons $\mathbb{F}$ l'ensemble des nombres flottants, sans préciser s'il s'agit des nombres en simple ou double précision, nos résultats étant indépendants de ce choix.

Pour les opérations élémentaires entre nombres flottants, quatre modes d'arrondi sont définis : il s'agit des modes vers $-\infty$, vers $+\infty$, vers zéro et au plus près. On les notera respectivement $\circ_{-\infty}$, $\circ_{+\infty}$, $\circ_0$ et $\circ_\sim$.

Soit $\mathbb{R}$ l'ensemble des nombres réels et $\uparrow_\circ : \mathbb{R} \to \mathbb{F}$ la fonction qui calcule l'arrondi d'un nombre réel suivant le mode $\circ \in \{\circ_{-\infty}, \circ_{+\infty}, \circ_0, \circ_\sim\}$. La norme IEEE 754 spécifie le comportement des opérations élémentaires $\Diamond \in \{+, -, \times, \div\}$ entre nombres flottants par :

$$f_1 \Diamond_{\mathbb{F},\circ} f_2 \ = \uparrow_\circ (f_1 \Diamond_{\mathbb{R}} f_2) \tag{2.2}$$

L'équation (2.2) indique que le résultat d'une opération $\Diamond_{\mathbb{F},\circ}$ entre deux nombres flottants sera le résultat que l'on aurait obtenu dans les réels (opérateur $\Diamond_{\mathbb{R}}$), arrondi suivant le mode $\circ$ choisi. L'arrondi de la fonction racine carrée est défini de façon similaire, mais, pour des raisons théoriques [LMT98], l'implémentation d'autres fonctions telles que les fonctions trigonométriques n'est pas spécifié.

Pour nos besoins ultérieurs, nous introduisons aussi la fonction $\downarrow_\circ$: $\mathbb{R} \rightarrow \mathbb{R}$ qui calcule l'erreur exacte survenant lorsqu'un nombre réel r est approché par $\uparrow_\circ (r)$. Par définition nous avons :

$$\downarrow_\circ (r) = r - \uparrow_\circ (r) \tag{2.3}$$

Remarquons que les opérations élémentaires sont des fonctions totales sur $\mathbb{F}$, c.à.d. que les résultats entre valeurs spéciales sont aussi définis. Par exemple, $1 \div +\infty = +0$, $+\infty \times +0 =$ NAN, etc. [Gol91, Hau96]. Dans la suite de ce document nous ne traiterons pas le cas des valeurs spéciales, supposant que les opérations que nous effectuons ont des opérandes valides et retournent des nombres normalisés.

De nombreux phénomènes peuvent mener à des pertes de précision importantes dans des calculs en nombres flottants. Par exemple, une élimination catastrophique survient lorsque l'on soustrait des valeurs légèrement erronées x et y proches l'une de l'autre [DM97, Gol91] : le résultat obtenu n'est que la soustraction des chiffres non significatifs, les chiffres significatifs s'étant compensés. Une absorption a lieu lorsque l'on additionne des nombres de magnitudes très différentes $x \ll y$; dans ce cas, $x +_{\mathbb{F}} y = y$ avec $x \neq 0$.

Nous appellerons erreurs du premier ordre les erreurs dues aux arrondis de valeurs initiales ou les erreurs d'arrondi introduites par les opérations élémentaires. Lorsque des termes d'erreur du premier ordre sont multipliés entre eux, nous obtenons des erreurs d'ordre supérieur. Par exemple, $(x + \varepsilon_x) \times (y + \varepsilon_y) = xy + x\varepsilon_y + y\varepsilon_x + \varepsilon_x\varepsilon_y$. Ici, $x\varepsilon_y + y\varepsilon_x$ est une nouvelle erreur du premier ordre, tandis que $\varepsilon_x\varepsilon_y$ est une erreur du second ordre.

Les erreurs survenant dans un calcul en nombres flottants peuvent être mesurées de plusieurs façons. L'erreur directe estime pour une entrée x donnée, la distance entre la solution exacte et la solution approchée. Pour une description de la notion d'erreur inverse [Wil63], le lecteur pourra se reporter à l'ouvrage de F. Chaitin-Chatelin et V. Frayssé [CCF96].

2.3 Calcul de l'erreur globale

Nous présentons ici la sémantique $[\![.]\!]_{\mathbb{E}}$ utilisée comme référence dans tout ce chapitre. $[\![.]\!]_{\mathbb{E}}$ calcule un nombre flottant, le résultat d'une série d'opérations conformes à la norme IEEE 754, ainsi que l'erreur survenant au cours de ce traitement. Autrement dit, cette sémantique calcule l'erreur avant, telle que nous l'avons définie à la section 2.2, entre la solution exacte à un problème et la solution approchée obtenue dans les flottants. Pour calculer cette erreur exactement, $[\![.]\!]_{\mathbb{E}}$ utilise des nombres réels, et, par conséquent, elle n'est pas directement implémentable. $[\![.]\!]_{\mathbb{E}}$ correspond à la sémantique $[\![.]\!]^{\mathcal{L}^0}$ introduite dans les articles [Mar02a, Mar06].

Formellement, dans $[\![.]\!]_{\mathbb{E}}$, une valeur v est représentée par un vecteur à deux dimensions $v = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$ où $f \in \mathbb{F}$ représente le flottant utilisé par l'ordinateur et $e \in \mathbb{R}$ représente l'erreur exacte attachée à f. $\vec{\varepsilon}_f$ et $\vec{\varepsilon}_e$ sont des variables formelles utilisées pour séparer les termes flottant et erreur

de v. Par exemple, en simple précision, en utilisant les fonctions $\uparrow_\circ$ et $\downarrow_\circ$ définies dans la section 2.2, le nombre réel $\frac{1}{3}$ est représenté par la valeur

$$v = \uparrow_\circ (\frac{1}{3})\vec{\varepsilon}_f + \downarrow_\circ (\frac{1}{3})\vec{\varepsilon}_e = 0.333333\vec{\varepsilon}_f + (\frac{1}{3} - 0.333333)\vec{\varepsilon}_e$$

Plus généralement, notre sémantique interprète une donnée d comme suit :

$$[\![d]\!]_{\mathbb{E}} = \uparrow_\circ (d)\vec{\varepsilon}_f + \downarrow_\circ (d)\vec{\varepsilon}_e \tag{2.4}$$

La sémantique des opérations élémentaires est définie dans la figure 2.1, les opérandes x_1 et x_2 étant données par l'équation (2.5).

$$x_1 = f_1\vec{\varepsilon}_f + e_1\vec{\varepsilon}_e \quad \text{et} \quad x_2 = f_2\vec{\varepsilon}_f + e_2\vec{\varepsilon}_e \tag{2.5}$$

$$x_1 + x_2 = \uparrow_\circ (f_1 + f_2)\vec{\varepsilon}_f + [e_1 + e_2 + \downarrow_\circ (f_1 + f_2)]\,\vec{\varepsilon}_e \tag{2.6}$$

$$x_1 - x_2 = \uparrow_\circ (f_1 - f_2)\vec{\varepsilon}_f + [e_1 - e_2 + \downarrow_\circ (f_1 - f_2)]\,\vec{\varepsilon}_e \tag{2.7}$$

$$x_1 \times x_2 = \uparrow_\circ (f_1 \times f_2)\vec{\varepsilon}_f + [e_1 f_2 + e_2 f_1 + e_1 e_2 + \downarrow_\circ (f_1 \times f_2)]\,\vec{\varepsilon}_e \tag{2.8}$$

$$\frac{1}{x_1} = \uparrow_\circ \left(\frac{1}{f}\right)\vec{\varepsilon}_f + \left[\downarrow_\circ \left(\frac{1}{f}\right) + \sum_{n\geq 1}(-1)^n \frac{e^n}{f^{n+1}}\right]\vec{\varepsilon}_e \tag{2.9}$$

FIG. 2.1 – *La sémantique* $[\![.]\!]_{\mathbb{E}}$.

Les équations (2.6) à (2.8) sont immédiates. Pour l'équation (2.9), nous utilisons le fait que $\frac{1}{1+x} = \sum_{n\geq 0}(-1)^n x^n$ pour tout x tel que $-1 \leq x \leq 1$. Nous avons ainsi :

$$\frac{1}{f+e} = \frac{1}{f} \times \frac{1}{1+\frac{e}{f}} = \frac{1}{f} \times \sum_{n\geq 0}(-1)^n \frac{e^n}{f^n}$$

Ce développement en série est valable pour $-1 \leq \frac{e}{f} \leq 1$ ou, de façon équivalente, tant que $|e| \leq |f|$, c.à.d. tant que l'erreur est inférieure, en valeur absolue, au flottant. La sémantique de la division, plus compliquée, sera donnée ultérieurement, à l'équation (3.8). La sémantique de la racine carrée peut être obtenue par une technique similaire à celle utilisée pour l'inverse mais les autres fonctions élémentaires sont plus difficiles à traiter, la norme IEEE 754 ne spécifiant pas leurs arrondis.

Soient $[\![.]\!]_{\mathbb{F}}$ et $[\![.]\!]_{\mathbb{R}}$ les fonctions sémantiques calculant l'évaluation d'une expression arithmétique dans les flottants et dans les réels, respectivement. $[\![.]\!]_{\mathbb{E}}$ calcule le nombre flottant produit par un programme et la différence exacte entre ce flottant et le résultat réel, comme le souligne la proposition suivante.

Proposition 1 *soit a une expression arithmétique. Si $[\![a]\!]_{\mathbb{E}} = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$ alors $[\![a]\!]_{\mathbb{F}} = f$ et $[\![a]\!]_{\mathbb{R}} = f + e$.*

En reliant $[\![.]\!]_{\mathbb{E}}$ à la sémantique $[\![.]\!]_{\mathbb{R}}$ des nombres réels et à la sémantique $[\![.]\!]_{\mathbb{F}}$ des nombres flottants, la proposition 1 nous fournit une preuve de correction de $[\![.]\!]_{\mathbb{E}}$.

$[\![.]\!]_{\mathbb{E}}$ est adaptée à la validation de systèmes critiques car elle conserve les nombres flottants utilisés par la version non instrumentée d'un code, celle qui est effectivement embarquée dans un système, et calcule à part le terme d'erreur e introduit par les nombres flottants. Cela évite, par exemple, les divergences de flot de contrôle entre $[\![.]\!]_{\mathbb{F}}$ et la sémantique utilisée pour la validation des programmes. Cependant, cette sémantique reste un outil théorique puisqu'elle utilise des nombres réels et la fonction $\downarrow_\circ$ qui ne peut pas être calculée par un ordinateur en général. Dans le reste de ce chapitre, cette sémantique nous servira de référence pour étudier et comparer d'autres sémantiques, approchées mais implémentables. Nous comparerons ces méthodes et déterminerons leur capacité à quantifier les termes f et e qui définissent les valeurs $f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$ de $[\![.]\!]_{\mathbb{E}}$. Les liens existants entre $[\![.]\!]_{\mathbb{E}}$ et les autres méthodes présentées ci-après sont résumés par les propositions 2, 3, 4 ainsi que par la figure 3.2 du chapitre 3.

2.4 Les méthodes d'intervalles

Méthode classique, la sémantique des intervalles $[\![.]\!]_{\mathbb{I}}$ permet d'encadrer de façon sûre le résultat réel d'un calcul [DM97, JKDW01, Moo63]. Naturellement, la sémantique d'une constante d est :

$$[\![d]\!]_{\mathbb{I}} = [\uparrow_{-\infty}(d), \uparrow_{+\infty}(d)] \tag{2.10}$$

Ensuite, étant donnés $x_1 = [\underline{x_1}, \overline{x_1}]$ et $x_2 = [\underline{x_2}, \overline{x_2}]$ deux intervalles de flottants et $\Diamond$ une opération élémentaire, on définit $i = [\underline{i}, \overline{i}]$ l'intervalle à bornes réelles résultat du calcul $i = x_1 \Diamond x_2$. On définit $[\![x_1 \Diamond x_2]\!]_{\mathbb{I}}$ par :

$$[\![x_1 \Diamond x_2]\!]_{\mathbb{I}} = [\uparrow_{-\infty}(\underline{i}), \uparrow_{+\infty}(\overline{i})] \tag{2.11}$$

Une méthode d'intervalles encadre un nombre réel par deux nombres flottants, les flottants immédiatement supérieur ou égal et inférieur ou égal aux résultats réels. En d'autres termes, en utilisant les notations de la section 2.3, la sémantique des intervalles calcule une approximation supérieure et une approximation inférieure de la somme $f + e$ correspondant au terme flottant f et au terme d'erreur e de $[\![.]\!]_{\mathbb{E}}$. Ceci est résumé par la proposition suivante.

Proposition 2 *Soit a une expression arithmétique telle que $[\![a]\!]_{\mathbb{E}} = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$ et $[\![a]\!]_{\mathbb{I}} = [\underline{x}, \overline{x}]$. Alors nous avons $[\![f + e]\!]_{\mathbb{I}} \subseteq [\underline{x}, \overline{x}]$.*

Remarquons que si les bornes des intervalles sont des nombres ayant la même précision que celle utilisée dans le programme non instrumenté, comme cela est sous-entendu dans la sémantique des équations (2.10) et (2.11), alors le résultat $[\underline{x}, \overline{x}]$ produit par la méthode des intervalles encadre à la fois le résultat flottant f et le résultat réel $f + e$:

$$|[\![a]\!]_{\mathbb{R}} - [\![a]\!]_{\mathbb{F}}| \leq |\overline{x} - \underline{x}| \qquad \text{où } [\underline{x}, \overline{x}] = [\![a]\!]_{\mathbb{I}} \tag{2.12}$$

Sinon, si les bornes des intervalles ont une plus grande précision que celle utilisée pour exécuter le code non instrumenté, comme cela est le cas dans les librairies d'intervalles multi-précision [GPR03], alors la méthode des intervalles encadre le résultat réel $f + e$ mais pas le résultat flottant f. Ainsi, dans ce dernier cas, $[\![.]\!]_{\mathbb{I}}$ ne permet pas de prédire le comportement du programme non instrumenté, exécuté avec de simples nombres flottants tel qu'il se trouvera par exemple une fois embarqué dans un système.

Par ailleurs, parce que $[\![.]\!]_{\mathbb{I}}$ additionne toujours les erreurs aux flottants, cette méthode ne permet pas de distinguer deux types d'erreurs :

1. les erreurs de sensibilité, dues au fait que de petites variations des entrées peuvent induire des variations importantes des résultats, et cela même dans les réels,
2. les erreurs numériques, dues au fait qu'un calcul réalisé en nombres flottants peut diverger par rapport au même calcul réalisé avec des réels.

Par exemple, des erreurs numériques apparaissent lors de l'exécution du programme suivant qui utilise des nombres flottants simple précision :

```
float x=1.0;
float y=1.0e-8;
for (int i=0;i<1e8;i++) {
  x=x-y ;
}
```

La valeur 10^{-8} étant soustraite 10^8 fois à 1, le résultat exact dans les réels est $x_{\mathbb{R}} = 0$. Mais 10^{-8} est inférieur au chiffre le moins significatif du flottant 1.0 et une absorption a lieu : $1.0 - 10^{-8} = 1.0$ dans les flottants en simple précision. Aussi, à la fin de ce programme $x_{\mathbb{F}} = 1.0$. Si l'on utilise pour les intervalles des bornes de même précision que celle utilisée dans le programme, la sémantique $[\![.]\!]_{\mathbb{I}}$ définie par les équations (2.10) et (2.11) renvoie nécessairement un intervalle $x_{\mathbb{I}} \supseteq [0, 1]$, puisque le résultat flottant et le résultat réel appartiennent à l'encadrement. En pratique, on obtient l'intervalle $x = [-12.44141, 1.000000]$ et ce résultat ne peut pas être amélioré, car changer la précision des bornes reviendrait à modifier la propriété calculée. Dans $[\![.]\!]_{\mathbb{E}}$, $x_{\mathbb{E}} = 1.0\vec{\varepsilon}_f - 1.0\vec{\varepsilon}_e$, ce qui indique que la valeur flottante de x est 1 et que l'erreur avant exacte sur x est -1.

Comme le montre cet exemple, $[\![.]\!]_{\mathbb{I}}$ fournit moins d'informations que $[\![.]\!]_{\mathbb{E}}$ car aucune distinction n'est faite entre flottants et erreurs. Néanmoins, lorsque la largeur de l'intervalle fourni par $[\![.]\!]_{\mathbb{I}}$ est faible, il est possible de conclure que le terme d'erreur est lui aussi faible, ce qui permet de valider le calcul (formellement le terme d'erreur est borné, en valeur absolue, par la largeur de l'intervalle).

De plus, les intervalles offrent une première façon d'implémenter la sémantique $[\![.]\!]_{\mathbb{E}}$, ce qui donne une nouvelle sémantique $[\![.]\!]_{\mathbb{EI}}$. Une valeur $v = f\vec{\varepsilon}_f + [\underline{x}, \overline{x}]\vec{\varepsilon}_e$ de $[\![.]\!]_{\mathbb{EI}}$ est composée d'un flottant f et d'un intervalle d'erreur $[\underline{x}, \overline{x}]$. La sémantique des opérations élémentaires est donnée par les règles de la figure 2.1 dans lesquelles les calculs sur les termes d'erreur sont effectués en utilisant $[\![.]\!]_{\mathbb{I}}$.

Pour finir, mentionnons deux librairies de calcul par intervalles couramment employées : Boost [BMP03] et MPFI [RR02], cette dernière s'appuyant sur la librairie d'arithmétique multi-précision MPFR [HLFZ01]. Des comparaisons entre les performances de plusieurs librairies sont présentées dans [GPR03]. D'autres façons de définir les intervalles ont aussi été étudiées, notamment la représentation centre-rayon qui permet souvent de minimiser les pertes de précision. Cette technique est utilisée, par exemple, par la librairie INTLAB [Rum99].

2.5 L'arithmétique stochastique

L'idée proposée par M. La Porte et J. Vignes pour l'arithmétique stochastique [LPV74] est d'exécuter plusieurs fois un même programme, en arrondissant à chaque fois le résultat de chaque opération au flottant supérieur ou inférieur, aléatoirement. Les chiffres communs aux différents résultats finaux obtenus sont supposés exacts, à une probabilité près [Che95, DM97, Vig93]. La

sémantique stochastique $[\![.]\!]_{\mathbb{S}}$, utilise ainsi le mode d'arrondi aléatoire $\uparrow_? : \mathbb{F} \rightarrow \mathbb{F}$ défini par :

$$\uparrow_? (d) = \begin{cases} \text{soit } \uparrow_{-\infty} (d) \\ \text{soit } \uparrow_{+\infty} (d) \end{cases} \quad \text{avec une probabilité de } \frac{1}{2} \tag{2.13}$$

Les n exécutions d'un programme nécessaires à l'arithmétique stochastique sont effectuées de façon synchrone, afin de réduire les problèmes de divergence de flots de contrôle qui pourraient survenir si celles-ci étaient effectuées de façon asynchrone. Ainsi, une valeur dans la sémantique $[\![.]\!]_{\mathbb{S}}$ est un n-uplet contenant les valeurs correspondant aux n exécutions. Une constante d est interprétée de la manière suivante :

$$[\![d]\!]_{\mathbb{S}} = (\uparrow_? (d), \ldots, \uparrow_? (d)) \tag{2.14}$$

et, pour les opérations élémentaires $\Diamond \in \{+, -, \times, \div\}$, nous avons :

$$[\![x \Diamond x']\!]_{\mathbb{S}} = \left(\uparrow_? (x_1 \Diamond x'_1), \ldots, \uparrow_? (x_n \Diamond x'_n)\right) \tag{2.15}$$

Intuitivement, $[\![.]\!]_{\mathbb{S}}$ repose sur l'hypothèse que les erreurs d'arrondi survenant au cours d'un calcul sont indépendantes. Cela permet d'obtenir des résultats moins pessimistes que ceux fournis par les méthodes d'intervalles dans lesquelles les erreurs sont constamment majorées. Plus précisément, si $x = (x_1, \ldots, x_n)$ désigne les résultats des n exécutions, alors la moyenne $\overline{x}$ est une bonne approximation du résultat $x_{\mathbb{R}}$ que l'on aurait obtenu dans les réels pour ce calcul. Soit $C(x, x_{\mathbb{R}})$ le nombre de chiffres communs à $\overline{x}$ et $x_{\mathbb{R}}$. En utilisant le test de Student, $[\![.]\!]_{\mathbb{S}}$ permet d'estimer $C(x, x_{\mathbb{R}})$ avec une probabilité P [Che95].

Proposition 3 *Soit a une expression arithmétique telle que $[\![a]\!]_{\mathbb{S}} = (x_1, \ldots, x_n)$ et $[\![a]\!]_{\mathbb{E}} = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$. Alors, avec une probabilité P, $\overline{x}$ et $f + e$ ont $C(\overline{x}, f + e)$ chiffres significatifs en commun, où*

$$C(\overline{x}, f + e) = \log_{10} \left(\frac{\sqrt{n}|\overline{x}|}{\sigma \tau_P} \right) \tag{2.16}$$

et

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2 \tag{2.17}$$

Par exemple, pour $n = 3$ et $P = 0.95$, $\tau_P = 4,303$. Cependant, la proposition 3 repose sur l'hypothèse que les erreurs d'arrondi sont des variables aléatoires uniformes et indépendantes, l'indépendance signifiant que les erreurs survenant à chaque pas de calcul ne sont pas corrélées entre elles. Ceci n'est pas toujours le cas, notamment dans les boucles. Par exemple, pour le programme introduit à la section 2.4, la même erreur d'arrondi est commise à chaque itération. Les résultats expérimentaux confirment que $[\![.]\!]_{\mathbb{S}}$ ne détecte pas les problèmes de précision présents dans cet exemple [Mar05a]. De plus, la preuve de correction de la proposition 3 suppose que les erreurs d'ordre supérieur sont négligeables par rapport aux erreurs du premier ordre. Dans la section 2.6, nous présenterons un exemple de programme pour lequel cette hypothèse n'est pas vérifiée.

Parce que $[\![.]\!]_{\mathbb{S}}$ approche, à probabilité P près, le résultat d'un programme p dans les réels, cette méthode ne permet pas de garantir que des pertes importantes de précision ne surviendront pas lors de l'exécution non instrumentée de ce même programme, lorsque $[\![.]\!]_{\mathbb{F}}$ sera utilisée, par exemple dans un système embarqué. Cependant, même si cette méthode a principalement été conçue à l'origine pour détecter des problèmes de stabilité dans des codes de calcul intensif, elle peut permettre

de corroborer la qualité de la précision d'un traitement numérique lorsque (1) $C(\overline{x}, x_{\mathbb{R}})$ est grand et que (2) $\overline{x}$ est proche du résultat flottant f.

Une façon d'améliorer l'arithmétique stochastique et de la rendre plus adaptée aux problèmes de validation pourrait consister à définir une nouvelle sémantique $[\![.]\!]_{\mathbb{ES}}$ fondée sur $[\![.]\!]_{\mathbb{E}}$. Dans $[\![.]\!]_{\mathbb{ES}}$, le terme d'erreur exact de $[\![.]\!]_{\mathbb{E}}$ est calculé en utilisant $[\![.]\!]_{\mathbb{S}}$. Une valeur $v = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$ de $[\![.]\!]_{\mathbb{ES}}$ est composée d'un flottant f et d'un terme d'erreur e qui est un nombre stochastique de $[\![.]\!]_{\mathbb{S}}$. Des premiers résultats expérimentaux confirment l'intérêt de cette nouvelle approche [Cha05]. Ainsi, pour le programme de la section 2.4, en utilisant des nombres stochastiques multi-précision d'une sémantique $[\![.]\!]_{\mathbb{M}}$ pour les termes d'erreur et en approchant $\downarrow_{\circ} (f_1 - f_2)$ par $[\![f_1 - f_2]\!]_{\mathbb{M}} - [\![f_1 - f_2]\!]_{\mathbb{F}}$, ce qui est correct, modulo les hypothèses probabilistes, car les termes d'erreur de $[\![.]\!]_{\mathbb{S}}$ sont arrondis aléatoirement au niveau du dernier chiffre significatif dans la précision de $[\![.]\!]_{\mathbb{M}}$, on obtient le résultat probant

$$x_{\mathbb{ES}} = 1.0\vec{\varepsilon}_f - 0.999999939\vec{\varepsilon}_e \quad \text{avec} \quad C(x_{\mathbb{R}}, x) > 28$$

En comparaison, la sémantique $[\![.]\!]_{\mathbb{S}}$, telle qu'elle est définie par les équations (2.14) et (2.15) et implémentée dans le logiciel CADNA[1], calcule, pour le même exemple, le résultat

$$x_{\mathbb{S}} = -0.3808 \quad \text{avec} \quad C(x_{\mathbb{R}}, x) = 4$$

Signalons toutefois qu'aucune preuve de correction de $[\![.]\!]_{\mathbb{ES}}$, n'a à ce jour été formulée. Pour cela il serait nécessaire de revisiter les preuves de correction de $[\![.]\!]_{\mathbb{S}}$ et de les adapter.

2.6 La différentiation automatique

Du point de vue de la différentiation automatique, un programme calcule une fonction g de dimension m de ses entrées, $g = (g_1, \ldots, g_m)^T$, pour laquelle il est possible de calculer, simultanément à sa valeur, celles de ses dérivées [BHN02, Gri00]. Si l'on appelle $d_1, \ldots, d_n$ les données d'un programme p, alors ce dernier calcule des résultats $v_1, \ldots, v_m$ tels que :

$$\begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} = \begin{pmatrix} g_1(d_1, \ldots, d_n) \\ \vdots \\ g_m(d_1, \ldots, d_n) \end{pmatrix} \tag{2.18}$$

$v_1, \ldots, v_m$ sont les résultats à la fin de l'exécution de p. Pour tout i, $1 \leq i \leq m$, v_i est une fonction g_i des données $d_1, \ldots, d_n$. La différentiation automatique calcule, outre les termes v_i, une approximation numérique des dérivées partielles $\frac{\partial g_i}{\partial d_j}(d_1, \ldots, d_n)$ pour tout $1 \leq j \leq n$. La valeur de ces dérivées partielles permet de déterminer si une faible variation des données initiales d_j induit une modification importante des résultats v_i, ce qui est aussi appelé la sensibilité des résultats par rapport aux entrées. Si $\frac{\partial g_i}{\partial d_j}(d_1, \ldots, d_n) \approx 0$, v_i n'est pas très sensible aux variations de d_j. En revanche, si $\frac{\partial g_i}{\partial d_j}(d_1, \ldots, d_n) > 1$ alors l'erreur sur d_j sera amplifiée dans v_i d'un facteur approximativement égal à $\frac{\partial g_i}{\partial d_j}(d_1, \ldots, d_n)$.

La façon la plus intuitive de calculer les valeurs des dérivées partielles est d'effectuer une approximation linéaire d'ordre 1 au moyen de la formule usuelle :

$$g'(x) \approx \frac{g(x + \Delta x) - g(x)}{\Delta x} \tag{2.19}$$

[1] CADNA : `http ://www-anp.lip6.fr/cadna/Documentation/Accueil.php`.

Cependant, cette technique donne souvent de mauvais résultats du fait que Δx est généralement petit. Tout d'abord, une absorption peut se produire dans le calcul de $x + \Delta x$; ensuite, $g(x + \Delta x)$ et $g(x)$ sont souvent des valeurs voisines et une élimination catastrophique peut survenir lors du calcul de $g(x + \Delta x) - g(x)$; finalement, les erreurs précédentes sont amplifiées par la division par Δx. Aussi, au lieu d'utiliser la formule de l'équation (2.19), on utilise généralement la règle de dérivation de fonctions composées :

$$(f \circ g)'(x) = \big[f(g(x))\big]' = g'(x) \times f'(g(x)) \tag{2.20}$$

Chacune des fonctions g_i est vue comme une composition de fonctions élémentaires telles que des additions, multiplications, etc. et les dérivées $\frac{\partial g_i}{\partial d_j}(d_0, \ldots, d_n)$ sont calculées en utilisant l'équation (2.20).

$$v_1 = (f_1, \delta_1, \ldots, \delta_n) \quad \text{et} \quad v_2 = (f_2, \eta_1, \ldots, \eta_n) \tag{2.21}$$

$$[\![v_1 + v_2]\!]_{\mathbb{D}} = (f_1 + f_2, \delta_1 + \eta_1, \delta_2 + \eta_2, \ldots, \delta_n + \eta_n) \tag{2.22}$$

$$[\![v_1 - v_2]\!]_{\mathbb{D}} = (f_1 - f_2, \delta_1 - \eta_1, \delta_2 - \eta_2, \ldots, \delta_n - \eta_n) \tag{2.23}$$

$$[\![v_1 \times v_2]\!]_{\mathbb{D}} = (f_1 \times f_2, f_1\eta_1 + f_2\delta_1, f_1\eta_2 + f_2\delta_2, \ldots, f_1\eta_n + f_2\delta_n) \tag{2.24}$$

$$[\![\frac{v_1}{v_2}]\!]_{\mathbb{D}} = \left(\frac{f_1}{f_2}, \frac{f_2\delta_1 - f_1\eta_1}{f_2^2}, \frac{f_2\delta_2 - f_1\eta_2}{f_2^2}, \ldots, \frac{f_2\delta_n - f_1\eta_n}{f_2^2}\right) \tag{2.25}$$

FIG. 2.2 – *La sémantique $[\![.]\!]_{\mathbb{D}}$ pour la différentiation automatique.*

La sémantique $[\![.]\!]_{\mathbb{D}}$ est présentée à la figure 2.2. Elle effectue une différentiation automatique élémentaire[2] des programmes. Pour une constante d_i, nous avons :

$$[\![d_i]\!]_{\mathbb{D}} \;=\; \big(d_i, \delta_1 = 0, \ldots, \delta_i = 1, \ldots, \delta_n = 0\big) \tag{2.26}$$

Une valeur v de $[\![.]\!]_{\mathbb{D}}$ est un $(n+1)$-uplet $(f, \delta_1, \ldots, \delta_n)$. Intuitivement, à une étape donnée de l'exécution d'un programme p, v représente le résultat d'un calcul intermédiaire. Autrement dit, pour calculer une fonction g, le programme commence par calculer une fonction $h_1(d_1, \ldots, d_n)$ telle que $g(d_1, \ldots, d_n) = h_2 \circ h_1(d_1, \ldots, d_n)$ pour une certaine fonction h_2 et la valeur courante v dans $[\![.]\!]_{\mathbb{D}}$ représente

$$v = (f, \delta_1, \ldots, \delta_n) = \left(h_1(d_1, \ldots, d_n), \frac{\partial h_1}{\partial d_1}(d_1, \ldots, d_n), \ldots, \frac{\partial h_1}{\partial d_n}(d_1, \ldots, d_n)\right). \tag{2.27}$$

La différentiation automatique peut être vue comme une façon approchée de calculer les termes d'erreur $e\vec{\varepsilon}_e$ de la sémantique $[\![.]\!]_{\mathbb{E}}$. En effet, étant donné un programme implantant le calcul d'une fonction g, $[\![.]\!]_{\mathbb{E}}$ calcule

$$[\![g(d_1, \ldots, d_n)]\!]_{\mathbb{E}} = x_r = f_r\vec{\varepsilon}_f + e_r\vec{\varepsilon}_e$$

Dans le cas le plus simple $m = n = 1$, c.à.d. pour un programme p implantant une fonction g d'une seule variable d_1, le terme d'erreur e_r attaché au résultat peut être estimé à partir des

[2]Des méthodes plus sophistiquées sont mentionnés à la fin de cette section.

données numériques $g(d_1)$ et $\frac{\partial g}{\partial d_1}(d_1)$ produits par $[\![g(d_1)]\!]_{\mathbb{D}}$. Soit e_{d_1} l'erreur initiale sur d_1. Par approximation linéaire nous avons

$$x_r \approx g(d_1)\vec{\varepsilon}_f + \left(e_{d_1} \times \frac{\partial g}{\partial d_1}(d_1)\right)\vec{\varepsilon}_e \tag{2.28}$$

Dans l'équation (2.28), l'erreur exacte e_r est approchée par $e_{d_1} \times \frac{\partial g}{\partial d_1}(d_1)$. Dans le cas général, c.à.d. si $m \geq 1$ et $n \geq 1$, nous avons la propriété suivante.

Proposition 4 *Soient $e_1, \ldots, e_n$ les erreurs initiales attachées aux données $d_1, \ldots, d_n$ et soient $v_1, \ldots, v_m$ les résultats d'un calcul tels qu'ils sont définis par l'équation (2.18). Si dans la sémantique $[\![.]\!]_{\mathbb{E}}$, pour tout $1 \leq i \leq m$,*

$$v_i = [\![g_i(d_1, \ldots, d_n)]\!]_{\mathbb{E}} = x_i\vec{\varepsilon}_f + e_{r_i}\vec{\varepsilon}_e \tag{2.29}$$

alors, dans $[\![.]\!]_{\mathbb{D}}$,

$$v_i = [\![g_i(d_1, \ldots, d_n)]\!]_{\mathbb{D}} = (y_i, \delta_{i,1}, \ldots, \delta_{i,n}) \tag{2.30}$$

tel que $x_i = y_i$ et tel que le terme d'erreur e_{r_i} attaché au résultat final v_i est linéairement approximé par :

$$e_{r_i} \approx \sum_{1 \leq j \leq n} e_j \times \delta_{i,j} \tag{2.31}$$

Le principal défaut de la différentiation automatique provient de l'approximation linéaire effectuée dans l'équation (2.28), qui peut amener à sous-estimer les erreurs (ou à les sur-estimer, bien que cela soit en général moins important pour la validation d'applications critiques). Par exemple, considérons la fonction

```
float g(float x,int n) {
  y=x;
  for (int i=0;i<n;i++) {
    y=y*x;
  };
  return y;
}
```

Dans ce programme, si le paramètre $x = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$ est tel que $f < 1$ et $f + e > 1$ alors, dans les nombres flottants $g(x) \to 0$ lorsque $n \to \infty$, tandis que dans les réels, $g(x) = g(f + e) \to \infty$ quand $n \to \infty$. Dans $[\![.]\!]_{\mathbb{D}}$, la valeur retournée par $\mathtt{g}(x, n)$ est (f^{n+1}, nf^n), le terme nf^n indiquant la sensibilité de g au paramètre x. Si $f < 1$ alors $nf^n \to 0$ quand $n \to \infty$ et l'approximation de l'équation (2.31) devient très imprécise. Par exemple, si $x = 0.95\vec{\varepsilon}_f + 0.1\vec{\varepsilon}_e$ alors dans un cas nous avons

$$[\![\mathtt{g}(x, n)]\!]_{\mathbb{E}} \to 0\vec{\varepsilon}_f + \infty\vec{\varepsilon}_e \quad \text{quand } n \to \infty$$

tandis que, dans l'autre,

$$[\![\mathtt{g}(x, n)]\!]_{\mathbb{D}} \to (0, 0) \quad \text{quand } n \to \infty$$

Pour cet exemple, l'équation (2.31) peut amener à la conclusion erronée que pour $x = 0.95\vec{\varepsilon}_f + 0.1\vec{\varepsilon}_e$, $e_x \approx 0$.

Par ailleurs, la différentiation automatique tient seulement compte des erreurs sur les données, négligeant les erreurs introduites par les arrondis sur les résultats des opérations en cours de calcul. Par exemple, en nous référant à l'équation (2.6), la sémantique $[\![.]\!]_{\mathbb{E}}$ pour une addition est :

$$x_1 + x_2 = \uparrow_\circ (f_1 + f_2)\vec{\varepsilon}_f + [e_1 + e_2 + \downarrow_\circ (f_1 + f_2)]\,\vec{\varepsilon}_e$$

$[\![.]\!]_{\mathbb{D}}$ permet d'estimer les termes e_1 et e_2 mais elle néglige l'erreur due à l'addition elle-même, c.à.d. le terme $\downarrow_\circ (f_1 + f_2)$. Enfin, les termes d'erreur d'ordre supérieur sont aussi négligés. Par exemple, le terme $e_1 e_2$ qui apparaît dans le résultat du produit de l'équation (2.8) est ignoré.

Bischof et al. ont récemment publié un article décrivant les principales librairies de différentiation automatique [BHN02]. Dans certains cas, la différentiation automatique peut aussi être utilisée pour améliorer la précision d'un calcul, par ajout de termes correctifs aux nombres flottants produits par un calcul (la méthode CENA [Lan01, LN97]). Dans ce cas, les erreurs d'arrondi dues aux opérations élémentaires ne sont plus négligées et la précision du résultat final peut être accrue. Les erreurs sur l'évaluation des dérivées partielles, quant à elles ne sont pas prises en compte.

2.7 Analyse statique

En ce qui concerne les traitements numériques, la validation d'un logiciel embarqué critique nécessite au moins de prouver que la précision d'une variable est toujours acceptable. Il est possible d'effectuer des exécutions instrumentées des programmes en utilisant des sémantiques fondées sur l'arithmétique stochastique ou la différentiation automatique. Cependant, ces méthodes ne permettent pas de couvrir l'ensemble des configurations possibles et les cas pathologiques sont particulièrement difficiles à trouver, car, dans les nombres flottants, des données très voisines peuvent produire des résultats éloignés en terme de précision (à cause des phénomènes d'élimination catastrophique ou des tests instables par exemple). De plus, pour une même trace d'exécution (parcours du même chemin du graphe de contrôle), la précision peut varier significativement selon les données utilisées. L'analyse statique traite ces problèmes en permettant de valider un programme pour un grand ensemble de données, typiquement des intervalles de variation des entrées.

De nombreux analyseurs statiques implémentent des analyses d'intervalles (voir par exemple [CCF+05]). La principale limite de cette approche a déjà été soulignée à la section 2.4 : lorsque dans un résultat, un intervalle est large, il n'est pas possible de déterminer s'il s'agit d'un problème de sensibilité présent aussi dans la fonction réelle ou d'un problème de précision numérique. De plus, les intervalles utilisés pour spécifier les entrées d'un programme à valider par analyse statique sont généralement larges et cette méthode produit souvent des résultats trop pessimistes pour permettre de valider la précision des calculs. L'utilisation de domaines relationnels [Min01, Min04, Min05] permet de réduire les imprécisions introduites par l'analyse statique mais ne peut en aucun cas supprimer les problèmes intrinsèques à la méthode des intervalles : la confusion entre erreurs de sensibilité et erreurs numériques, comme cela a été mentionné à la section 2.4.

Aucune analyse statique fondée sur l'arithmétique stochastique n'a été définie à ce jour. Cependant, comme l'a suggéré E. Goubault [Gou01], il semblerait intéressant de construire une analyse statique combinant $[\![.]\!]_{\mathbb{S}}$ ou $[\![.]\!]_{\mathbb{ES}}$ et les analyses statiques de programmes probabilistes proposées par D. Monniaux [Mon01a, Mon01b].

Les méthodes de différentiation automatique semblent être de bonnes candidates pour l'analyse statique même si elles sont limitées par le fait que certains termes d'erreur sont négligés et

par l'approximation linéaire. Il est envisageable de définir une sémantique $[\![.]\!]_{\mathbb{ED}}$ qui ne négligerait plus ces termes et de majorer les dérivées suffisamment précisément pour obtenir des termes d'erreur proches de la réalité. De premiers travaux prometteurs ont été menés dans cette direction par A. Chapoutot [Cha05].

Enfin, nous reviendrons dans le chapitre suivant sur des analyses statiques utilisant une généralisation de la sémantique $[\![.]\!]_{\mathbb{E}}$ introduite à la section 2.3.

2.8 Perspectives

Comme nous l'avons vu au cours de ce chapitre, chaque sémantique dynamique pour la précision numérique calcule une propriété bien particulière qu'il est important d'énoncer clairement préalablement à son utilisation à des fins de vérification et de validation de programmes. Les différences peuvent être subtiles : nous avons vu par exemple que les méthodes d'intervalles calculaient des propriétés très différentes selon la précision utilisée pour les bornes.

Une analyse statique ne faisant que calculer une sur-approximation, pour un ensemble d'exécutions, des résultats produits par un programme dans une sémantique dynamique donnée, le choix de cette dernière influe fortement sur l'intérêt de l'analyse que l'on peut obtenir. Or, à ce jour, indépendamment des techniques permettant de les calculer, les propriétés permettant d'affirmer qu'un traitement numérique est sûr ne sont pas clairement établies : erreur absolue ou erreur relative, erreur directe ou erreur inverse, de nombreux critères peuvent être imaginés. De plus, on peut observer que certaines approches sont mieux adaptées à la détection de certains phénomènes numériques. Par exemple, l'erreur relative associée à un calcul contenant une élimination catastrophique sera importante, tandis que l'erreur absolue sera faible. A contrario, l'erreur relative est moins bien adaptée que l'erreur absolue pour la détection d'absorptions.

La définition de nouvelles sémantiques dynamiques pour la précision numérique est un axe de recherche important. Nous avons vu que de telles sémantiques pouvaient être obtenues, entre autres, par combinaison des méthodes présentées dans ce chapitre. Par exemple, les couplages entre erreur globale et arithmétique stochastique ou erreur globale et différentiation automatique semblent prometteurs. Un travail important reste cependant à fournir pour définir précisément ces nouvelles méthodes et estimer leur adéquation à la détection d'un phénomène numérique donné. Nous reviendrons sur la pertinence des critères de correction pour la précision numérique à la section 6.2.

Enfin, la gestion des divergences de flots de contrôle demeure un sujet difficile et très mal traité à ce jour. Ce problème survient lorsque, dans un programme, un branchement conditionnel est évalué différemment dans l'arithmétique de la machine, disons $\mathbb{F}$, et dans l'arithmétique idéale, disons $\mathbb{R}$ ou son approximation calculée par une méthode quelconque. Dans ce cas, les erreurs associées aux opérations arithmétiques ne sont plus calculées de façon pertinente par la plupart des méthodes : $[\![.]\!]_{\mathbb{E}}$ et $[\![.]\!]_{\mathbb{D}}$ suivent uniquement la branche flottante et $[\![.]\!]_{\mathbb{S}}$ uniquement la branche réelle (éventuellement en signalant le risque de divergence par une alarme). $[\![.]\!]_{\mathbb{I}}$, avec des bornes de même précision que celle de la machine cible peut permettre, par analyse statique (c.à.d en faisant l'union des résultats des résultats obtenus sur chaque branche aux points de confluence), de calculer une majoration de l'erreur sûre mais imprécise. Aucune de ces solution n'est vraiment satisfaisante.

Chapitre 3

Les séries d'erreurs

3.1 Introduction

Il est souvent difficile pour un programmeur de déterminer la qualité du résultat d'un traitement numérique et de comprendre quelles opérations sont responsables des principales pertes de précision [Gol91, Knu97]. Dans ce chapitre, nous présentons les bases théoriques de Fluctuat [GMP02, Mar04], un analyseur permettant de détecter les sources d'imprécision les plus importantes survenant au cours d'un calcul. Notre approche consiste à définir une sémantique non standard précise $[\![.]\!]^{\mathcal{L}^*}$, fondée sur la norme IEEE 754, et, ensuite, à définir de nouvelles sémantiques approchées, plus utiles et plus efficaces en pratique.

Toutes ces sémantiques calculent la propagation des erreurs d'arrondi survenant en cours de calcul. On obtient ainsi la contribution à l'erreur globale sur un résultat des imprécisions créées par les arrondis successifs. Nous présentons tout d'abord brièvement, à la section 3.2, la sémantique standard définissant l'évaluation des expressions arithmétiques en nombres flottants. Puis, à la section 3.3, nous présentons la sémantique la plus générale, notée $[\![.]\!]^{\mathcal{L}^*}$, qui permet de connaître la contribution à l'erreur globale des erreurs dues à chaque opération élémentaire, et à tous les ordres. Ces informations permettent de comprendre les raisons pour lesquelles le résultat d'un traitement numérique est imprécis.

Nous définissons ensuite, à la section 3.4, une famille de sémantiques notée $([\![.]\!]^{\mathcal{L}^n})_{n\in\mathbb{N}}$, afin de calculer la contribution à l'erreur globale des erreurs des n premiers ordres, tout en évaluant globalement l'erreur d'ordre supérieur à n. Il s'agit là d'une première approximation de la sémantique $[\![.]\!]^{\mathcal{L}^*}$, permettant d'utiliser des domaines plus compacts, et, par conséquent de réduire le temps de calcul et l'espace mémoire nécessaires à l'analyse d'un programme.

Une seconde approximation, présentée à la section 3.5 et effectuée aussi pour améliorer les performances de l'analyseur, consiste à regrouper de façon cohérente certains termes d'erreur. On obtient ainsi une nouvelle famille de sémantiques, telle que, si $\mathcal{J}$ est une partition des points de contrôle du programme, $[\![.]\!]^{\mathcal{J}^n}$ calcule la propagation des erreurs des n premiers ordres dues aux arrondis survenant dans les portions de code définies par $\mathcal{J}$. On peut ainsi étudier la contribution d'une ligne de code, d'une fonction ou encore d'une formule intermédiaire à l'erreur globale sur le résultat d'un calcul.

3.2 Sémantique standard

Afin d'étudier la propagation des erreurs au cours d'un calcul, nous supposons que les expressions arithmétiques sont annotées par des étiquettes uniques ℓ, ℓ_1, ℓ_2, etc. comme défini à l'équation (3.1) :

$$a^\ell ::= r^\ell \mid a_0^{\ell_0} +^\ell a_1^{\ell_1} \mid a_0^{\ell_0} -^\ell a_1^{\ell_1} \mid a_0^{\ell_0} \times^\ell a_1^{\ell_1} \mid a_0^{\ell_0} \div^\ell a_1^{\ell_1} \mid \sqrt{a_0^{\ell_0}}^\ell \tag{3.1}$$

r représente une valeur scalaire. Dans les sémantiques présentées dans ce chapitre, les étiquettes sont utilisées pour identifier les erreurs survenant en cours de calcul. Par exemple, pour l'expression $r_0^{\ell_0} +^\ell r_1^{\ell_1}$ les erreurs initiales correspondant à r_0 et r_1 sont attachées à des variables formelles $\vec{\varepsilon}_{\ell_0}$ et $\vec{\varepsilon}_{\ell_1}$ et l'erreur due à l'arrondi du résultat de l'addition est attaché à la variable formelle $\vec{\varepsilon}_\ell$.

L'ensemble des étiquettes utilisées pour annoter un programme est noté $\mathcal{L}$ et nous utilisons la sémantique opérationnelle par petits pas définie ci-dessous, dans laquelle le symbole $\Diamond$ représente indifféremment une des opérations $+$, $-$, $\times$ ou $\div$.

$$\frac{a_0^{\ell_0} \to a_2^{\ell_2}}{a_0^{\ell_0} \Diamond^\ell a_1^{\ell_1} \to a_2^{\ell_2} \Diamond^\ell a_1^{\ell_1}} \qquad \frac{a_1^{\ell_1} \to a_2^{\ell_2}}{r_0^{\ell_0} \Diamond^\ell a_1^{\ell_1} \to r_0^{\ell_0} \Diamond^\ell a_2^{\ell_2}}$$

$$\frac{r = r_0 \Diamond^\ell r_1}{r_0^{\ell_0} \Diamond^\ell r_1^{\ell_1} \to r^\ell} \qquad \frac{a_0^{\ell_0} \to a_1^{\ell_1}}{\sqrt{a_0^{\ell_0}}^\ell \to \sqrt{a_1^{\ell_1}}^\ell} \qquad \frac{r = \sqrt{r_0}}{\sqrt{r_0^{\ell_0}}^\ell \to r^\ell}$$

Dans la suite de ce chapitre, nous allons définir plusieurs domaines pour les valeurs et nous allons proposer plusieurs sémantiques non standard pour les opérations élémentaires. Nous nous concentrons sur la sémantique des expressions arithmétiques, la gestion du reste du langage étant classique [GMP01]. A ce niveau, le seul point méritant d'être mentionné concerne les tests des conditionnelles et des boucles, lorsque le résultat d'une comparaison dans la sémantique des nombres flottants diffère de celui que l'on obtient dans la sémantique des réels. Dans ce cas, $[\![.]\!]_{\mathbb{F}}$ et $[\![.]\!]_{\mathbb{R}}$ conduisent à des exécutions de morceaux de code différents et nous suivons le chemin choisi par la sémantique $[\![.]\!]_{\mathbb{F}}$, de façon à mimer le comportement que le programme aurait dans un système embarqué. Cependant, les termes d'erreurs ne peuvent plus être calculés (bien qu'il serait peut-être possible de continuer à les calculer pour des divergences locales des deux flots de contrôle).

Précisons enfin que les étiquettes sont uniquement attachées aux expressions arithmétiques qui seules génèrent de nouveaux termes d'erreur. Une étiquette ℓ correspond à l'occurrence syntaxique d'une donnée ou d'un opérateur dans le code. Si une opération $\Diamond^\ell$ est exécutée plusieurs fois, alors le coefficient attaché à $\vec{\varepsilon}_\ell$ contient la somme des erreurs d'arrondi dues aux différentes exécutions de $\Diamond$. Par exemple, si l'affectation $x = r_1^{\ell_1} \Diamond^\ell r_2^{\ell_2}$ apparaît dans un corps de boucle, alors dans la série d'erreurs attachée à x au bout de n itérations, le coefficient de $\vec{\varepsilon}_\ell$ contient la somme des erreurs d'arrondi introduites par $\Diamond^\ell$ lors des n premières étapes.

3.3 Sémantique générale des séries d'erreurs

Nous présentons ici la sémantique générale des séries d'erreurs, notée $[\![.]\!]^{\mathcal{L}^*}$ et initialement introduite dans [Mar02a, Mar06]. Cette sémantique détaille la contribution à l'erreur globale sur

le résultat d'un calcul de toutes les erreurs de tous ordres. Les séries utilisées par $[\![.]\!]^{\mathcal{L}^*}$ peuvent contenir trop de termes pour être calculées efficacement en pratique et nous proposerons au cours des sections 3.4 et 3.5 des versions plus abstraites de cette sémantique. Cependant, la correction des sémantiques abstraites repose sur celle de $[\![.]\!]^{\mathcal{L}^*}$, que nous allons tout d'abord étudier.

3.3.1 Définition de $[\![.]\!]^{\mathcal{L}^*}$

La sémantique $[\![.]\!]^{\mathcal{L}^*}$ calcule la propagation de toutes les erreurs d'arrondi de tous ordres survenant au cours d'un calcul. Intuitivement, une constante r^ℓ d'un programme est représentée par la série $f\vec{\varepsilon} + \omega^\ell \vec{\varepsilon}_\ell$, où $f = \uparrow_\circ (r)$ est le nombre flottant approchant r et $\omega^\ell = \downarrow_\circ (r)$ est l'erreur d'arrondi. Rappelons que les fonctions $\uparrow_\circ$ et $\downarrow_\circ$ ont été définies à la section 2.2. f et ω^ℓ sont les coefficients numériques d'une série et $\vec{\varepsilon}$ et $\vec{\varepsilon}_\ell$ sont des variables formelles correspondant au nombre flottant et à l'erreur due au point ℓ.

Un nombre r d'étiquette ℓ_0 et représentant le résultat de l'évaluation de l'expression arithmétique $a_0^{\ell_0}$ est représenté par la série

$$r^{\ell_0} = f\vec{\varepsilon} + \sum_{u \in \overline{\mathcal{L}^+}} \omega^u \vec{\varepsilon}_u \tag{3.2}$$

dans laquelle $\mathcal{L}$ est l'ensemble des étiquettes de $a_0^{\ell_0}$ et $\overline{\mathcal{L}^+}$ est un sous-ensemble des mots sur l'alphabet $\mathcal{L}$. $\overline{\mathcal{L}^+}$ est formellement défini plus tard dans cette section. Dans l'équation (3.2), f est le nombre flottant approchant r, que l'on attache toujours à la variable formelle $\vec{\varepsilon}$. Soit ℓ un mot d'une lettre de $\overline{\mathcal{L}^+}$. Dans la série formelle $\sum_{u \in \overline{\mathcal{L}^+}} \omega^u \vec{\varepsilon}_u$, $\omega^\ell \vec{\varepsilon}_\ell$ représente la contribution à l'erreur globale de l'erreur du premier ordre survenue au point ℓ pendant le calcul de $a_0^{\ell_0}$. $\omega^\ell \in \mathbb{R}$ est la valeur numérique de ce terme d'erreur, tandis que $\vec{\varepsilon}_\ell$ est une variable formelle permettant d'identifier l'origine de cette imprécision.

Etant donné un mot $u = \ell_1 \ell_2 \dots \ell_n \in \overline{\mathcal{L}^+}$ de longueur $n \geq 2$, $\vec{\varepsilon}_u$ est la variable formelle du terme d'erreur d'ordre n apparu par combinaison des erreurs commises aux points $\ell_1, \dots, \ell_n$. Par exemple, si l'on considère la multiplication au point ℓ_3 de deux valeurs initiales $r_1^{\ell_1} = (f_1\vec{\varepsilon} + \omega_1^{\ell_1}\vec{\varepsilon}_{\ell_1})$ et $r_2^{\ell_2} = (f_2\vec{\varepsilon} + \omega_2^{\ell_2}\vec{\varepsilon}_{\ell_2})$, on obtient :

$$r_1^{\ell_1} \times^{\ell_3} r_2^{\ell_2} = \uparrow_\circ (f_1 f_2)\vec{\varepsilon} + f_2\omega_1^{\ell_1}\vec{\varepsilon}_{\ell_1} + f_1\omega_2^{\ell_2}\vec{\varepsilon}_{\ell_2} + \omega_1^{\ell_1}\omega_2^{\ell_2}\vec{\varepsilon}_{\ell_1\ell_2} + \downarrow_\circ (f_1 f_2)\vec{\varepsilon}_{\ell_3} \tag{3.3}$$

Comme le montre l'équation (3.3), le nombre flottant résultat de cette multiplication a pour valeur $\uparrow_\circ (f_1 f_2)$. Ceci est garanti par la norme IEEE 754 comme nous avons pu le voir à la section 2.2. Les erreurs initiales $\omega_1^{\ell_1}\vec{\varepsilon}_{\ell_1}$ et $\omega_2^{\ell_2}\vec{\varepsilon}_{\ell_2}$ sont multipliées par f_2 et f_1 respectivement. De plus, la multiplication génère une nouvelle erreur d'arrondi, $\downarrow_\circ (f_1 f_2)$, que l'on attache naturellement à l'étiquette $\vec{\varepsilon}_{\ell_3}$. Finalement, ce calcul fait aussi apparaître une erreur du second ordre de magnitude $\omega_1^{\ell_1}\omega_2^{\ell_2}$. Nous l'attachons à la variable formelle $\vec{\varepsilon}_{\ell_1\ell_2}$ pour garder trace de son origine : la combinaison des erreurs dues à ℓ_1 et ℓ_2.

Intuitivement, nous souhaitons identifier les termes d'erreurs $\vec{\varepsilon}_{\ell_1\ell_2}$ et $\vec{\varepsilon}_{\ell_2\ell_1}$ qui correspondent tous deux à l'erreur du second ordre due à ℓ_1 et ℓ_2. De façon générale, étant donnée une suite de n lettres $\ell_1, \dots, \ell_n$, nous souhaitons identifier tous les termes d'erreur d'ordre n correspondant à des mots obtenus par permutation des lettres $\ell_1, \dots, \ell_n$. Formellement, soit $\mathcal{L}^*$ l'ensemble des mots de longueur finie sur l'alphabet $\mathcal{L}$. Le mot vide est noté ϵ, $|u|$ est la longueur de u et $u \cdot v$ est la concaténation de u et v. Nous utilisons la relation d'équivalence $\sim$ qui identifie tous les mots

composés des mêmes lettres : soient $u, v \in \mathcal{L}^*$, $u \sim v$ si et seulement si, pour toute lettre $\ell \in \mathcal{L}$, le nombre d'occurrences de ℓ dans u est égal au nombre d'occurrences de ℓ dans v.

Soit $\overline{\mathcal{L}^*}$ l'ensemble quotient $\mathcal{L}^*/\sim$. Nous prenons comme élément représentatif d'une classe le plus petit mot u selon l'ordre lexicographique. On note $\overline{\mathcal{L}^+}$ l'ensemble $\overline{\mathcal{L}^*} \setminus \{\epsilon\}$. Pour tout mot $u \in \overline{\mathcal{L}^+}$, la variable formelle $\vec{\varepsilon}_u$ correspond à une erreur d'ordre n, où $n = |u|$. $\vec{\varepsilon}_\epsilon = \vec{\varepsilon}$ est la variable formelle utilisée pour le terme flottant f de la série, le seul connu par un ordinateur lors d'une exécution non instrumentée d'un programme. Nous utilisons indifféremment les termes f et ω^ϵ pour noter le coefficient de la variable $\vec{\varepsilon}_\epsilon$.

On note :

$$\mathcal{F}(\mathcal{D}, \overline{\mathcal{L}^*}) = \{ \sum_{u \in \overline{\mathcal{L}^*}} \omega^u \vec{\varepsilon}_u \ : \ \forall u,\ \omega^u \in \mathcal{D}\}$$

l'ensemble des séries dont les variables formelles sont indicées par des éléments de $\overline{\mathcal{L}^*}$ et dont les coefficients ω^u appartiennent au domaine $\mathcal{D}$. Dans cette section ainsi que dans les sections 3.4 et 3.5, nous ne considérons que des sémantiques concrètes et, par conséquent, il n'est pas nécessaire de disposer d'un ordre informationnel pour comparer les éléments de $\mathcal{F}(\mathcal{D}, \overline{\mathcal{L}^*})$. Un tel ordre sera introduit dans la section 4.2, pour l'analyse statique.

La sémantique $[\![.]\!]^{\mathcal{L}^*}$ utilise le domaine $\mathbb{R}^{\mathcal{L}^*} = \mathcal{F}(\mathbb{R}, \overline{\mathcal{L}^*})$ des séries d'erreurs. Les opérations élémentaires entre valeurs de $\mathbb{R}^{\mathcal{L}^*}$ sont définies à la figure 3.1. Dans cette section, $\mathcal{W}$ représente le monoïde libre commutatif $\mathcal{W} = (\mathcal{L}^*/\sim, \epsilon, \cdot)$, dont les éléments appartiennent à $\mathcal{L}^*/\sim$, avec le mot vide ϵ comme élément neutre et muni de l'opérateur associatif $\cdot$ de concaténation. $\mathcal{W}^+$ représente l'ensemble $\{x \in \mathcal{W} \ : \ x \neq \epsilon\}$. Au cours des sections 3.4 et 3.5, nous utiliserons d'autres monoïdes, munis d'autres opérateurs associatifs.

Dans la figure 3.1, la série $\sum_{u \in \overline{\mathcal{L}^*}} \omega^u \vec{\varepsilon}_u$ correspondant au résultat d'une opération $\Diamond^\ell$ contient la combinaison des erreurs des opérandes ainsi qu'un nouveau terme d'erreur $\downarrow_\circ (f_1 \Diamond_{\mathbb{R}} f_2)\vec{\varepsilon}_\ell$ provenant de l'arrondi de l'opération $\Diamond_{\mathbb{F}}$ effectué au point de contrôle ℓ. Les règles pour l'addition et la soustraction sont naturelles : les erreurs sont ajoutées ou retranchées terme à terme et un nouveau terme d'indice ℓ est ajouté pour l'arrondi du résultat.

La multiplication requiert plus d'attention car elle génère de nouveaux termes d'erreur d'ordre supérieur, par multiplication des termes des opérandes. Par exemple, pour $\ell_1, \ell_2 \in \mathcal{L}$, et pour des termes d'erreur du premier ordre $\omega_1^{\ell_1}\vec{\varepsilon}_{\ell_1}$ and $\omega_2^{\ell_2}\vec{\varepsilon}_{\ell_2}$, l'opération $\omega_1^{\ell_1}\vec{\varepsilon}_{\ell_1} \times \omega_2^{\ell_2}\vec{\varepsilon}_{\ell_2}$ génère une erreur du second ordre notée $\omega_1^{\ell_1}\omega_2^{\ell_2}\vec{\varepsilon}_{\ell_1\ell_2}$.

La série formelle résultant de l'opération r^{-1^ℓ} est obtenue au moyen d'un développement en série :

$$\frac{1}{1+x} = \sum_{n \geq 0} (-1)^n x^n \quad \text{pour tout } x \text{ tel que } -1 < x < 1$$

Soit $r = f\vec{\varepsilon} + e$ où $e = \sum_{u \in \mathcal{W}^+} \omega^u \vec{\varepsilon}_u$. Comme pour la sémantique $[\![.]\!]_{\mathbb{E}}$ du chapitre 2, nous avons :

$$\frac{1}{f+e} = \frac{1}{f} \times \frac{1}{1+\frac{e}{f}} = \frac{1}{f} \times \sum_{n \geq 0} (-1)^n \frac{e^n}{f^n}$$

et nous obtenons l'équation (3.7) de la figure 3.1. Remarquons que, la série utilisée devant être convergente, la technique utilisée ci-dessus ne peut pas être appliquée dans tous les cas. Pour la fonction inverse, le disque de convergence de la série $\sum_{n \geq 0}(-1)^n x^n = (1+x)^{-1}$ a pour rayon $\rho = 1$. Aussi, dans l'équation (3.7), nous supposons que $-1 < \sum_{u \in \mathcal{W}^+} \frac{\omega^u}{f_1} < 1$. Cette contrainte

$$r_1 +^\ell r_2 \stackrel{\text{def}}{=} \uparrow_\circ (f_1+f_2)\vec{\varepsilon} + \sum_{u\in\mathcal{W}^+} (\omega_1^u+\omega_2^u)\vec{\varepsilon}_u + \downarrow_\circ (f_1+f_2)\vec{\varepsilon}_\ell \tag{3.4}$$

$$r_1 -^\ell r_2 \stackrel{\text{def}}{=} \uparrow_\circ (f_1-f_2)\vec{\varepsilon} + \sum_{u\in\mathcal{W}^+} (\omega_1^u-\omega_2^u)\vec{\varepsilon}_u + \downarrow_\circ (f_1-f_2)\vec{\varepsilon}_\ell \tag{3.5}$$

$$r_1 \times^\ell r_2 \stackrel{\text{def}}{=} \uparrow_\circ (f_1 f_2)\vec{\varepsilon} + \sum_{\substack{u\in\mathcal{W}\\ v\in\mathcal{W}\\ |u\cdot v|>0}} \omega_1^u\omega_2^v\vec{\varepsilon}_{u\cdot v} + \downarrow_\circ (f_1 f_2)\vec{\varepsilon}_\ell \tag{3.6}$$

$$(r_1)^{-1^\ell} \stackrel{\text{def}}{=} \uparrow_\circ (f_1^{-1})\vec{\varepsilon} + \frac{1}{f_1}\sum_{n\geq 1}(-1)^n \left(\sum_{u\in\mathcal{W}^+}\frac{\omega_1^u}{f_1}\vec{\varepsilon}_u\right)^n + \downarrow_\circ (f_1^{-1})\vec{\varepsilon}_\ell \tag{3.7}$$

$$r_1 \div^\ell r_2 \stackrel{\text{def}}{=} \uparrow_\circ \left(\frac{f_1}{f_2}\right)\vec{\varepsilon} + \downarrow_\circ \left(\frac{f_1}{f_2}\right)\vec{\varepsilon}_\ell + \tag{3.8}$$

$$\sum_{\substack{n\geq 0,\ u\in\mathcal{W}\\ d_{v_1}+\ldots+d_{v_k}=n\\ d_{v_1},\ldots,d_{v_k}\geq 0\\ |uv_1^{d_{v_1}}\ldots v_k^{d_{v_k}}|>0}} (-1)^n \times \frac{\omega_1^u}{f_2} \times \frac{n!}{d_{v_1}!\ldots d_{v_k}!} \times \prod_{v\in\mathcal{W}^+}\left(\frac{\omega_2^v}{f_2}\right)^{d_v} \vec{\varepsilon}_{uv_1^{d_{v_1}}\ldots v_k^{d_{v_k}}}$$

$$\sqrt{r_1}^\ell \stackrel{\text{def}}{=} \uparrow_\circ (\sqrt{f_1})\vec{\varepsilon} + \sum_{n\geq 1}\left[\frac{\frac{1}{2}(-\frac{1}{2})\ldots(\frac{3}{2}-n)}{n!} \times \sqrt{f_1} \times \left(\sum_{u\in\mathcal{W}^+}\frac{\omega_1^u}{f_1}\vec{\varepsilon}_u\right)^n\right] + \downarrow_\circ (\sqrt{f_1})\vec{\varepsilon}_\ell \tag{3.9}$$

FIG. 3.1 – *Définition des opérations pour la sémantique* $[\![.]\!]^{\mathcal{L}^*}$.

impose que la somme des termes d'erreur d'une série soit inférieure, en valeur absolue, au nombre flottant auquel ils sont attachés.

La sémantique de la division est obtenue par combinaison des équations (3.6) et (3.7). De fait, $r_1 \div^\ell r_2 = r_1 \times^\ell (r_2)^{-1^\ell}$, sauf en ce qui concerne l'arrondi $\downarrow_\circ (f_1 \div f_2)\vec{\varepsilon}_\ell$. En effet, suivant la norme IEEE 754, l'erreur d'arrondi introduite par la division est exactement $\downarrow_\circ (f_1 \div f_2)$, ce qui diffère du terme introduit par $r_1 \times^\ell (r_2)^{-1^\ell}$. Par conséquent, la division doit être définie en une seule étape. Dans l'équation (3.8), $d_1, \ldots, d_k$ sont des entiers positifs ou nuls utilisés pour les coefficients multinomiaux introduits par le développement :

$$\sum_{n\geq 0}(t_1+\ldots+t_k)^n = \sum_{n\geq 0}\sum_{\substack{d_1+\ldots+d_k=n\\ d_1\geq 0,\ldots,d_k\geq 0}} \frac{n!}{d_1!\ldots d_k!}t_1^{d_1}\ldots t_k^{d_k}$$

Enfin, dans l'équation (3.9), la sémantique de la fonction racine carrée est obtenue de façon similaire à celle de la fonction inverse. Rappelons cependant que les autres fonctions élémentaires (trigonométriques par exemple) sont plus difficiles à gérer, la norme IEEE 754 ne précisant pas la

façon dont elles sont arrondies. Dans ce cas, des hypothèses dépendant du système utilisé doivent être posées pour définir le comportement de la fonction $\downarrow_\circ$.

Pour une expression $a_0^{\ell_0}$ telle que $\text{Lab}(a_0^{\ell_0}) \subseteq \mathcal{L}$, la sémantique $[\![.]\!]^{\mathcal{L}^*}$ est définie par le domaine $\mathbb{R}^{\mathcal{L}^*}$ pour les valeurs et par les règles de réduction $\rightarrow^{\mathcal{L}}$ obtenues en substituant les opérateurs de $\mathbb{R}^{\mathcal{L}^*}$ aux opérateurs $\Diamond$ utilisés dans les règles de la section 3.2.

3.3.2 Correction de $[\![.]\!]^{\mathcal{L}^*}$

Pour la correction des opérateurs définis à la figure 3.1, nous pouvons aisément vérifier que les membres gauches et droits des équations (3.4) à (3.9) représentent les mêmes quantités. Autrement dit, nous pouvons nous assurer que pour tout opérateur $\Diamond$ et pour toutes valeurs $r_1 = \sum_{u\in\mathcal{W}} \omega_1^u \vec{\varepsilon}_u$, $r_2 = \sum_{u\in\mathcal{W}} \omega_2^u \vec{\varepsilon}_u$ et $r = r_1 \Diamond^\ell r_2 = \sum_{u\in\mathcal{W}} \omega^u \vec{\varepsilon}_u$, nous avons :

$$\sum_{u\in\mathcal{W}} \omega^u = \left(\sum_{u\in\mathcal{W}} \omega_1^u\right) \Diamond \left(\sum_{u\in\mathcal{W}} \omega_2^u\right) \tag{3.10}$$

Cependant, ce critère de correction est trop faible car il ne permet pas de garantir que les termes d'erreur sont un à un correctement propagés au cours du calcul. Par exemple, l'équation (3.11) propose une définition erronée de l'addition, car les erreurs attachées à $\vec{\varepsilon}_{\ell_1}$ et $\vec{\varepsilon}_{\ell_2}$ sont interverties dans le résultat.

$$\begin{array}{c} (f_1\vec{\varepsilon} + \omega^{\ell_1}\vec{\varepsilon}_{\ell_1}) +^\ell (f_2\vec{\varepsilon} + \omega^{\ell_2}\vec{\varepsilon}_{\ell_2}) \\ \overset{\text{faux!}}{=} \\ \uparrow_\circ (f_1+f_2)\vec{\varepsilon} + \omega^{\ell_1}\vec{\varepsilon}_{\ell_2} + \omega^{\ell_2}\vec{\varepsilon}_{\ell_1} + \downarrow_\circ (f_1+f_2)\vec{\varepsilon}_\ell \end{array} \tag{3.11}$$

Définir l'addition à partir d'une généralisation de l'équation (3.11) nous amènerait à une formule indésirable, vérifiant pourtant le critère de correction exprimé par l'équation (3.10).

Nous souhaitons prouver que de telles confusions n'ont pas été commises dans la définition des opérations élémentaires, notamment pour la multiplication et la division. Pour cela, nous allons comparer la sensibilité aux variations des coefficients ω_1^u et ω_2^u des termes d'erreur présents dans les membres gauches et droits des équations (3.4) à (3.9). La sensibilité de r_1 et r_2 est donnée par $\frac{\partial^n}{\partial\omega_{k_1}^{u_1}\ldots\omega_{k_n}^{u_n}}$ pour un sous-ensemble fini $\omega_{k_1}^{u_1}, \ldots, \omega_{k_n}^{u_n}$ des coefficients, avec $k_i = 1$ ou 2, $u_i \in \mathcal{W}^+$, $1 \leq i \leq n$. Par exemple, ce critère permet de détecter que l'équation (3.11) n'est pas correcte, les deux membres n'ayant pas la même sensibilité à ω^{ℓ_1} :

$$\frac{\partial}{\partial\omega^{\ell_1}} \left((f_1\vec{\varepsilon} + \omega^{\ell_1}\vec{\varepsilon}_{\ell_1}) +^\ell (f_2\vec{\varepsilon} + \omega^{\ell_2}\vec{\varepsilon}_{\ell_2}) \right) = \vec{\varepsilon}_{\ell_1} \tag{3.12}$$

tandis que :

$$\frac{\partial}{\partial\omega^{\ell_1}} \left(\uparrow_\circ (f_1+f_2)\vec{\varepsilon} + \omega^{\ell_1}\vec{\varepsilon}_{\ell_2} + \omega^{\ell_2}\vec{\varepsilon}_{\ell_1} + \downarrow_\circ (f_1+f_2)\vec{\varepsilon}_\ell \right) = \vec{\varepsilon}_{\ell_2} \tag{3.13}$$

Tout d'abord, nous introduisons le lemme 5 concernant les dérivées partielles du premier ordre.

Lemme 5 *Soit $\Diamond \in \{+, -, \times, \div\}$ une opération élémentaire et soit $\Diamond^\ell \in \{+^\ell, -^\ell, \times^\ell, \div^\ell\}$ l'opération non standard correspondante telle qu'elle est définie par les équations (3.4) à (3.9). Pour tous $r_1 = \sum_{u\in\mathcal{W}} \omega_1^u \vec{\varepsilon}_u$, $r_2 = \sum_{u\in\mathcal{W}} \omega_2^u \vec{\varepsilon}_u$ et pour tout $u_0 \in \mathcal{W}^+ \setminus \{\ell\}$ nous avons :*

$$\frac{\partial(r_1\Diamond r_2)}{\partial\omega_1^{u_0}} = \frac{\partial(r_1\Diamond^\ell r_2)}{\partial\omega_1^{u_0}} \quad \textit{et} \quad \frac{\partial(r_1\Diamond r_2)}{\partial\omega_2^{u_0}} = \frac{\partial(r_1\Diamond^\ell r_2)}{\partial\omega_2^{u_0}} \tag{3.14}$$

PREUVE. (multiplication)

Nous avons d'une part :

$$\frac{\partial(r_1 \times^\ell r_2)}{\partial \omega_1^{u_0}} = \frac{\partial}{\partial \omega_1^{u_0}} \left(\sum_{u,v \in \mathcal{W}} \omega_1^u \omega_2^v \vec{\varepsilon}_{uv} \right)$$

et en utilisant l'égalité :

$$\sum_{u,v \in \mathcal{W}} \omega_1^u \omega_2^v \vec{\varepsilon}_{uv} = \sum_{v \in \mathcal{W}} \omega_1^{u_0} \omega_2^v \vec{\varepsilon}_{u_0 v} + \sum_{u \in \mathcal{W} \setminus \{u_0\}, v \in \mathcal{W}} \omega_1^u \omega_2^v \vec{\varepsilon}_{uv}$$

on obtient :

$$\frac{\partial(r_1 \times^\ell r_2)}{\partial \omega_1^{u_0}} = \sum_{v \in \mathcal{W}} \omega_2^v \vec{\varepsilon}_{u_0 v}$$

D'autre part :

$$\frac{\partial}{\partial \omega_1^{u_0}} \left(\sum_{u \in \mathcal{W}} \omega_1^u \vec{\varepsilon}_u \times \sum_{v \in \mathcal{W}} \omega_2^u \vec{\varepsilon}_u \right) =$$

$$\sum_{u \in \mathcal{W}} \omega_1^u \vec{\varepsilon}_u \times \frac{\partial}{\partial \omega_1^{u_0}} \left(\sum_{v \in \mathcal{W}} \omega_2^v \vec{\varepsilon}_v \right) + \sum_{v \in \mathcal{W}} \omega_2^v \vec{\varepsilon}_v \times \frac{\partial}{\partial \omega_1^{u_0}} \left(\sum_{u \in \mathcal{W}} \omega_1^u \vec{\varepsilon}_u \right) =$$

$$0 + \left(\sum_{v \in \mathcal{W}} \omega_2^v \vec{\varepsilon}_v \right) \times \vec{\varepsilon}_{u_0} = \sum_{v \in \mathcal{W}} \omega_2^v \vec{\varepsilon}_{u_0 v}$$

□

Le lemme 5 garantit que la sensibilité aux variations d'un seul terme d'erreur dans les entrées est correctement propagée dans le résultat. La proposition 6 présentée ci-dessous généralise ce résultat à la sensibilité aux variations d'un ensemble fini de coefficients.

Proposition 6 *Soit $\Diamond \in \{+, -, \times, \div\}$ une opération élémentaire et soit $\Diamond^\ell \in \{+^\ell, -^\ell, \times^\ell, \div^\ell\}$ l'opérateur correspondant tel que défini par les équations (3.4) à (3.9). Pour tous $r_1 = \sum_{u \in \mathcal{W}} \omega_1^u \vec{\varepsilon}_u$, $r_2 = \sum_{u \in \mathcal{W}} \omega_2^u \vec{\varepsilon}_u$ et pour tous $\omega_{k_1}^{u_1}, \ldots, \omega_{k_n}^{u_n}$, $k_i = 1$ ou 2, $u_i \in \mathcal{W}^+ \setminus \{\ell\}$, $1 \leq i \leq n$, nous avons :*

$$\frac{\partial^n (r_1 \Diamond r_2)}{\partial \omega_{k_1}^{u_1} \ldots \omega_{k_n}^{u_n}} = \frac{\partial^n (r_1 \Diamond^\ell r_2)}{\partial \omega_{k_1}^{u_1} \ldots \omega_{k_n}^{u_n}} \tag{3.15}$$

PREUVE.

La preuve se fait par récurrence, en utilisant le lemme 5.

□

3.4 Restriction aux erreurs des n premiers ordres

La sémantique $[\![.]\!]^{\mathcal{L}^*}$, présentée à la section 3.3, calcule la propagation des erreurs de n'importe quel ordre survenant au cours d'un traitement numérique. Il s'agit d'un modèle général mais il est couramment admis qu'en pratique, les erreurs d'ordres supérieurs à un (ou exceptionnellement deux) sont négligeables. Cependant, même si d'un point de vue pratique nous souhaitons

uniquement connaître la contribution des erreurs des n premiers ordres à l'erreur globale sur le résultat d'un calcul, il est nécessaire de vérifier, pour des questions de sûreté, que les erreurs d'ordre supérieur sont effectivement négligeables.

Dans cette section, nous présentons une famille $(\llbracket.\rrbracket^{\mathcal{L}^n})_{n\in\mathbb{N}}$ de sémantiques telles que la sémantique $\llbracket.\rrbracket^{\mathcal{L}^n}$ détaille la contribution à l'erreur globale des erreurs des n premiers ordres et regroupe, dans le coefficient d'une seule variable formelle spéciale, la contribution globale des erreurs d'ordre supérieur à n. Une valeur r de $\llbracket.\rrbracket^{\mathcal{L}^n}$ est représentée par :

$$r = f\vec{\varepsilon} + \sum_{u\in\overline{\mathcal{L}^+},\ |u|\leq n} \omega^u\vec{\varepsilon}_u + \omega^{hi}\vec{\varepsilon}_{hi} \tag{3.16}$$

Le terme $\omega^{hi}\vec{\varepsilon}_{hi}$ de cette série agglomère les erreurs élémentaires d'ordre supérieur à n. Ainsi, en commençant par $n = 1$, il est possible d'examiner la contribution à l'erreur globale des erreurs du premier ordre. Si le coefficient ω^{hi} est négligeable, la sémantique $\llbracket.\rrbracket^{\mathcal{L}^1}$ fournit assez d'informations pour comprendre la nature de l'erreur. Dans le cas contraire, $\llbracket.\rrbracket^{\mathcal{L}^1}$ indique que l'erreur d'ordre supérieur est importante sans en indiquer la provenance. Cette information supplémentaire peut être obtenue en utilisant la sémantique $\llbracket.\rrbracket^{\mathcal{L}^n}$ pour une certaine valeur de n supérieure à un.

Soit $\mathcal{L}^n$ l'ensemble des mots de longueur au plus n sur l'alphabet $\mathcal{L}$ et soit $\overline{\mathcal{L}^n} = (\mathcal{L}^n/\sim) \cup \{hi\}$. Le mot $hi \notin \mathcal{L}^*$ est un mot spécial représentant de façon unique tous les mots de longueur supérieure à n. Nous définissons un nouvel opérateur de concaténation :

$$u \cdot_n v = \begin{cases} u \cdot v \text{ si } |u \cdot v| \leq n \text{ et } u, v \neq hi \\ hi \text{ sinon} \end{cases} \tag{3.17}$$

Par soucis de lisibilité, nous écrivons $u \cdot v$ au lieu de $u \cdot_n v$ chaque fois qu'il sera clair, d'après le contexte, que u et v appartiennent à $\overline{\mathcal{L}^n}$. Le domaine des séries d'erreurs d'ordre au plus n est $\mathbb{R}^{\mathcal{L}^n} = \mathcal{F}(\mathbb{R}, \overline{\mathcal{L}^n})$. Les opérations élémentaires sur $\mathbb{R}^{\mathcal{L}^n}$ sont définies par les équations (3.4) à (3.9) de la section 3.3, dans lesquelles $\mathcal{W}$ représente maintenant le nouveau monoïde $\mathcal{W} = (\overline{\mathcal{L}^n}, \epsilon, \cdot_n)$.

Soit $\llbracket.\rrbracket^{\mathcal{L}^n}$ la sémantique définie en prenant le domaine $\mathbb{R}^{\mathcal{L}^n}$ pour les valeurs et par les règles de réduction de la section 3.2. $\llbracket.\rrbracket^{\mathcal{L}^n}$ détaille la contribution des erreurs des n premiers ordres à l'erreur globale sur le résultat d'un calcul.

Nous nous intéressons maintenant à la correction de $(\llbracket.\rrbracket^{\mathcal{L}^n})_{n\in\mathbb{N}}$. Rappelons tout d'abord que la correction de la sémantique la plus générale $\llbracket.\rrbracket^{\mathcal{L}^*}$ repose sur le lemme 5 et la proposition 6. Pour tout entier n, $\llbracket.\rrbracket^{\mathcal{L}^n}$ est une sémantique concrète, plus utilisable que $\llbracket.\rrbracket^{\mathcal{L}^*}$ en pratique, mais une preuve de correction similaire à celle de $\llbracket.\rrbracket^{\mathcal{L}^*}$ semble difficile à écrire, à cause du terme d'erreur d'ordre supérieur qui agglomère de nombreux termes d'erreurs. Plutôt que de reprendre l'approche de la section 3.3, nous allons montrer que $\llbracket.\rrbracket^{\mathcal{L}^n}$ est une approximation conservatrice de $\llbracket.\rrbracket^{\mathcal{L}^*}$ et que, pour tout $m \leq n$, $\llbracket.\rrbracket^{\mathcal{L}^m}$ est une approximation conservatrice de $\llbracket.\rrbracket^{\mathcal{L}^n}$.

Soient m et n deux entiers tels que $m \leq n$. Toutes nos sémantiques étant concrètes, nous allons comparer les sémantiques collectrices associées, c.à.d. que nous allons relier un ensemble d'exécutions de $\llbracket.\rrbracket^{\mathcal{L}^n}$ à un ensemble d'exécutions de $\llbracket.\rrbracket^{\mathcal{L}^m}$ au moyen de la correspondance de Galois suivante dans laquelle $\wp(X)$ représente l'ensemble des parties de X :

$$\langle \wp(\mathcal{F}(\mathbb{R}, \overline{\mathcal{L}^n})), \subseteq \rangle \quad \underset{\alpha^{n,m}}{\overset{\gamma^{m,n}}{\leftrightarrows}} \quad \langle \wp(\mathcal{F}(\mathbb{R}, \overline{\mathcal{L}^m})), \subseteq \rangle \tag{3.18}$$

$\alpha^{n,m}$ et $\gamma^{m,n}$ sont tout d'abord définies pour de simples valeurs, puis pour des ensembles de

valeurs.

$$\alpha^{n,m}\left(\sum_{u\in\overline{\mathcal{L}^n}}\omega^u\vec{\varepsilon}_u\right)\stackrel{\text{def}}{=}\sum_{u\in\overline{\mathcal{L}^m}\setminus\{hi\}}\omega^u\vec{\varepsilon}_u+\left(\sum_{u\in(\overline{\mathcal{L}^n}\setminus\overline{\mathcal{L}^m})\cup\{hi\}}\omega^u\right)\vec{\varepsilon}_{hi}$$

Pour un certain ensemble X,

$$\alpha^{n,m}(X)=\{\alpha^{n,m}(x)\ :\ x\in X\}$$

L'abstraction des coefficients attachés à $\vec{\varepsilon}_u$, pour tout $u\in\overline{\mathcal{L}^m}\setminus\{hi\}$, est naturelle. $\alpha^{n,m}$ accumule aussi les termes d'erreur d'ordre supérieur à m et attache le résultat à la variable formelle $\vec{\varepsilon}_{hi}$. Ensuite, la concrétisation est définie par :

$$\gamma^{m,n}\left(\sum_{u\in\overline{\mathcal{L}^m}}\nu^u\vec{\varepsilon}_u\right)\stackrel{\text{def}}{=}\left\{\sum_{u\in\overline{\mathcal{L}^n}}\omega^u\vec{\varepsilon}_u\ :\ \left|\begin{array}{l}\omega^u=\nu^u\text{ if }u\in\overline{\mathcal{L}^m}\setminus\{hi\}\\ \sum_{u\in(\overline{\mathcal{L}^n}\setminus\overline{\mathcal{L}^m})\cup\{hi\}}\omega^u=\nu^{hi}\end{array}\right.\right\}$$

et :

$$\gamma^{m,n}(X)=\cup_{x\in X}\gamma^{m,n}(x)$$

$\gamma^{m,n}$ associe à une série $\sum_{u\in\overline{\mathcal{L}^m}}\nu^u\vec{\varepsilon}_u\in\mathcal{F}(\mathbb{R},\overline{\mathcal{L}^m})$ un ensemble de séries de la forme $\sum_{u\in\overline{\mathcal{L}^n}}\omega^u\vec{\varepsilon}_u\in\mathcal{F}(\mathbb{R},\overline{\mathcal{L}^n})$ tel que $\omega^u=\nu^u$ pour tout $u\in\overline{\mathcal{L}^m}\setminus\{hi\}$ et tel que ν^{hi} est égal à la somme des termes restants.

Le lemme 7 prouve la correction des opérations élémentaires de $\mathcal{F}(\mathbb{R},\overline{\mathcal{L}^m})$ par rapport à la correction de ces mêmes opérations dans $\mathcal{F}(\mathbb{R},\overline{\mathcal{L}^n})$.

Lemme 7 *Soit ℓ un point de contrôle, soient $r^{\mathcal{L}^n},s^{\mathcal{L}^n}\in\mathcal{F}(\mathbb{R},\overline{\mathcal{L}^n})$ des séries d'erreurs, et soient $r^{\mathcal{L}^m}=\alpha^{n,m}(r^{\mathcal{L}^n})$, $s^{\mathcal{L}^m}=\alpha^{n,m}(s^{\mathcal{L}^n})$, $1\leq m\leq n$. Pour tout opérateur $\Diamond\in\{+,-,\times,\div\}$ nous avons :*

$$r^{\mathcal{L}^n}\Diamond^\ell s^{\mathcal{L}^n}\in\gamma^{m,n}(r^{\mathcal{L}^m}\Diamond^\ell s^{\mathcal{L}^m})$$

PREUVE. (Multiplication)

Soient :

$$r^{\mathcal{L}^n}=\sum_{u\in\overline{\mathcal{L}^n}}\omega_r^u\vec{\varepsilon}_u$$

et

$$s^{\mathcal{L}^n}=\sum_{u\in\overline{\mathcal{L}^n}}\omega_s^u\vec{\varepsilon}_u$$

Tout d'abord, d'après l'équation (3.6), nous avons :

$$\begin{array}{rcl}t^{\mathcal{L}^n}&=&r^{\mathcal{L}^n}\times^\ell s^{\mathcal{L}^n}\\&=&\uparrow_\circ(\omega_r^\epsilon\omega_s^\epsilon)\vec{\varepsilon}_\epsilon+\sum_{\substack{u,v\in\overline{\mathcal{L}^n}\\|u.v|>0}}\omega_r^u\omega_s^v\vec{\varepsilon}_{u.v}+\downarrow_\circ(\omega_r^\epsilon\omega_s^\epsilon)\vec{\varepsilon}_\ell\end{array}\tag{3.19}$$

De façon similaire, si $r^{\mathcal{L}^m}=\sum_{u\in\overline{\mathcal{L}^m}}\nu_r^u\vec{\varepsilon}_u$ et si $s^{\mathcal{L}^m}=\sum_{u\in\overline{\mathcal{L}^m}}\nu_s^u\vec{\varepsilon}_u$ alors :

$$\begin{array}{rcl}t^{\mathcal{L}^m}&=&r^{\mathcal{L}^m}\times^\ell s^{\mathcal{L}^m}\\&=&\uparrow_\circ(\nu_r^\epsilon\nu_s^\epsilon)\vec{\varepsilon}_\epsilon+\sum_{\substack{u,v\in\overline{\mathcal{L}^m}\\|u.v|>0}}\nu_r^u\nu_s^v\vec{\varepsilon}_{u.v}+\downarrow_\circ(\nu_r^\epsilon\nu_s^\epsilon)\vec{\varepsilon}_\ell\end{array}\tag{3.20}$$

Par définition de $\gamma^{m,n}$,

$$\gamma^{m,n}(t^{\mathcal{L}^m}) = \left\{ \sum_{u \in \overline{\mathcal{L}^n}} \omega^u \vec{\varepsilon}_u \ : \ \left| \begin{array}{l} \omega^u = \nu_t^u \text{ if } u \in \overline{\mathcal{L}^m} \setminus \{hi\} \\ \sum_{u \in (\overline{\mathcal{L}^n} \setminus \overline{\mathcal{L}^m}) \cup \{hi\}} \omega^u = \nu_t^{hi} \end{array} \right. \right\} \tag{3.21}$$

où, dans (3.21),

$$\nu_t^u = \sum_{u_1 u_2 = u} \nu_r^{u_1} \nu_s^{u_2}$$

et

$$\nu_t^{hi} = \sum_{u_1 u_2 = u \in (\overline{\mathcal{L}^n} \setminus \overline{\mathcal{L}^m}) \cup \{hi\}} \nu_r^{u_1} \nu_s^{u_2}$$

Nous devons montrer que $\sum_{u \in (\overline{\mathcal{L}^n} \setminus \overline{\mathcal{L}^m}) \cup \{hi\}} \omega_t^u = \nu_t^{hi}$. Nous utilisons les notations $M = \overline{\mathcal{L}^m} \setminus \{hi\}$ et $N = (\overline{\mathcal{L}^n} \setminus \overline{\mathcal{L}^m}) \cup \{hi\}$. Nous avons :

$$\begin{aligned} \sum_{u \in N} \omega_t^u &= \sum_{\substack{u \in \overline{\mathcal{L}^n}, v \in \overline{\mathcal{L}^n} \\ u.v \in N}} \omega_r^u \omega_s^v \\ &= \sum_{\substack{u, v \in M \\ u.v \in N}} \omega_r^u \omega_s^v + \sum_{u, v \in N} \omega_r^u \omega_s^v \\ &+ \sum_{\substack{u \in M, v \in N \\ u.v \in N}} \omega_r^u \omega_s^v + \sum_{\substack{u \in N, v \in M \\ u.v \in N}} \omega_r^u \omega_s^v \end{aligned}$$

Puisque $r^{\mathcal{L}^m} = \alpha^{n,m}(r^{\mathcal{L}^n})$ et $s^{\mathcal{L}^m} = \alpha^{n,m}(s^{\mathcal{L}^n})$, $\sum_{u \in N} \omega_r^u = \nu_r^{hi}$ et $\sum_{u \in N} \omega_s^u = \nu_s^{hi}$. Par conséquent :

$$\sum_{u \in N} \omega_t^u = \sum_{\substack{u, v \in M \\ u.v \in N}} \omega_r^u \omega_s^v + \nu_r^{hi} \nu_s^{hi} + \sum_{u \in M} \omega_r^u \nu_s^{hi} + \sum_{v \in M} \nu_r^{hi} \omega_s^v$$

Puisque $r^{\mathcal{L}^m} = \alpha^{n,m}(r^{\mathcal{L}^n})$ et $s^{\mathcal{L}^m} = \alpha^{n,m}(s^{\mathcal{L}^n})$, pour tout mot u tel que $|u| \leq m$, nous avons $\omega_r^u = \nu_r^u$ et $\omega_s^u = \nu_s^u$. Ainsi,

$$\begin{aligned} \sum_{u \in N} \omega_t^u &= \sum_{\substack{u, v \in M \\ u.v \in N}} \nu_r^u \nu_s^v + \nu_r^{hi} \nu_s^{hi} + \sum_{u \in M} \nu_r^u \nu_s^{hi} + \sum_{v \in M} \nu_r^{hi} \nu_s^v \\ &= \sum_{\substack{u, v \in \overline{\mathcal{L}^m} \\ u.v \in N}} \nu_r^u \nu_s^v = \nu_t^{hi} \end{aligned}$$

□

Remarquons que le lemme 7 et sa preuve incluent le cas $n = \infty$ (où $[\![.]\!]^{\mathcal{L}^*}$ remplace $[\![.]\!]^{\mathcal{L}^\infty}$). Afin d'étendre le lemme 7 à des séquences de pas de réduction, on introduit la fonction $\mathcal{R}$ définie par l'équation (3.22). $\text{Lab}(a_0^{\ell_0})$ est l'ensemble des étiquettes apparaissant dans $a_0^{\ell_0}$.

$$\mathcal{R}(a_0^{\ell_0}) \ : \left| \begin{array}{ccc} \text{Lab}(a_0^{\ell_0}) & \rightarrow & \wp\left(\mathcal{F}(\mathbb{R}, \overline{\mathcal{L}^n})\right) \\ \ell & \mapsto & \left\{ \begin{array}{l} \{r\} \text{ si } a^\ell = r^\ell \\ \emptyset \text{ sinon} \end{array} \right. \end{array} \right. \tag{3.22}$$

$\mathcal{R}(a_0^{\ell_0})(\ell)$ calcule un singleton contenant une valeur si la sous-expression a^ℓ dans $a_0^{\ell_0}$ est une valeur. $\mathcal{R}(a_0^{\ell_0})(\ell)$ calcule l'ensemble vide sinon.

Proposition 8 *Soient $a_0^{\ell_0 \mathcal{L}^m}$ et $a_0^{\ell_0 \mathcal{L}^n}$ des expressions arithmétiques syntaxiquement équivalentes telles que pour tout $\ell \in \mathcal{L}$, nous avons :*

$$\mathcal{R}(a_0^{\ell_0 \mathcal{L}^n})(\ell) \subseteq \gamma^{m,n}(\mathcal{R}(a_0^{\ell_0 \mathcal{L}^m})(\ell))$$

Si $a_0^{\ell_0 \mathcal{L}^m} \rightarrow a_1^{\ell_1 \mathcal{L}^m}$ alors $a_0^{\ell_0 \mathcal{L}^n} \rightarrow a_1^{\ell_1 \mathcal{L}^n}$ tel que $a_1^{\ell_1 \mathcal{L}^m}$ et $a_1^{\ell_1 \mathcal{L}^n}$ sont des expressions syntaxiquement équivalentes et tel que pour tout $\ell \in \mathcal{L}$, $\mathcal{R}(a_1^{\ell_1 \mathcal{L}^n})(\ell) \subseteq \gamma^{m,n}(\mathcal{R}(a_1^{\ell_1 \mathcal{L}^m})(\ell))$.

Etant donnée une expression $a_0^{\ell_0}$, la proposition 8 indique comment $[\![.]\!]^{\mathcal{L}^n}(a_0^{\ell_0})$ et $[\![.]\!]^{\mathcal{L}^{n+1}}(a_0^{\ell_0})$ sont en correspondance, pour tout entier $n \geq 0$. La preuve de cette propriété est une simple induction sur la structure de l'expression a_0.

La sémantique $[\![.]\!]^{\mathcal{L}^n}$ repose sur le domaine $\mathbb{R}^{\mathcal{L}^n} = \mathcal{F}(\mathbb{R}, \overline{\mathcal{L}^n})$. La sémantique $[\![.]\!]^{\mathcal{L}^*}$ peut être vue comme la limite à l'infini de ce modèle. Symétriquement, la sémantique $[\![.]\!]^{\mathcal{L}^0}$ utilise des séries de la forme $\omega^\epsilon \vec{\varepsilon}_\epsilon + \omega^{hi} \vec{\varepsilon}_{hi}$ et calcule l'erreur globale : elle correspond à la sémantique $[\![.]\!]_{\mathbb{E}}$ définie au chapitre 2. Plus généralement, il existe une chaîne de correspondances de Galois entre les sémantiques aux différents ordres :

$$[\![.]\!]^{\mathcal{L}^*}(a_0^{\ell_0}) \ \leftrightarrows \ \ldots \ [\![.]\!]^{\mathcal{L}^n}(a_0^{\ell_0}) \ \leftrightarrows \ [\![.]\!]^{\mathcal{L}^{n-1}}(a_0^{\ell_0}) \ \ldots \ \leftrightarrows \ [\![.]\!]^{\mathcal{L}^0}(a_0^{\ell_0})$$

$[\![.]\!]^{\mathcal{L}^*}(a_0^{\ell_0})$ est la sémantique procurant le plus d'informations : elle donne la contribution de toutes les erreurs élémentaires de tous ordres. A l'opposé, $[\![.]\!]^{\mathcal{L}^0}(a_0^{\ell_0})$ est la sémantique procurant le moins d'informations : elle donne uniquement l'erreur globale sur le résultat d'un calcul.

3.5 Sémantiques à grain variable

Nous présentons ici une nouvelle famille de sémantiques qui généralisent celles des sections 3.3 et 3.4. Intuitivement, nous ne nous intéressons plus aux erreurs dues à des points de contrôle isolés, ou, autrement dit, à des opérations élémentaires. Au lieu de cela, nous considérons les erreurs introduites par des morceaux de code de tailles variables et formant une partition du programme. On pourra ainsi s'intéresser, par exemple, à la contribution à l'erreur globale de l'erreur introduite par une formule intermédiaire, une fonction, ou une ligne de code.

En pratique, cette nouvelle famille de sémantiques est importante pour réduire l'espace mémoire et le temps de calcul nécessaires à l'analyse. D'un point de vue théorique, il est nécessaire de prouver la correction de ces nouvelles sémantiques par rapport à la sémantique générale $[\![.]\!]^{\mathcal{L}^*}$ présentée à la section 3.3. Pour cela, nous montrons que les différentes partitions des points de contrôle peuvent être partiellement ordonnées de façon à ce que les sémantiques reposant sur des partitions comparables sont elles-même comparables.

Soit $\mathcal{J} = \{J_1, J_2, \ldots, J_p\} \in \mathcal{P}(\mathcal{L})$ une partition des points de contrôle. Nous utilisons maintenant des mots sur l'alphabet $\mathcal{J}$. $\mathcal{J}^n$ représente les mots de longueur maximale n et $\overline{\mathcal{J}^n} = (\mathcal{J}^n/\sim) \cup \{hi\}$. La concaténation $\cdot_n$ utilisée pour former les mots de $\overline{\mathcal{J}^n}$ est définie par l'équation (3.17).

Pour un ordre maximal d'erreur $n \in \mathbb{N}$, on considère la famille $(\mathcal{F}(\mathbb{R}, \overline{\mathcal{J}^n}))_{\mathcal{J}\in\mathcal{P}(\mathcal{L})}$ de domaines, notée de façon équivalente $(\mathbb{R}^{\mathcal{J}^n})_{\mathcal{J}\in\mathcal{P}(\mathcal{L})}$. Intuitivement, une étiquette unique est utilisée pour identifier toutes les opérations appartenant à une même classe, dans la partition choisie. Une valeur $r^{\mathcal{J}^n} \in \mathbb{R}^{\mathcal{J}^n}$ est notée :

$$r^{\mathcal{J}^n} = f\vec{\varepsilon} + \sum_{\substack{u \in \overline{\mathcal{J}^{n+}} \\ u = J_1 \ldots J_k}} \left(\sum_{\substack{v = \ell_1 \ldots \ell_k \in \overline{\mathcal{L}^n} \\ \forall i,\ 1 \leq i \leq k,\ \ell \in J_i}} \omega^v \right) \vec{\varepsilon}_u = f\vec{\varepsilon} + \sum_{u \in \overline{\mathcal{J}^{n+}}} \omega^u \vec{\varepsilon}_u$$

Si $|u| = 1$, le mot $u = J$ identifie l'erreur du premier ordre due aux opérations appartenant à la classe de J. Les opérations élémentaires entre valeurs de $\mathbb{R}^{\mathcal{J}^n}$ sont définies par les équations (3.4) à (3.9) de la section 3.3, dans lesquelles $\mathcal{W}$ est le nouveau monoïde $(\overline{\mathcal{J}^n}, \epsilon, \cdot_n)$.

Remarquons que ces nouvelles sémantiques généralisent celles de la section 3.4 qui reposent sur une partition particulière : $\mathcal{J} = \{\{\ell\} \ : \ \ell \in \mathcal{L}\}$. Une autre partition intéressante est obtenue en utilisant des singletons pour les points de contrôle correspondant aux données initiales et en regroupant tous les autres points de contrôle : on peut ainsi calculer la contribution à l'erreur globale des erreurs dues aux données, ou, autrement dit, réaliser une analyse de sensibilité (voir le chapitre 2).

Dans la suite de cette section, nous allons montrer comment des sémantiques fondées sur des partitions comparables sont comparables. Cela nous fournira aussi une preuve de correction pour cette nouvelle famille de sémantiques. Intuitivement, une partition $\mathcal{J}_2$ des points de contrôle est plus grossière qu'une partition $\mathcal{J}_1$ si $\mathcal{J}_2$ agglomère certaines classes de $\mathcal{J}_1$. Pour un ordre maximal d'erreur n et en utilisant cet ordre partiel sur les partitions, la partition $\mathcal{J} = \{\{\ell\} \ : \ \ell \in \mathcal{L}\}$ correspond à celle utilisée par la sémantique $[\![.]\!]^{\mathcal{L}^n}$ et est la partition la plus précise d'ordre n. Nous montrons que toute sémantique utilisant une partition $\mathcal{J}_2$, plus grossière qu'une partition $\mathcal{J}_1$, est une approximation de la sémantique utilisant $\mathcal{J}_1$. Par conséquent, une sémantique utilisant une partition quelconque des points de contrôle est une approximation de la sémantique la plus générale à l'ordre n, c.à.d. $[\![.]\!]^{\mathcal{L}^n}$, ainsi qu'une approximation de la sémantique générale $[\![.]\!]^{\mathcal{L}^*}$. L'ordre partiel sur l'ensemble des partitions des points de contrôle d'un programme est formellement défini par :

Définition 9 *Soit $\mathcal{J}_1$ et $\mathcal{J}_2$ deux partitions de l'ensemble $\mathcal{L}$. $\mathcal{J}_2$ est une partition de $\mathcal{L}$ plus grossière que $\mathcal{J}_1$, noté $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$, si et seulement si $\forall J_1 \in \mathcal{J}_1,\ \exists J_2 \in \mathcal{J}_2 \ : \ J_1 \subseteq J_2$.*

Si $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$ alors certaines classes de la partition $\mathcal{J}_1$ ont été agglomérées dans $\mathcal{J}_2$. $\dot{\subseteq}$ est utilisé pour ordonner partiellement les partitions de l'ensemble $\mathcal{L}$ des étiquettes.

La fonction de traduction $\tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}$ définie ci-dessous transforme les mots du langage $\overline{\mathcal{J}_1^n}$ en des mots du langage $\overline{\mathcal{J}_2^n}$.

$$\left\{ \begin{array}{rcl} \tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}(\epsilon) & = & \epsilon \\ \tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}(J_1.u) & = & J_2.\tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}(u) \text{ où } J_1 \in \mathcal{J}_1,\ J_1 \subseteq J_2,\ J_2 \in \mathcal{J}_2 \end{array} \right. \tag{3.23}$$

La correction de la sémantique utilisant une partition $\mathcal{J}_2$ de $\mathcal{L}$ repose sur le fait que, pour tout $\mathcal{J}_2 \in \mathcal{P}(\mathcal{L})$ tel que $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$, il existe une correspondance de Galois :

$$\langle \wp(\mathcal{F}(\mathbb{R}, \overline{\mathcal{J}_1^n})), \subseteq \rangle \quad \underset{\alpha^{\mathcal{J}_1^n, \mathcal{J}_2^n}}{\overset{\gamma^{\mathcal{J}_2^n, \mathcal{J}_1^n}}{\leftrightarrows}} \quad \langle \wp(\mathcal{F}(\mathbb{R}, \overline{\mathcal{J}_2^n})), \subseteq \rangle$$

définie par :

$$\alpha^{\mathcal{J}_1^n, \mathcal{J}_2^n}\left(\sum_{u \in \overline{\mathcal{J}_1^n}} \omega^u \vec{\varepsilon}_u\right) \stackrel{\text{def}}{=} \sum_{u \in \overline{\mathcal{J}_1^n}} \omega^u \vec{\varepsilon}_{\tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}(u)}$$

$$\gamma^{\mathcal{J}_2^n, \mathcal{J}_1^n}\left(\sum_{v \in \overline{\mathcal{J}_2^n}} \nu^u \vec{\varepsilon}_v\right) \stackrel{\text{def}}{=} \left\{ \sum_{u \in \overline{\mathcal{J}_1^n}} \omega^u \vec{\varepsilon}_u \ : \ \sum_{\tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}(u) = v} \omega^u = \nu^v \right\}$$

A nouveau, pour des ensembles X et Y,

$$\alpha^{\mathcal{J}_1^n, \mathcal{J}_2^n}(X) = \{\alpha^{\mathcal{J}_1^n, \mathcal{J}_2^n}(x) \ : \ x \in X\}$$

et

$$\gamma^{\mathcal{J}_2^n, \mathcal{J}_1^n}(Y) = \cup_{y \in Y} \gamma^{\mathcal{J}_2^n, \mathcal{J}_1^n}(y)$$

Soit J un élément de la partition la plus grossière $\mathcal{J}_2$. Pour tout terme d'erreur du premier ordre $\omega^u \vec{\varepsilon}_u$ de la série d'erreurs $r^{\overline{\mathcal{J}_1^n}} = \sum_{u \in \overline{\mathcal{J}_1^n}} \omega^u \vec{\varepsilon}_u$, l'abstraction $\alpha^{\mathcal{J}_1^n, \mathcal{J}_2^n}(r^{\overline{\mathcal{J}_1^n}})$ définit le coefficient ν_J attaché à $\vec{\varepsilon}_J$ comme étant la somme des coefficients $\omega_{J'}$ tels que $J' \in \mathcal{J}_1$ et $J' \subseteq J$. La fonction $\gamma^{\mathcal{J}_2^n, \mathcal{J}_1^n}$ calcule l'ensemble des séries d'erreurs telles que $\sum_{\tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}(u) = v} \omega^u = \nu^v$. Le lemme 10 permet d'établir la correction des opérations élémentaires définies par les équations (3.4) à (3.9) pour les domaines présentés dans cette section.

Lemme 10 *Soit ℓ un point de contrôle, soient $\mathcal{J}_1$ et $\mathcal{J}_2$ des partitions de $\mathcal{L}$ telles que $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$ et soient $r^{\mathcal{J}_1^n}, s^{\mathcal{J}_1^n} \in \mathcal{F}(\mathbb{R}, \overline{\mathcal{J}_1^n})$. Si $r^{\mathcal{J}_2^n} = \alpha^{\mathcal{J}_1^n, \mathcal{J}_2^n}(r^{\mathcal{J}_1^n})$, $s^{\mathcal{J}_2^n} = \alpha^{\mathcal{J}_1^n, \mathcal{J}_2^n}(s^{\mathcal{J}_1^n})$ alors, pour tout $\Diamond \in \{+, -, \times, \div\}$, nous avons :*

$$r^{\mathcal{J}_1^n} \Diamond^\ell s^{\mathcal{J}_1^n} \in \gamma^{\mathcal{J}_2^n, \mathcal{J}_1^n}(r^{\mathcal{J}_2^n} \Diamond^\ell s^{\mathcal{J}_2^n})$$

PREUVE. (Multiplication)
On utilise les notations :

$$r^{\mathcal{J}_1^n} = \sum_{u \in \overline{\mathcal{J}_1^n}} \omega_r^u \vec{\varepsilon}_u, \qquad s^{\mathcal{J}_1^n} = \sum_{u \in \overline{\mathcal{J}_1^n}} \omega_s^u \vec{\varepsilon}_u,$$

$$r^{\mathcal{J}_2^n} = \sum_{u \in \overline{\mathcal{J}_2^n}} \nu_r^u \vec{\varepsilon}_u, \qquad s^{\mathcal{J}_2^n} = \sum_{u \in \overline{\mathcal{J}_2^n}} \nu_s^u \vec{\varepsilon}_u.$$

Soit $\tau(u) = \tau_{\mathcal{J}_1^n, \mathcal{J}_2^n}(u)$. Nous avons :

$$t^{\mathcal{J}_1^n} = r^{\mathcal{J}_1^n} \times^\ell s^{\mathcal{J}_1^n} = \uparrow_\circ (\omega_r^\epsilon \omega_s^\epsilon) \vec{\varepsilon}_\epsilon + \sum_{\substack{u, v \in \overline{\mathcal{J}_1^n} \\ |u.v| > 0}} \omega_r^u \omega_s^v \vec{\varepsilon}_{u.v} + \downarrow_\circ (\omega_r^\epsilon \omega_s^\epsilon) \vec{\varepsilon}_\ell$$

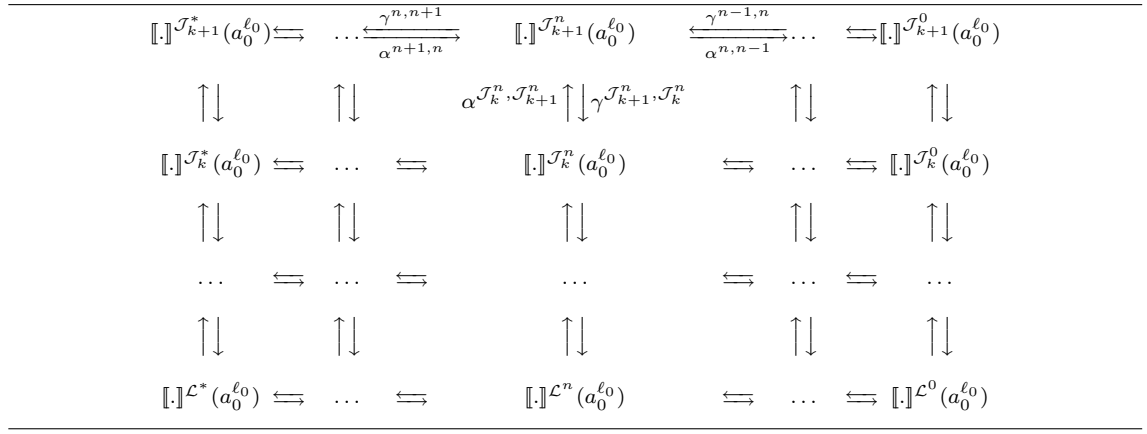

FIG. 3.2 – *Correspondances entre les différentes sémantiques $[\![.]\!]^{\mathcal{J}_k^n}$ pour un ordre maximal d'erreur n et pour une chaîne de partitions $\{\{\ell\} : \ell \in \mathcal{L}\} \dot{\subseteq} \mathcal{J}_1 \dot{\subseteq} \ldots \dot{\subseteq} \mathcal{J}_k \dot{\subseteq} \ldots \dot{\subseteq} \{\mathcal{L}\}$.*

$$t^{\mathcal{J}_2^n} = r^{\mathcal{J}_2^n} \times^{\ell} s^{\mathcal{J}_2^n} = \uparrow_{\circ} (\nu_r^{\epsilon}\nu_s^{\epsilon})\vec{\varepsilon}_{\epsilon} + \sum_{\substack{u,v \in \overline{\mathcal{J}_2^n} \\ |u.v|>0}} \nu_r^u \nu_s^v \vec{\varepsilon}_{u.v} + \downarrow_{\circ} (\nu_r^{\epsilon}\nu_s^{\epsilon})\vec{\varepsilon}_{\ell}$$

Le point clef de cette preuve consiste à montrer que pour tout $u \in \overline{\mathcal{J}_2^n}$, $\sum_{\tau(v)=u} \omega_t^v = \nu_t^u$.

$$\begin{aligned}
\sum_{\tau(v)=u} \omega_t^v &= \sum_{\tau(v_1.v_2)=u} \omega_r^{v_1}\omega_s^{v_2} = \sum_{\substack{\tau(v_1).\tau(v_2)=u_1.u_2 \\ u_1.u_2=u}} \omega_r^{v_1}\omega_s^{v_2} \\
&= \sum_{u_1.u_2=u} \left(\sum_{\substack{\tau(v_1)=u_1 \\ \tau(v_2)=u_2}} \omega_r^{v_1}\omega_s^{v_2} \right) \\
&= \sum_{u_1.u_2=u} \left(\sum_{\tau(v_1)=u_1} \omega_r^{v_1} \times \sum_{\tau(v_2)=u_2} \omega_s^{v_2} \right) = \sum_{u_1.u_2=u} \nu_r^{u_1}\nu_s^{u_2} = \nu_t^u
\end{aligned}$$

□

La sémantique définie par le domaine $\mathbb{R}^{\mathcal{J}^n}$ pour les valeurs et par les règles de réduction de la section 3.2 est notée $[\![.]\!]^{\mathcal{J}^n}$. La proposition 11 montre comment les sémantiques $[\![.]\!]^{\mathcal{J}_1^n}$ et $[\![.]\!]^{\mathcal{J}_2^n}$ sont reliées, pour des partitions comparables $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$ de l'ensemble $\mathcal{L}$ des étiquettes.

Proposition 11 *Soient $\mathcal{J}_1$ et $\mathcal{J}_2$ des partitions de $\mathcal{L}$ telles que $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$ et soient $a_0^{\ell_0\mathcal{J}_1^n}$ et $a_0^{\ell_0\mathcal{J}_2^n}$ des expressions arithmétiques syntaxiquement équivalentes telles que pour tout $\ell \in \mathcal{L}$, nous avons :*

$$\mathcal{R}(a_0^{\ell_0\mathcal{J}_2^n})(\ell) \subseteq \gamma^{\mathcal{J}_1^n,\mathcal{J}_2^n}(\mathcal{R}(a_0^{\ell_0\mathcal{J}_1^n})(\ell))$$

Si $a_0^{\ell_0\mathcal{J}_1^n} \rightarrow a_1^{\ell_1\mathcal{J}_1^n}$ alors $a_0^{\ell_0\mathcal{J}_2^n} \rightarrow a_1^{\ell_1\mathcal{J}_2^n}$ tel que $a_1^{\ell_1\mathcal{J}_1^n}$ et $a_1^{\ell_1\mathcal{J}_2^n}$ sont des expressions syntaxiquement équivalentes et tel que pour tout $\ell \in \mathcal{L}$, $\mathcal{R}(a_1^{\ell_1\mathcal{J}_2^n})(\ell) \subseteq \gamma^{\mathcal{J}_1^n,\mathcal{J}_2^n}(\mathcal{R}(a_1^{\ell_1\mathcal{J}_2^n})(\ell))$.

La preuve est une induction sur la structure de l'expression a_0. Par ailleurs, cette proposition implique que, pour un ordre d'erreur maximal donné n et pour une chaîne de partitions C, il existe une correspondance de Galois entre les sémantiques utilisant C. Supposons en effet que :

$$C = \big(\big(\{\{\ell\} \ : \ell \in \mathcal{L}\} \ = \mathcal{J}_0\big) \dot{\subseteq} \ \ldots \ \dot{\subseteq} \ \ldots \ \dot{\subseteq} \ \mathcal{J}_k \ \dot{\subseteq} \ \ldots \ \dot{\subseteq} \ \{\mathcal{L}\}\big)$$

Alors,

$$[\![.]\!]^{\mathcal{L}^n}(a_0^{\ell_0}) \ \leftrightarrows \ \ldots \ [\![.]\!]^{\mathcal{J}_k^n}(a_0^{\ell_0}) \ \leftrightarrows \ \ldots \ \leftrightarrows \ [\![.]\!]^{\{\mathcal{L}\}^n}(a_0^{\ell_0})$$

En combinant les propositions 8 et 11, nous pouvons aussi relier les sémantiques $[\![.]\!]^{\mathcal{J}_k^n}$ et $[\![.]\!]^{\mathcal{J}_k^{n+1}}$ pour tout $\mathcal{J}_k \in C$ et pour tout $n \in \mathbb{N}$. Ceci est résumé par la figure 3.2. Pour tout entier n et pour toute partition $\mathcal{J}_k$, $[\![.]\!]^{\mathcal{J}_k^n}$ définit une sémantique particulière :

- $[\![.]\!]^{\mathcal{L}^*}$ est la sémantique procurant le plus d'informations ;
- à l'opposé, la sémantique $[\![.]\!]^{\mathcal{L}^0}$ calcule l'erreur globale et correspond à la sémantique $[\![.]\!]_{\mathbb{E}}$ du chapitre 2. Il s'agit de la sémantique procurant le moins d'informations ;
- pour tout $k > 0$, $[\![.]\!]^{\mathcal{J}_k^0} = [\![.]\!]^{\mathcal{L}^0}$ (pour $n = 0$, toutes les partitions donnent la même sémantique) ;
- $[\![.]\!]^{\mathcal{J}_0^2}$ calcule globalement les erreurs des deux premiers ordres ;
- pour tout $\mathcal{J}_k$, $[\![.]\!]^{\mathcal{J}_k^1}$ calcule la contribution à l'erreur globale des erreurs du premier ordre introduites par différents morceaux de code identifiés par $\mathcal{J}_k$.

Remarquons que les séries de $\mathbb{R}^{\mathcal{J}_2^n}$ contiennent moins de termes que celles de $\mathbb{R}^{\mathcal{J}_1^n}$, si $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$. L'utilisation d'un grain plus grossier permet de réduire sensiblement l'espace mémoire et le temps de calcul nécessaires à l'analyse d'un programme.

3.6 Perspectives

Toutes les sémantiques présentées dans ce chapitre demeurent théoriques : elles ne sont pas directement implémentables puisqu'elles comportent des termes d'erreur à valeurs dans les réels, et, par conséquent, non-représentables en machine. Leur utilisation comme base formelle pour la définition d'analyses statiques ouvre la voie à de très nombreux travaux concernant, entre autres, le choix dynamique (en cours d'analyse) d'une partition particulière des points de contrôle, les analyses relationnelles ou l'analyse de systèmes hybrides. Nous reviendrons sur ces prolongements au cours du chapitre suivant.

Par ailleurs, les techniques envisagées au chapitre 2 pour calculer, à l'aide de sémantiques dynamiques, les termes d'erreur de la sémantique $[\![.]\!]_{\mathbb{E}}$ (notamment les méthodes $[\![.]\!]_{\mathbb{EI}}$, $[\![.]\!]_{\mathbb{ES}}$ et $[\![.]\!]_{\mathbb{ED}}$) sont généralisables aux séries d'erreurs. En particulier, nous pouvons remarquer que les dérivées partielles produites par la différentiation automatique permettent d'approximer linéairement certain termes du premier ordre des séries d'erreurs (ceux correspondant aux données, les erreurs d'arrondies dues aux opérations arithmétiques étant elles négligées).

Finalement, les séries d'erreurs permettent de comprendre partiellement les raisons pour lesquelles le résultat d'un calcul est imprécis, en indiquant quelles erreurs d'arrondi initiales ont été amplifiées au cours des traitements. D'autres informations complémentaires permettraient de mieux comprendre les phénomènes de pertes de précision. Il serait par exemple intéressant de savoir, de façon duale, quelles opérations amplifient principalement les erreurs initiales, indépendamment de ces dernières. Une autre sémantique, a priori difficile à définir mais qui pourrait être d'un grand intérêt consisterait à utiliser des séries d'erreurs relatives.

Chapitre 4

Séries d'erreurs et analyse statique

4.1 Introduction

La sémantique des séries d'erreurs présentée au chapitre précédent constitue la base théorique de nos travaux dans le domaine de la précision numérique. Elle suscite néanmoins de nombreuses nouvelles interrogations auxquelles nous allons partiellement répondre au cours de ce chapitre. NP-complétude du partitionnement dynamique des points de contrôle, analyse de stabilité fondée sur le calcul d'exposants de Lyapunov ou encore extension aux systèmes hybrides discrets-continus, de nombreux thèmes méritent d'être encore approfondis. Les réponses partielles apportées dans les pages suivantes ne font que souligner l'ampleur des travaux qu'il reste à mener dans ce domaine.

4.2 Partitionnement dynamique des points de contrôle

Comme le montre la figure 3.2, de nombreuses sémantiques peuvent être utilisées pour calculer plus ou moins précisément la contribution à l'erreur globale des différentes pertes de précision survenant au cours d'un calcul. D'un point de vue pratique, le nombre de termes des séries est un paramètre important, qui influence directement le temps de calcul et l'espace mémoire nécessaires à l'analyse d'un programme [PGM04, Mar06].

Dans cette section, nous montrons comment le choix d'une sémantique particulière $[\![.]\!]^{\mathcal{J}^n}$ affecte la précision et le temps de calcul d'une analyse. Nous nous penchons ensuite sur le problème du changement dynamique de sémantique, c.à.d. le problème du passage d'une sémantique $[\![.]\!]^{\mathcal{J}_1^n}$ à une autre $[\![.]\!]^{\mathcal{J}_2^m}$ en cours d'analyse, pour des questions de performances et de précision du résultat. Il s'agit là d'un problème de partitionnement dynamique des points de contrôle utilisés comme indices pour les termes des séries d'erreurs [Bou92, Jea02, Mar06].

Tout d'abord, appelons $\langle \mathcal{I}_{\mathbb{F}}, \subseteq \rangle$ le domaine des intervalles dont les bornes sont des nombres flottants et considérons l'abstraction :

$$\langle \wp(\mathcal{F}(\mathbb{R}, \overline{\mathcal{J}^n})), \subseteq \rangle \quad \underset{\alpha^{\mathcal{I}}}{\overset{\gamma^{\mathcal{I}}}{\leftrightarrows}} \quad \langle \mathcal{F}(\mathcal{I}_{\mathbb{F}}, \overline{\mathcal{J}^n}), \sqsubseteq \rangle$$

Dans $\mathcal{F}(\mathcal{I}_{\mathbb{F}}, \overline{\mathcal{J}^n})$, les séries d'erreurs abstraites sont ordonnées terme à terme, c.à.d. que :

$$\sum_{u \in \overline{\mathcal{J}^n}} \omega_1^u \vec{\varepsilon}_u \sqsubseteq \sum_{u \in \overline{\mathcal{J}^n}} \omega_2^u \vec{\varepsilon}_u \iff \forall u \in \overline{\mathcal{J}^n},\ \omega_1^u \subseteq \omega_2^u \tag{4.1}$$

Soit X un ensemble de nombres flottants et soit $\Phi : \wp(\mathbb{F}) \to \mathcal{I}_{\mathbb{F}}$ la fonction calculant l'enveloppe convexe $\Phi(X)$ de X, ou, autrement dit, le plus petit intervalle de $\mathcal{I}_{\mathbb{F}}$ contenant X. Pour un ensemble de séries d'erreurs $(\sum_{u\in\overline{\mathcal{J}^n}} \omega_i^u \vec{\varepsilon}_u)_{i\in I}$ indicées par $i \in I$, $\alpha^{\mathcal{I}}$ et $\gamma^{\mathcal{I}}$ sont définis par :

$$\alpha^{\mathcal{I}}\left(\left\{\sum_{u\in\overline{\mathcal{J}^n}} \omega_i^u \vec{\varepsilon}_u \ : \ i \in I\right\}\right) \stackrel{\text{def}}{=} \sum_{u\in\overline{\mathcal{J}^n}} \Phi\left(\{\omega_i^u \ : \ i \in I\}\right) \vec{\varepsilon}_u \tag{4.2}$$

et

$$\gamma^{\mathcal{I}}\left(\sum_{u\in\overline{\mathcal{J}^n}} \nu^u \vec{\varepsilon}_u\right) \stackrel{\text{def}}{=} \left\{\sum_{u\in\overline{\mathcal{J}^n}} \omega^u \vec{\varepsilon}_u \ : \ \forall u \in \overline{\mathcal{J}^n},\ \omega^u \in \nu^u\right\} \tag{4.3}$$

Nous allons maintenant montrer comment le choix d'une partition des points de contrôle influence l'analyse statique d'un programme. Soient $m \leq n$, soient $\mathcal{J}_1$ et $\mathcal{J}_2$ deux partitions telles que $\mathcal{J}_1 \dot{\subseteq} \mathcal{J}_2$ et soient $r^{\mathcal{J}_1^n} \in \mathcal{F}(\mathcal{I}_{\mathbb{F}}, \overline{\mathcal{J}_1^n})$, $s^{\mathcal{J}_1^n} \in \mathcal{F}(\mathcal{I}_{\mathbb{F}}, \overline{\mathcal{J}_1^n})$, $r^{\mathcal{J}_2^m} \in \mathcal{F}(\mathcal{I}_{\mathbb{F}}, \overline{\mathcal{J}_2^m})$, et $s^{\mathcal{J}_2^m} \in \mathcal{F}(\mathcal{I}_{\mathbb{F}}, \overline{\mathcal{J}_2^m})$ tels que :

$$r^{\mathcal{J}_2^m} = \alpha^{\mathcal{I}} \circ \alpha^{\mathcal{J}_1^n, \mathcal{J}_2^m} \circ \gamma^{\mathcal{I}}(r^{\mathcal{J}_1^n})$$

et

$$s^{\mathcal{J}_2^m} = \alpha^{\mathcal{I}} \circ \alpha^{\mathcal{J}_1^n, \mathcal{J}_2^m} \circ \gamma^{\mathcal{I}}(s^{\mathcal{J}_1^n})$$

En pratique, il arrive souvent que $r^{\mathcal{J}_2^m} \sqsubseteq s^{\mathcal{J}_2^m}$ alors que $r^{\mathcal{J}_1^n} \not\sqsubseteq s^{\mathcal{J}_1^n}$ et $s^{\mathcal{J}_1^n} \not\sqsubseteq r^{\mathcal{J}_1^n}$. Autrement dit, deux valeurs peuvent être comparables dans la sémantique de plus gros grain tandis qu'elles sont incomparables dans celle de plus petit grain. Ce constat est important pour l'analyse statique des boucles, lorsque la condition d'arrêt dépend d'une comparaison entre deux valeurs. L'analyse peut terminer au bout de moins d'itérations dans $[\![.]\!]^{\overline{\mathcal{J}_2^m}}$ que dans $[\![.]\!]^{\overline{\mathcal{J}_1^n}}$. Par exemple, considérons la boucle :

```
|1| while !(x<=y) {
|2|   y = x;
|3|   x = x+a;
|4|   x = b*x;
|5| }
```

On suppose que, initialement :

$$a = [0.0, 0.14] + [0.0, 0.002]\vec{\varepsilon}_a \quad b = [0.5, 0.7]$$

$$x = [0.0, 0.66] + [0.0, 0.006]\vec{\varepsilon}_x \quad y = [0.0, 0.0]$$

Par soucis de clarté, nous allons utiliser pour cet exemple un ensemble simplifié de nombres flottants dont la mantisse est formée de deux chiffres décimaux. On utilise une partition associant un point de contrôle par ligne de code. Soient $\alpha = [0.0, 0.002]$ et $\beta = [0.0, 0.006]$. Les valeurs successives de `x` aux lignes |3| et |4| sont :

$$\begin{array}{lrcl}
|3| & x & = & [0.0, 0.80]\vec{\varepsilon} + \alpha\vec{\varepsilon}_a + \beta\vec{\varepsilon}_x \\
|4| & x & = & [0.0, 0.56]\vec{\varepsilon} + [0.5, 0.7]\alpha\vec{\varepsilon}_a + [0.5, 0.7]\beta\vec{\varepsilon}_x \\
 & & = & [0.0, 0.56]\vec{\varepsilon} + [0.0, 0.0014]\vec{\varepsilon}_a + [0.0, 0.0042]\vec{\varepsilon}_x \\
|3| & x & = & [0.0, 0.70]\vec{\varepsilon} + (\alpha + 0.7\alpha)\vec{\varepsilon}_a + 0.7\beta\vec{\varepsilon}_x \\
|4| & x & = & [0.0, 0.49]\vec{\varepsilon} + (0.7\alpha + 0.7^2\alpha)\vec{\varepsilon}_a + 0.7^2\beta\vec{\varepsilon}_x \\
 & & = & [0.0, 0.49]\vec{\varepsilon} + [0.0, 0.00238]\vec{\varepsilon}_a + [0.0, 0.00294]\vec{\varepsilon}_x
\end{array}$$

Soient x_1 et x_2 les valeurs de x après une et deux itérations. Nous avons :

$$x_1 = [0.0, 0.56]\vec{\varepsilon} + [0.0, 0.0014]\vec{\varepsilon}_a + [0.0, 0.0042]\vec{\varepsilon}_x \tag{4.4}$$

$$x_2 = [0.0, 0.49]\vec{\varepsilon} + [0.0, 0.00238]\vec{\varepsilon}_a + [0.0, 0.00294]\vec{\varepsilon}_x \tag{4.5}$$

Dans notre sémantique, $x_2 \not\sqsubseteq x_1$ à cause du coefficient de $\vec{\varepsilon}_a$ et quelques itérations supplémentaires sont nécessaires pour faire converger l'analyse. De plus, du fait que $x_2 \not\sqsubseteq x_1$, un analyseur statique peut réaliser un élargissement qui générerait une grande perte de précision. Dans ce cas, nous pourrions obtenir :

$$x = x_1 \nabla x_2 = [0.0, 0.56]\vec{\varepsilon} + [0.0, +\infty]\vec{\varepsilon}_a + [0.0, 0.0042]\vec{\varepsilon}_x$$

Cependant, dans la sémantique à plus gros grain utilisant une partition qui agglomère les termes d'erreurs $\vec{\varepsilon}_a$ et $\vec{\varepsilon}_x$, nous avons :

$$x'_1 = [0.0, 0.56]\vec{\varepsilon} + [0.0, 0.0056]\vec{\varepsilon}_{a,x} \tag{4.6}$$

$$x'_2 = [0.0, 0.49]\vec{\varepsilon} + [0.0, 0.00532]\vec{\varepsilon}_{a,x} \tag{4.7}$$

et $x'_2 \sqsubseteq x'_1$. Cet exemple illustre comment la convergence d'une analyse peut dépendre du choix d'une partition particulière. D'un côté, nous souhaitons utiliser une partition fine des points de contrôle, afin de connaître précisément les sources d'erreurs, tandis que d'un autre, il est souhaitable de regrouper des termes d'erreurs pour accélérer la convergence de l'analyse et éviter les élargissements.

Etant données deux séries x_1 and x_2 indicées par un ensemble $\mathcal{J}$ de points de contrôle, comme dans les équations (4.4) et (4.5), nous souhaitons trouver la plus grande partition $\mathcal{J}'$ de $\mathcal{J}$ permettant d'affirmer que $\alpha^{\mathcal{J},\mathcal{J}'}(x_1) \sqsubseteq \alpha^{\mathcal{J},\mathcal{J}'}(x_2)$.

Pour simplifier, nous nous concentrons sur les bornes supérieures des intervalles utilisés comme coefficients des séries, le problème des bornes inférieures étant dual. Le problème du partitionnement dynamique peut alors être formellement défini comme suit :

Définition 12 (Partitionnement dynamique (DP)) *Etant donnés deux n-uplets d'entiers positifs ou nuls $W = \langle \omega_1, \ldots, \omega_n \rangle$ et $W' = \langle \omega'_1, \ldots, \omega'_n \rangle$, existe-t-il une partition $\mathcal{P}$ de $\{1, \ldots, n\}$, de taille t, telle que*

$$\forall X \in \mathcal{P}, \sum_{i \in X} \omega_i \leq \sum_{i \in X} \omega'_i \tag{4.8}$$

Sans perte de généralité, DP est défini pour des coefficients entiers.

Proposition 13 *DP est NP-complet.*

La preuve est une réduction à partir du problème Partition, connu pour être lui-même NP-complet [GJ79] et dont la définition est rappelée ci-dessous.

Définition 14 (Partition) *Etant donné un ensemble d'entiers positifs $A = \{a_1, \ldots, a_n\}$, existe-t-il un sous-ensemble I de A tel que :*

$$\sum_{a_i \in I} a_i = \sum_{a_i \in A \setminus I} a_i$$

PREUVE. (Proposition 13)

DP appartient bien à la classe NP, il est en effet trivial de vérifier si une instance donnée est effectivement une solution au problème en temps polynomial. Soit $A = \{a_1, \ldots, a_n\}$ une instance quelconque I_1 de Partition. On construit, en temps polynomial, à partir de I_1, une instance I_2 de DP définie par les équations (4.9) à (4.13). Comme dans la définition 12, t représente le nombre de classes de la partition.

$$t = 2 \tag{4.9}$$

$$\omega_i = \left(\sum_{a_j \in A} a_j\right) + a_i,\ 1 \leq i \leq n \tag{4.10}$$

$$\omega'_i = \left(\sum_{a_j \in A} a_j\right) - a_i,\ 1 \leq i \leq n \tag{4.11}$$

$$\omega_{n+1} = \omega_{n+2} = 0 \tag{4.12}$$

$$\omega'_{n+1} = \omega'_{n+2} = \sum_{a_j \in A} a_j \tag{4.13}$$

Supposons qu'il existe un algorithme polynomial pour résoudre DP. Cet algorithme permet de calculer une solution à I_2 qui vérifie :

$$\forall X \in \mathcal{P},\ \sum_{i \in X} \omega_i \leq \sum_{i \in X} \omega'_i \tag{4.14}$$

Tout d'abord, remarquons que puisque $t = 2$, $\mathcal{P} = \{X, \overline{X}\}$, avec $\overline{X} = \{1, 2, \ldots, n+2\} \setminus X$. De plus, $n+1$ et $n+2$ ne peuvent pas appartenir à la même classe. En effet, puisque $\omega'_i < \omega_i$, pour $1 \leq i \leq n$, si $n+1$ et $n+2$ appartiennent à X (resp. $\overline{X}$), alors l'équation (4.14) ne serait pas satisfaite par $\overline{X}$ (resp. X). Intéressons-nous à l'ensemble X pour lequel nous avons :

$$\begin{aligned}
\sum_X \omega_i &\leq \sum_X \omega'_i \\
|X| \sum_A a_i + \sum_X a_i &\leq |X| \sum_A a_i - \sum_X a_i + \omega'_{n+1} \\
\sum_X a_i &\leq \omega'_{n+1} - \sum_X a_i \\
\sum_X a_i &\leq \sum_A a_i - \sum_X a_i = \sum_{\overline{X}} a_i
\end{aligned}$$

Concernant $\overline{X}$, en utilisant la même technique de preuve on obtient :

$$\sum_{\overline{X}} a_i \leq \sum_X a_i$$

On en déduit l'égalité des deux quantités. Nous avons trouvé une partition $\mathcal{P}$ de A qui est une solution à Partition. On en déduit que DP est NP-complet.

□

Bien que DP soit NP-complet dans le cas général, certaines heuristiques fonctionnent correctement en pratique. Par exemple, on peut initialement utiliser une sémantique qui associe différents

points de contrôle aux i premières instances des opérateurs présents dans une boucle. Ensuite, pour les comparaisons, on peut regrouper tous les termes d'erreurs correspondant à une même opération. Cette heuristique donne de bons résultats en pratique, lorsque les calculs effectués dans les boucles sont stables. Des développements supplémentaires afin de définir des heuristiques plus performantes mériteraient d'être menés.

4.3 Stabilité des calculs effectués dans des boucles

Dans cette section, nous présentons une analyse relationnelle permettant de déterminer si les calculs effectués dans un corps de boucle sont stables [Mar02b]. Cette analyse utilise la sémantique des séries d'erreurs et un critère de stabilité pour les applications non-linéaires fondé sur le calcul d'exposants de Lyapunov abstraits. Les exposants de Lyapunov sont couramment employés pour étudier la stabilité des systèmes dynamiques [ASY96, Moo92]. Intuitivement, ce critère consiste à déterminer si la fonction calculée dans un corps de boucle est contractante ou expansive. Dans notre approche, ce test est non seulement réalisé pour les fonctions correspondant aux calculs en nombres flottants, mais aussi pour les fonctions correspondant à la propagation des erreurs d'arrondi, telles qu'elles sont définies par la sémantique des séries d'erreurs.

4.3.1 Stabilité d'une boucle

Soit $f : \mathbb{R}^m \to \mathbb{R}^m$ et soit $\vec{x}_0 \in \mathbb{R}^m$. Nous souhaitons étudier le comportement de la trajectoire de $\vec{x}_0$ par f, c.à.d. de la suite $(\vec{x}_n)_n$ des itérées définies par :

$$\vec{x}_n = f(\vec{x}_{n-1}) = f^n(\vec{x}_0) \tag{4.15}$$

Nous souhaitons déterminer si les erreurs initiales sur $\vec{x}_0$ ainsi que les erreurs d'arrondi survenant au cours du calcul de $(\vec{x}_n)_n$ augmentent ou décroissent à chaque itération. Par exemple, considérons les itérées de la fonction $g : x \mapsto x^2$ avec x_0 voisin de 1. Clairement, la suite $(g^n(x_0))_n$ converge vers 0 si $x_0 < 1$, est constante si $x = 1$ et diverge si $x > 1$. De plus, pour deux valeurs initiales voisines $x_0 > 1$ et $x'_0 > 1$, $|g^n(x_0) - g^n(x'_0)| \to \infty$ quand $n \to \infty$. Dans ce dernier cas, dans un programme calculant $(g^n(x_0))_n$, une erreur initiale minime sur x_0 peut considérablement modifier le résultat final. La fonction g est dite instable au voisinage de 1.

De nombreux travaux ont été menés sur la théorie des systèmes de fonctions itérées (iterated function systems ou IFS) [ASY96, Bar93, Bea91, Moo92]. Pour les fonctions à plusieurs dimensions, l'espace est généralement divisé en deux parties, l'ensemble de Fatou et l'ensemble de Julia contenant respectivement les x_0 pour lesquels les trajectoires $(g^n(x_0))_n$ sont stables et instables [Bea91]. Pour de nombreuses fonctions, la frontière entre ces deux zones est fractale (voir la figure 4.1).

Nous ne souhaitons pas étudier directement la stabilité d'une application f correspondant au corps d'une boucle. Au lieu de cela, nous allons nous intéresser à la stabilité des équations sémantiques générées à partir du corps de boucle par la sémantique des séries d'erreurs. Ainsi, nous déterminons non seulement la stabilité des calculs en nombres flottants mais aussi celle des erreurs d'arrondi survenant en cours de calcul. Considérons à nouveau la fonction $g : x \mapsto x^2$ et la valeur initiale, donnée sous forme de série d'erreurs $x_0^\ell = 0.95\vec{\varepsilon} + 0.1\vec{\varepsilon}_\ell$. Si l'on ne s'intéresse qu'à la composante flottante, $\hat{x}_0 = 0.95$, $(g^n(\hat{x}_0))_n \to 0$ et nous concluons que cette séquence est stable dans le voisinage immédiat de 0.95. Cependant, dans les réels nous aurions $\hat{x}_0 = 1.05$

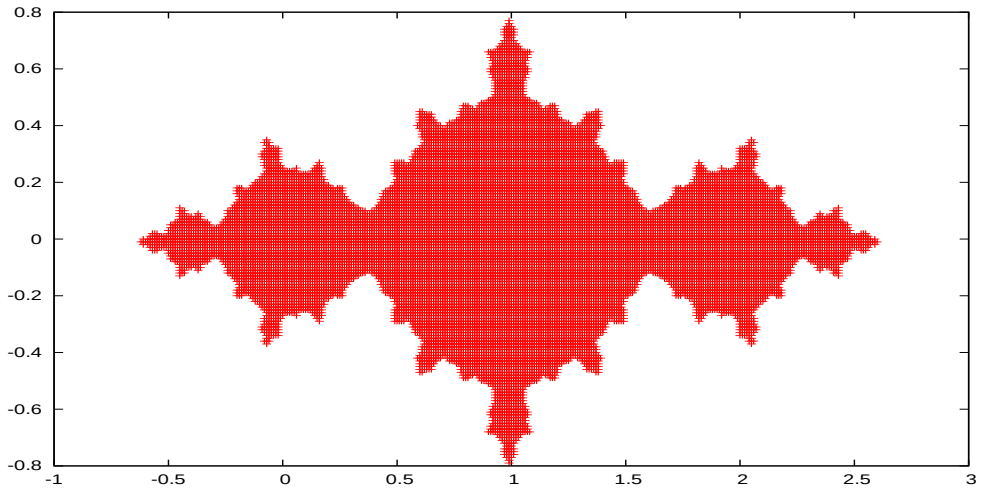

FIG. 4.1 – *Ensemble de Fatou pour les itérées de la fonction* $f : \mathbb{C} \to \mathbb{C}$, $z \mapsto z^2 - 1$.

et $(g^n(\hat{x}_0))_n \to \infty$. Aussi, la meilleure information que l'on puisse donner à un programmeur est que, si $x_0^\ell = 0.95\vec{\varepsilon} + 0.1\vec{\varepsilon}_\ell$ alors les itérées de g sont stables mais les erreurs de calcul sont, elles, instables.

La façon dont est générée la fonction f correspondant aux équations sémantiques à partir de la syntaxe d'un programme est décrite à la section 4.3.2. Pour l'instant, nous supposons que f est donnée et nous nous concentrons sur l'étude de sa stabilité. Dans le cas particulier où $f : \mathbb{R}^m \to \mathbb{R}^m$ est une application linéaire, on peut étudier sa stabilité à partir de ses valeurs propres. Si l'on suppose que la matrice M de f a des valeurs propres distinctes et réelles, alors M est similaire à une matrice diagonale :

$$N = \begin{pmatrix} x_1 & & 0 \\ & \ddots & \\ 0 & & x_m \end{pmatrix} \tag{4.16}$$

et la n-ième itérée f^n peut être calculée à partir de $N^n = \begin{pmatrix} x_1^n & & 0 \\ & \ddots & \\ 0 & & x_m^n \end{pmatrix}$. Ainsi, les itérées de f sont stables si $0 \leq |x_i| < 1$ pour tout $1 \leq i \leq m$. Cette dernière propriété est toujours valable si f possède des valeurs propres multiples ou complexes. Dans le cas linéaire, il est ainsi possible d'utiliser des algorithmes pour calculer des sur-approximations des valeurs propres des applications correspondant aux corps de boucles et d'en déduire leur stabilité. Cette approche a été suggérée par E. Goubault [Gou01].

Cependant, de nombreux programmes effectuent des calculs non-linéaires et des non-linéarités supplémentaires sont introduites dans f par les fonctions modélisant la propagation des erreurs

d'arrondi. Par exemple, pour la fonction g utilisée précédemment dans cette section, nous avons :

$$\hat{g} : \begin{pmatrix} \hat{x} \\ e_x \\ e_{xh} \end{pmatrix} \mapsto \begin{pmatrix} \hat{x}^2 \\ 2xe_x \\ e_x^2 + e_{xh}^2 + 2\hat{x}e_{xh} + 2e_x e_{xh} \end{pmatrix} \tag{4.17}$$

$\hat{g}$ est une version simplifiée de la fonction générée pour l'application $g : x \mapsto x^2$. Le nombre réel x est décomposé en trois termes, $\hat{x}$ pour le nombre flottant, e_x et e_{xh} pour l'erreur du premier ordre et pour l'erreur d'ordre supérieur (on utilise ici la sémantique des séries d'erreurs à l'ordre un). $\hat{g}$ est obtenue à partir de g en posant $x = \hat{x} + e_x + e_{xh}$ et en regroupant les termes de façon cohérente dans le développement. Dans cet exemple simplifié, certains termes d'erreurs sont négligés (par exemple l'erreur due à la multiplication). En prenant pour valeurs initiales $\hat{x} = 0.95$, $e_x = 0.1$ et $e_{xh} = 0$, on peut aisément vérifier que les itérées de $\hat{g}$ tendent vers $(0, 0, \infty)$. Il s'agit là d'un exemple dans lequel les erreurs d'ordre supérieur ne sont pas négligeables alors que celles du premier ordre le sont.

Pour pouvoir analyser des calculs non-linéaires, nous ne pourrons donc pas utiliser une simple matrice, telle que N dans l'équation (4.16). Le rôle des valeurs propres sera joué par les exposants de Lyapunov qui permettent de mesurer le taux de séparation des itérées, pour une trajectoire partant d'un point $x_0 \in \mathbb{R}^m$. Cette notion est expliquée intuitivement dans le paragraphe suivant.

Soient $f : \mathbb{R} \to \mathbb{R}$ et $x_0 \in \mathbb{R}$ tels que $f(x_0) = x_0$. x_0 est un point fixe attractif (resp. répulsif) de f si la dérivée f' est telle que $f'(x_0) < 1$ (resp. $f'(x_0) > 1$). Si x_0 est répulsif, une trajectoire de f partant d'un point dans le voisinage de x_0 s'éloignera de la trajectoire partant exactement de x_0 d'un facteur de $|f'(x_0)|$ par itération environ. Sinon, si le point fixe est attractif, la seconde trajectoire tendra vers la première. Maintenant, si x_0 est un point périodique de f, c.à.d. si $x_0 \neq f(x_0) \neq \ldots \neq f^k(x_0) = x_0$, alors x_0 est un point périodique attractif ou répulsif, selon la valeur de $a = |f'(x_0)| \times |f'(x_1)| \times \ldots \times |f'(x_k)|$ par rapport à 1. A nouveau, si x_0 est répulsif alors toute trajectoire partant d'un voisinage de x_0 s'éloignera à chaque itération de l'orbite périodique d'un facteur moyen de $a^{\frac{1}{k}}$ par itération. Enfin, pour un point x_0 non périodique, le taux moyen de séparation par itérée est :

$$a = \lim_{k \to \infty} (|f'(x_0)| \times |f'(x_1)| \times \ldots \times |f'(x_k)|)^{\frac{1}{k}}$$

L'exposant de Lyapunov de la trajectoire est défini comme étant le logarithme naturel de la quantité précédente, c.à.d. :

$$\lambda = \lim_{k \to \infty} \frac{\ln|f'(x_0)| + \ldots + \ln|f'(x_k)|}{k} \tag{4.18}$$

Pour une fonction $f : \mathbb{R}^m \to \mathbb{R}^m$, les valeurs propres de la matrice jacobienne Df sont substituées à celle de la dérivée première dans le raisonnement précédent et nous obtenons m exposants de Lyapunov.

Définition 15 *Soit $f : \mathbb{R}^m \to \mathbb{R}^m$ et soit e_i^n, $1 \leq i \leq m$ la i-ième plus grande valeur propre de $Df^n(x_0)$. Le i-ième exposant de Lyapunov est défini par :*

$$\lambda_i = \lim_{n \to \infty} \frac{\ln e_i^n}{n}$$

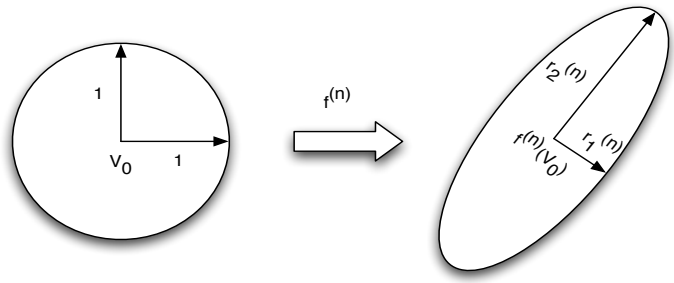

FIG. 4.2 – *Image de la sphère unitaire après n itérations de la fonction f.*

f est *stable* au voisinage de x_0 si le plus grand exposant de Lyapunov est un nombre négatif. Sinon, les coefficients $\lambda_i > 0$ correspondent aux directions instables. $\lambda_i = 0$ signifie qu'il y a une divergence faible (sous-exponentielle) dans la i-ième direction. Dans $\mathbb{R}^m$, calculer les exposants de Lyapunov de $f : \mathbb{R}^m \to \mathbb{R}^m$ revient aussi à calculer les images par f de la sphère unitaire U de centre $\vec{x_0} \in \mathbb{R}^m$ [ASY96]. Comme cela est illustré par la figure 4.2, on obtient une ellipsoïde E de dimension m. La longueur des axes de E donne des informations sur la stabilité de f. En effet, les images de U peuvent évoluer vers plusieurs formes :

- les images de U tendent vers un simple point p. f est contractante dans toutes les directions. p est un point fixe attractif de f, aussi appelé un puits ;
- les images de U sont strictement croissantes dans toutes les directions. f est expansive dans toutes les directions et le point fixe correspondant est une source ;
- U devient une ellipsoïde fine et allongée. f est contractante dans certaines directions et expansive dans d'autres. Le point fixe correspondant est une selle.

Ainsi, les exposants de Lyapunov calculent le taux de contraction ou d'expansion par f d'un voisinage d'un point initial x_0. Cette vision nous amène à donner une définition alternative, équivalente à la définition 15.

Définition 15 (définition alternative) *Soit $f : \mathbb{R}^m \to \mathbb{R}^m$ et soit r_i^n, $1 \leq i \leq m$, la longueur du i-ième plus long axe orthogonal de l'ellipsoïde $Df^n(x_0)U$. Le i-ième exposant de Lyapunov est défini de façon équivalente à la définition 15 par :*

$$\lambda_i = \lim_{n \to \infty} \frac{\ln r_i^n}{n}$$

Nous avons vu que les exposants de Lyapunov pouvaient être définis de deux façons équivalentes, au moyen des valeurs propres de la matrice jacobienne de l'application, ou à partir des images par f d'une sphère unitaire. Ces deux définitions mènent à deux algorithmes, présentés dans [Mar02b] pour étudier la stabilité des calculs effectués dans un corps de boucle. Le plus simple de ces algorithmes est présenté à la section 4.3.3.

4.3.2 Génération des équations sémantiques

Comme nous l'avons vu à la section 4.3.1, l'étude de la stabilité d'une fonction f modélisant la propagation des erreurs dans un calcul itératif repose sur le calcul d'exposants de Lyapunov.

La définition exacte de f provient de la sémantique des séries d'erreurs et nous utiliserons ici la sémantique à l'ordre un. Dans cette section, notre but est de construire la matrice jacobienne Df de f, directement à partir de la syntaxe du programme. Le calcul des exposants, à partir de Df est présenté à la section 4.3.3.

Soient $r_1^{\ell_1} = f_1\vec{\varepsilon} + \sum_{\ell\in\overline{\mathcal{L}^1}} \omega_1^\ell\vec{\varepsilon}_\ell$ et $r_2^{\ell_2} = f_2\vec{\varepsilon} + \sum_{\ell\in\overline{\mathcal{L}^1}} \omega_2^\ell\vec{\varepsilon}_\ell$ deux séries d'erreurs définies aux points de contrôle ℓ_1 et ℓ_2. On s'intéresse à l'opération $r_1^{\ell_1} \Diamond^{\ell_3} r_2^{\ell_2}$. Cette opération calcule une nouvelle valeur $r_3^{\ell_3} = f_3\vec{\varepsilon} + \sum_{\ell\in\overline{\mathcal{L}^1}} \omega_3^\ell\vec{\varepsilon}_\ell$, au point ℓ_3, et peut être vue comme une fonction des variables $f_1, \omega_1^{\ell_1}, \ldots, \omega_1^{\ell_k}, \omega_1^{hi}, f_2, \omega_2^{\ell_2}, \ldots, \omega_2^{\ell_k}, \omega_2^{hi}$ et $f_3, \omega_3^{\ell_1}, \ldots, \omega_3^{\ell_k}, \omega_3^{hi}$, où $\overline{\mathcal{L}^1}$ est l'ensemble des mots de contrôle, de cardinal k. Par exemple, la fonction f_+ correspondant à une addition $r_1^{\ell_1} +^{\ell_3} r_2^{\ell_2}$ transforme le vecteur :

$$\vec{v} = (f_1, \omega_1^{\ell_1}, \ldots, \omega_1^{\ell_k}, \omega_1^{hi}, f_2, \omega_2^{\ell_1}, \ldots, \omega_2^{\ell_k}, \omega_2^{hi}, f_3, \omega_3^{\ell_1}, \ldots, \omega_3^{\ell_k}, \omega_3^{hi})$$

en :

$$\begin{array}{rcl} f_+(\vec{v}) & = & \big(\quad f_1, \quad \omega_1^{\ell_1} \quad ,\ldots, \quad \omega_1^{\ell_3} \quad ,\ldots, \quad \omega_1^{hi}, \\ & & \quad\ f_2, \quad \omega_2^{\ell_1} \quad ,\ldots, \quad \omega_2^{\ell_3} \quad ,\ldots, \quad \omega_2^{hi}, \\ & & \uparrow_\circ (f_1+f_2), \quad \omega_1^{\ell_1}+\omega_2^{\ell_1} \quad ,\ldots, \quad \omega_1^{\ell_3}+\omega_2^{\ell_3}+\downarrow_\circ (f_1+f_2) \quad ,\ldots, \quad \omega_1^{hi}+\omega_2^{hi}\big) \end{array}$$

Les termes relatifs aux opérandes demeurent inchangés dans le résultat, tandis que les coefficients relatifs à r_3 sont modifiés selon la sémantique de l'addition. La matrice jacobienne de f_+, notée B^+ est directement obtenue à partir de f_+. La dérivée du terme $\downarrow_\circ (f_1+f_2)$ par rapport à f_1 ou f_2 est égale à 1 ou -1 et nous prendrons, dans les matrices d'intervalles de la section 4.3.3, la valeur $[-1,1]$ pour ce terme. Par exemple nous avons :

$$B^+ = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ B^+_{\ell_3,\ell_1} & B^+_{\ell_3,\ell_2} & 0 \end{pmatrix} \tag{4.19}$$

avec :

$$B^+_{\ell_3,\ell_1} = \begin{array}{c} \\ f_3 \\ \omega_3^{\ell_1} \\ \vdots \\ \omega_3^{\ell_3} \\ \vdots \\ \omega_3^{\ell_n} \\ \omega_3^{\ell_{hi}} \end{array} \begin{array}{c} \begin{array}{ccccccc} \frac{\partial}{\partial f_1} & \frac{\partial}{\partial w_1^{\ell_1}} & \cdots & \frac{\partial}{\partial w_1^{\ell}} & \cdots & \frac{\partial}{\partial w_1^{\ell_k}} & \frac{\partial}{\partial w_1^{hi}} \end{array} \\ \begin{pmatrix} 1 & 0 & \ldots & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ddots & & & \vdots & 0 \\ 0 & 0 & \ddots & \ddots & & \vdots & \vdots \\ 1 & \vdots & \ddots & 1 & \ddots & \vdots & 0 \\ 0 & \vdots & & \ddots & \ddots & 0 & \vdots \\ 0 & \vdots & & & \ddots & 1 & 0 \\ 0 & 0 & \ldots & 0 & \ldots & 0 & 1 \end{pmatrix} \end{array} \tag{4.20}$$

$B^+_{\ell_3,\ell_2}$ est définie de façon analogue et I représente un bloc identité. La matrice jacobienne $B^\times$ pour la multiplication est obtenue de façon similaire à partir de la fonction :

$$\begin{array}{rcl} f_\times(\vec{v}) & = & \big(\quad f_1, \quad \omega_1^{\ell_1} \quad ,\ldots, \quad \omega_1^{\ell_3} \quad ,\ldots, \quad \omega_1^{hi}, \\ & & \quad\ f_2, \quad \omega_2^{\ell_1} \quad ,\ldots, \quad \omega_2^{\ell_3} \quad ,\ldots, \quad \omega_2^{hi}, \\ & & \uparrow_\circ (f_1\times f_2), \quad f_1\omega_2^{\ell_1}+f_2\omega_1^{\ell_1} \quad ,\ldots, \quad f_1\omega_2^{\ell_3}+f_2\omega_1^{\ell_3}+\downarrow_\circ (f_1\times f_2) \quad ,\ldots, \quad \omega_3^{hi}\big) \end{array}$$

dans laquelle $\omega_3^{hi} = f_1\omega_2^{hi} + f_2\omega_1^{hi} + \sum_{\ell,\ell'\in\overline{\mathcal{L}^1}^+} \omega_1^\ell\omega_2^{\ell'}$.

La matrice jacobienne $B^{\times}$ est construite comme B^{+}. Par exemple, on obtient le bloc :

$$B^{\times}_{\ell_3,\ell_1} = \begin{array}{c} \\ f_3 \\ \omega_3^{\ell_1} \\ \vdots \\ ; \ \omega_3^{\ell_3} \\ \vdots \\ \omega_3^{\ell_n} \\ \omega_3^{\ell_{hi}} \end{array} \begin{array}{c} \begin{array}{ccccccc} \frac{\partial}{\partial f_1} & \frac{\partial}{\partial w_1^{\ell_1}} & \cdots & \frac{\partial}{\partial w_1^{\ell}} & \cdots & \frac{\partial}{\partial w_1^{\ell_k}} & \frac{\partial}{\partial w_1^{\ell_{hi}}} \end{array} \\ \begin{pmatrix} f_2 & 0 & \dots & \dots & \dots & 0 & 0 \\ \omega_2^{\ell_1} & f_2 & \ddots & & & \vdots & 0 \\ \vdots & 0 & \ddots & \ddots & & \vdots & \vdots \\ U & \vdots & \ddots & f_2 & \ddots & \vdots & 0 \\ \vdots & \vdots & & \ddots & \ddots & 0 & \vdots \\ \omega_2^{\ell_n} & 0 & \dots & \dots & 0 & f_2 & 0 \\ \omega_2^{\ell_{hi}} & S & \dots & S & \dots & S & S' \end{pmatrix} \end{array} \tag{4.21}$$

avec :

$$U = \omega_2^{\ell_2} + f_2 \qquad S = \sum_{\ell \in \overline{\mathcal{L}^1}^{+}} \omega_2^{\ell} \qquad S' = S + f_2$$

Etant donné un morceau de code p correspondant à un corps de boucle et le langage $\overline{\mathcal{L}^1}$ construit sur les étiquettes de p, la matrice jacobienne M de la fonction sémantique correspondant à p est construite bloc par bloc. Par exemple, l'opération $r_1^{\ell_1} \times^{\ell_3} r_2^{\ell_2}$ nous amène à remplir les blocs correspondant aux dérivées partielles des variables de r_3 par les variables de r_1 et r_2. En bref, si $\frac{\partial r^{\ell_3}}{\partial r^{\ell_1}}$ et $\frac{\partial r^{\ell_3}}{\partial r^{\ell_2}}$ sont des blocs de la matrice globale, la multiplication nous amène à remplir M de la manière suivante :

$$M = \begin{array}{c} \\ \vdots \\ r^{\ell_1} \\ \vdots \\ ; \ r^{\ell_2} \\ \vdots \\ r^{\ell_3} \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \cdots & \frac{\partial}{\partial r_{\ell_1}^{\rightarrow}} & \cdots & \frac{\partial}{\partial r_{\ell_2}^{\rightarrow}} & \cdots & \frac{\partial}{\partial r_{\ell_3}^{\rightarrow}} & \cdots \end{array} \\ \begin{pmatrix} . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & B^{\times}_{\ell_3,\ell_1} & . & B^{\times}_{\ell_3,\ell_2} & . & . & . \\ . & . & . & . & . & . & . \end{pmatrix} \end{array} \tag{4.22}$$

Formellement, soit σ : Id $\rightarrow \overline{\mathcal{L}^1}$ un environnement $\sigma \in$ Env associant à chaque variable une étiquette et soit $M \in$ Mat une matrice jacobienne. Pour une expression e^{ℓ}, la valeur $\mathcal{E}(e^{\ell}, M, \sigma)$ est affectée à M par la fonction $\mathcal{E}$: Expr $\times$ Mat $\times$ Env $\rightarrow$ Mat définie par :

$$\mathcal{E}(c^{\ell}, M, \sigma) = M$$

$$\mathcal{E}(x^{\ell}, M, \sigma) = M + I_{\ell,\sigma(x)}$$

$$\mathcal{E}\big((e_1^{\ell_1} \Diamond^{\ell} \varepsilon_2^{\ell_2}), M, \sigma\big) = \mathcal{E}\big(e_2^{\ell_2}, \mathcal{E}(e_1^{\ell_1}, M, \sigma), \sigma\big) + B^{\Diamond}_{\ell,\ell_1} + B^{\Diamond}_{\ell,\ell_2}$$

Une constante c ne modifie pas M. I_{ℓ_1,ℓ_2} représente le bloc identité inséré à la position $\frac{\partial r^{\ell_1}}{\partial r^{\ell_2}}$. Si $\sigma(x) = \ell_2$ alors une occurrence de x^{ℓ_1} nous conduit à insérer le bloc I_{ℓ_1,ℓ_2} dans la matrice M. Pour une opération $\Diamond$, M est modifiée comme décrit dans l'équation (4.22). Pour un morceau de code p correspondant au corps d'une boucle, la matrice M est générée par la fonction $\mathcal{P}$: Prg $\times$ Mat $\times$ Env $\rightarrow$ Mat $\times$ Env définie ci-dessous.

$$\mathcal{P}(x^{\ell_1} = e^{\ell_2}, M, \sigma) = \big(\mathcal{E}(e^{\ell_2}, M, \sigma) + I_{\ell_1,\ell_2}\ ,\ \sigma[x \mapsto \ell_2]\big)$$

$$\mathcal{P}\ (p_1; p_2, M, \sigma) = \big(\lambda(M', \sigma').\mathcal{P}(p_2, M', \sigma')\big)\ \mathcal{P}(p_1, M, \sigma)$$

La matrice complète pour un corps de boucle p est générée par $\mathcal{P}(p, O, [])$ où O représente la matrice nulle et [] l'environnement vide. Par exemple, la matrice obtenue pour le programme donné ci-dessous est :

```
while (x! = 0){
    p : x^ℓ4 = x^ℓ1 ×^ℓ3 x^ℓ2 ;
}
```

$$\mathcal{P}(p, O, []) = \ ;\ \begin{array}{c} \\ r^{\ell_1} \\ r^{\ell_2} \\ r^{\ell_3} \\ r^{\ell_4} \end{array} \begin{array}{c} \begin{array}{cccc} \frac{\partial}{\partial r\vec{\ell}_1} & \frac{\partial}{\partial r\vec{\ell}_2} & \frac{\partial}{\partial r\vec{\ell}_3} & \frac{\partial}{\partial r\vec{\ell}_4} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & I \\ 0 & 0 & 0 & I \\ B^{\times}_{\ell_3,\ell_1} & B^{\times}_{\ell_3,\ell_2} & 0 & 0 \\ 0 & 0 & I & 0 \end{pmatrix} \end{array}$$

4.3.3 Calcul abstrait des exposants de Lyapunov

Dans cette section, nous présentons un algorithme fondé sur la première définition des exposants de Lyapunov (définition 15) et permettant de vérifier globalement la stabilité de l'ensemble des termes des séries d'erreurs. Un second algorithme, plus complexe et capable de détecter les directions instables, a aussi été présenté dans [Mar02b].

Nous allons utiliser les matrices jacobiennes générées selon la méthode de la section 4.3.2. Notre analyse est une interprétation abstraite de la sémantique des séries d'erreurs dans laquelle les coefficients scalaires sont approchés par des intervalles, comme à la section 4.2. Par conséquent, elle permet de prouver la stabilité d'un ensemble de systèmes dynamiques, pour un ensemble de conditions initiales.

Comme nous l'avons vu à la section 4.3.1, les exposants de Lyapunov permettent d'étudier la stabilité d'une trajectoire d'une fonction itérée $f\ :\ \mathbb{R}^m \to \mathbb{R}^m$ à partir d'un point initial $x_0 \in \mathbb{R}^m$. D'après la définition 15, les exposants sont calculés à partir des valeurs propres de la matrice jacobienne $Df^n(x_0)$ de la n-ième itérée de f. Rappelons qu'une trajectoire est stable si tous les exposants sont négatifs. Notre analyse repose sur une interprétation abstraite de la sémantique des séries d'erreurs dans laquelle les coefficients sont des intervalles. $Df^n(x_0)$ est calculée comme suit. On utilise le fait que $Df^n(x_0) = D(f(f^{n-1}(x_0)))$ et on en déduit :

$$\begin{aligned} Df^n(x_0) &= Df^{n-1}(x_0) \times Df(f^{n-1}(x_0)) \\ &= Df^{n-1}(x_0) \times Df(x_{n-1}) \end{aligned}$$

Finalement, par récurrence, $Df^n(x_0) = Df(x_{n-1}) \times \ldots \times Df(x_0)$. Par conséquent, nous allons étudier le produit de matrices d'intervalles $Df^n(x_0) = Df(x_{n-1}) \times \ldots \times Df(x_0)$ dont les valeurs propres indiquent la stabilité des n premières itérées de f. Pour cela, nous allons tout d'abord évaluer (par interprétation abstraite) les n premières itérées de la fonction afin d'obtenir les valeurs $x_0,\ x_1 = f(x_0), \ldots, x_n = f(x_{n-1})$. A chaque étape k, la matrice Df, générée à partir de la syntaxe du programme comme nous l'avons vu à la section 4.3.2 et dont les éléments sont des expressions formelles, est évaluée numériquement en fonction de x_k. On obtient ainsi $Df(x_k)$ et le nouveau produit $Df^{k+1}(x_0) = Df(x_k) \times \ldots \times Df(x_0)$ est calculé.

$Df^k(x_0)$ est une matrice d'intervalle et nous utilisons la méthode des disques de Gerschgorin [Cha88, Gou01] pour calculer une borne supérieure du module de sa plus grande valeur propre. Pour une matrice A de taille $m \times m$, les valeurs propres sont contenues dans un ensemble $\mathcal{D}_1 \cap \mathcal{D}_2$, où $\mathcal{D}_1$ et $\mathcal{D}_2$ sont des disques du plan complexe $\mathbb{C}$ définis par :

$$\mathcal{D}_1 = \cup_{1 \le i \le m} \mathcal{D}_{1,i} \qquad\qquad \mathcal{D}_2 = \cup_{1 \le j \le m} \mathcal{D}_{2,j}$$

$\mathcal{D}_{1,i}$ est le disque de centre a_{ii} et de rayon $r_{1,i} = \sum_{1 \leq j \leq m, j \neq i} |a_{ij}|$. De manière analogue, $\mathcal{D}_{2,j}$ est le disque de centre a_{jj} et de rayon $r_{2,j} = \sum_{1 \leq i \leq m, i \neq j} |a_{ij}|$. On en déduit qu'une borne supérieure du module de la plus grande valeur propre de A est donnée par :

$$G(A) = \max\big(|a_{ii}| + \max(r_{1,i}, r_{2,i}) \ : \ 1 \leq i \leq m\big)$$

Suivant la définition 15, le plus grand exposant de Lyapunov après k itérations est borné par l'exposant de Lyapunov abstrait :

$$\lambda_k^\sharp = \frac{\ln\big(G(Df^k(x_0))\big)}{k} \tag{4.23}$$

Notre algorithme s'arrête dès que $\lambda^\sharp < 0$. Ceci garantit en effet que f^{k_0} est une fonction contractante et, par conséquent, $f^{k_0}(x_0) \sqsubseteq x_0$, où $\sqsubseteq$ est l'ordre terme à terme sur les séries d'erreurs défini à la section 4.2. Tous les termes d'erreur sont décroissants après k_0 itérations et $f^{k_0}(f^{k_0}(x_0)) \sqsubseteq f^{k_0}(x_0) \sqsubseteq x_{k_0}$, etc. Notons que cette inégalité n'implique pas que les termes d'erreur sont toujours décroissants entre les itérations nk_0 et $(n+1)k_0$ pour un entier $n > 0$. Cependant, elle implique qu'aucun terme ne diverge significativement puisque la fonction devient contractante après quelques itérations supplémentaires. Remarquons aussi que l'on peut forcer l'algorithme à s'arrêter après un nombre prédéfini d'itérations k_{max} et à retourner un résultat conservatif consistant à dire que la boucle est instable si notre test d'arrêt n'a pas été vérifié auparavant. En résumé, notre algorithme procède suivant les cinq étapes suivantes :

`tant que` $\lambda^\sharp >= 0$ `faire`

1. `déplier la boucle une fois` : $\rho^\sharp = [\![p]\!]\ \rho^\sharp$; $k = k + 1$
2. `évaluer` Df : $\Delta = Df(\rho^\sharp)$
3. `calculer la nouvelle matrice jacobienne` Df^k : $Df^k = \Delta \times Df^{k-1}$
4. `approcher la plus grande valeur propre` : $e = \mathrm{G}(Df^k)$
5. `calculer le plus grand exposant après` k `itérations` : $\lambda^\sharp = \frac{\ln(e)}{k}$

Nous concluons cette section en présentant des résultats expérimentaux obtenus par une implémentation de notre algorithme. Les deux courbes de la figure 4.3 correspondent à la boucle dont le corps calcule `x = x*x`. L'axe des abscisses correspond aux itérations k, tandis que l'axe des ordonnées donne les valeurs $\lambda_k^\sharp$ des exposants abstraits de Lyapunov.

Le premier cas correspond à un ensemble de trajectoires stables puisque les valeurs initiales sont donnée par la série abstraite $\mathtt{x} = [0.0, 0.95] + [0.0, 0.01]\vec{\varepsilon}_x$. $\lambda_k^\sharp$ devient négatif après quelques itérations, confirmant ce fait. Dans le second cas nous avons $\mathtt{x} = [0.0, 0.95] + [0.0, 0.1]\vec{\varepsilon}_x$ (l'erreur est dix fois plus grande). Même si ce calcul en nombres flottants standard est stable et converge vers zéro, le terme d'erreur d'ordre supérieur est instable comme nous l'avons vu à la section 4.3.1. Ceci est confirmé par les résultats expérimentaux donnés par la seconde courbe de la figure 4.3 où nous pouvons voir que $\lambda_k^\sharp$ est croissant positif.

Notre second exemple utilise un algorithme numérique classique, la méthode itérative de Jacobi, qui permet de trouver une solution approchée à un système d'équations linéaires. Considérons les systèmes :

$$(S_1) \ : \ \left\{ \begin{array}{l} 2x + 3y = 3 \\ 4x + \frac{3}{2}y = 3 \end{array} \right. \qquad\qquad (S_2) \ : \ \left\{ \begin{array}{l} 2x + y = \frac{5}{3} \\ x + 3y = \frac{5}{2} \end{array} \right.$$

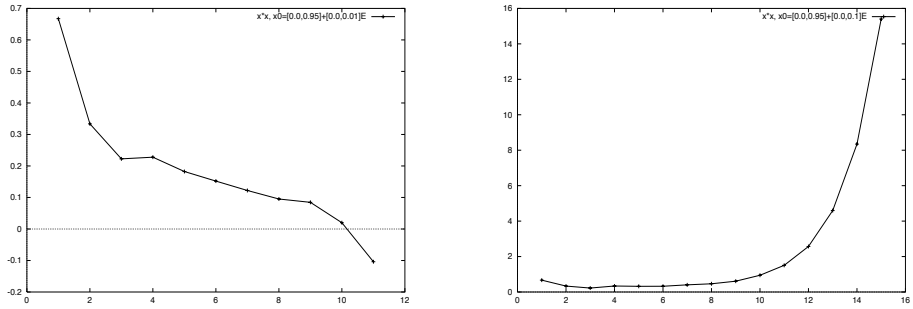

FIG. 4.3 – *Exposants abstraits $\lambda^\sharp$ calculés par notre implémentation de l'algorithme pour le premier exemple de la section 4.3.3.*

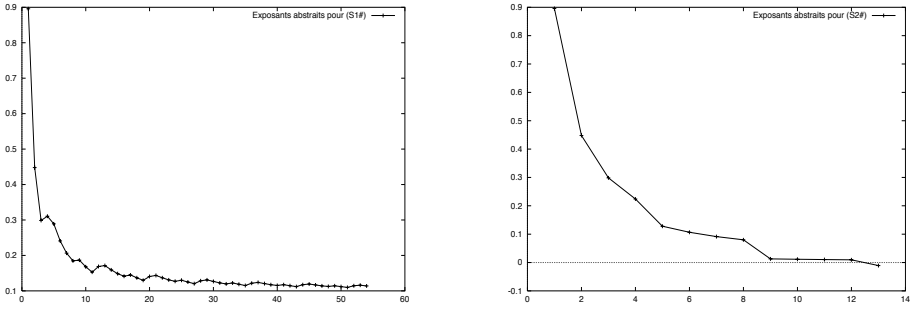

FIG. 4.4 – *Exposants abstraits $\lambda^\sharp$ calculés par notre implémentation de l'algorithme pour le second exemple de la section 4.3.3.*

Pour résoudre (S_1) et (S_2) par la méthode de Jacobi [GVL90], on itère les séquences suivantes :

$$(S_1)\ :\ \begin{cases} x_{n+1} = \frac{3}{2} - \frac{3}{2}y_n \\ y_{n+1} = 2 - \frac{8}{3}x_n \end{cases} \qquad (S_2)\ :\ \begin{cases} x_{n+1} = \frac{5}{6} - \frac{1}{2}y_n \\ y_{n+1} = \frac{5}{6} - \frac{1}{3}x_n \end{cases}$$

(S_1) et (S_2) ont tous deux pour solution $x = \frac{1}{2}$ et $y = \frac{2}{3}$, mais (S_1) est instable pour la méthode de Jacobi tandis que (S_2) est stable[1]. Nous obtenons même, à partir d'un programme C implantant (S_1) en double précision, compilé avec GCC et exécuté sur un processeur Pentium III, au bout de quinze itérations $x = 19408738461538.50$, $y = 13275241025642.0$ si $x_0 = y_0 = 3$. Pour les mêmes valeurs initiales, notre implantation de (S_2) donne après quinze itérations $x = 0.500001$ et $y = 0.666668$. Les courbes de la figure 4.4 donnent les valeurs des exposants abstraits de Lyapunov obtenus pour les deux systèmes abstraits suivants :

$$(S_1^\sharp)\ :\ \begin{cases} x_{n+1} = [1.2, 1.6] - [1.2, 1.6]y_n \\ y_{n+1} = [2.0, 2.5] - [2.5, 3.0]x_n \end{cases} \qquad (S_2^\sharp)\ :\ \begin{cases} x_{n+1} = [0.80, 0.85] - [0.4, 0.6]y_n \\ y_{n+1} = [0.80, 0.85] - [0.30, 0.35]x_n \end{cases}$$

[1]Une condition suffisante pour que la méthode de Jacobi soit stable est que $|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$, avec a_{ij} les coefficients des variables [GVL90].

Les valeurs initiales sont $x_0^\sharp = y_0^\sharp = [2.0, 3.0]$. La courbe correspondant à $(S_2^\sharp)$ devient négative après quelques itérations ce qui prouve la stabilité du calcul correspondant. La courbe correspondant au système instable $(S_1^\sharp)$ demeure positive, ce qui permet de détecter l'instabilité. Remarquons enfin que, dans le cas stable (S_2), la fonction que l'on itère est linéaire et est définie par la matrice : $M = \begin{pmatrix} \frac{5}{6} & \frac{-1}{2} \\ \frac{5}{6} & \frac{-1}{3} \end{pmatrix}$. Cependant, $G(M) = \frac{5}{3}$ et, pour fonctionner, la technique proposée dans [Gou01] dans les cas linéaires et qui consiste à utiliser le test $G(M) < 1$ nécessite aussi de dérouler la boucle.

4.4 Analyse de systèmes hybrides discrets-continus

En général, les analyseurs statiques utilisés pour valider des logiciels embarqués emploient un modèle très approximatif de l'environnement physique dans lequel évoluent les systèmes qu'ils pilotent. Pour prendre un exemple extrême (des exemples plus raisonnables sont donnés en abondance dans les articles dédiés aux systèmes hybrides [ACH+95, Hen96, HHWT98, Mos99]), l'analyse statique du système de contrôle-commande d'un avion devrait être couplé à un modèle abstrait de l'environnement de l'aéronef comprenant les moteurs, les ailes et l'atmosphère elle-même. En effet, les systèmes embarqués interagissent fortement avec leur environnement, utilisant des valeurs physiques fournies par des capteurs et rétroagissant sur ces paramètres grâce à des actionneurs. En pratique, dans des programmes écrits par exemple en langage C, les capteurs correspondent à des variables volatiles et, pour l'analyse, l'utilisateur doit fournir des intervalles de variation de ces données que l'on détermine en général à partir des valeurs minimales et maximales pouvant être captées. Cependant, pour être sûr, un analyseur doit considérer le cas extrême où ces paramètres physiques peuvent passer d'une borne à l'autre de leur intervalle de variation dans des temps très courts (entre deux itérations d'une boucle par exemple), alors qu'en pratique ils évoluent beaucoup plus lentement. Ces hypothèses conduisent un analyseur à sur-approximer très nettement ses résultats. L'abstraction de l'environnement physique est encore plus cruciale pour des systèmes embarqués ne pouvant pas faire l'objet de tests grandeur nature, comme les engins spatiaux dont la sûreté repose uniquement sur des outils de vérification.

Les résultats préliminaires sur ce sujet, décrits dans cette section, ont été présentés dans [Mar05b] et ces travaux sont actuellement poursuivis par O. Bouissou [Bou05]. Des travaux connexes ont été publiés dans [Ber05, BBK+01, Cou05, Fer04, HRP94].

4.4.1 Abstraction des paramètres physiques

Les paramètres physiques d'un programme sont souvent modélisés à l'aide d'équations différentielles. Soit $\mathbb{R}^+$ l'ensemble des réels positifs ou nuls et soit $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. Soit $\mathcal{C}_+^1$ l'ensemble des fonctions $\mathbb{R}^+ \to \overline{\mathbb{R}}$ continuement dérivables par morceaux. $\mathcal{C}_+^1$ est partiellement ordonné par la relation :

$$\forall f_1, f_2 \in \mathcal{C}_+^1,\ f_1 \dot{\prec} f_2 \iff \forall t \in \mathbb{R}^+,\ f_1(t) \leq f_2(t)$$

De plus, nous utilisons la relation $\prec$ définie par :

$$(f_1, f_2) \prec (g_1, g_2) \iff g_1 \dot{\prec} f_1 \wedge f_2 \dot{\prec} g_2$$

Nous avons alors une première correspondance de Galois :

$$\mathcal{T}_0 = \langle \wp(\mathcal{C}^1_+), \subseteq, \cup, \cap, \top_\subseteq, \bot_\subseteq \rangle \underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}} \langle (\mathcal{C}^1_+)^2_\bot, \prec, \vee, \wedge, \top_\prec, \bot_\prec \rangle = \mathcal{T}_1 \tag{4.24}$$

où $\wp(X)$ représente l'ensemble des parties de X, $\top_\subseteq = \mathcal{C}^1_+$, $\bot_\subseteq = \emptyset$, $(\mathcal{C}^1_+)^2_\bot = (\mathcal{C}^1_+ \times \mathcal{C}^1_+) \cup \{\bot_\prec\}$, $\top_\prec = (f_{\bot_{\dot\prec}} \ : \ x \mapsto -\infty, f_{\top_{\dot\prec}} \ : \ x \mapsto +\infty)$, $f_1 \vee f_2 = f$ telle que $\forall x \in \mathbb{R}^+,\ f(x) = f_1(x)$ si $f_1(x) \geq f_2(x)$ et $f(x) = f_2(x)$ sinon. $\wedge$ est défini de façon analogue. Remarquons que pour $f_1 \in \mathcal{C}^1_+$, $f_2 \in \mathcal{C}^1_+$, $f_1 \vee f_2 \in \mathcal{C}^1_+$ et $f_1 \wedge f_2 \in \mathcal{C}^1_+$.

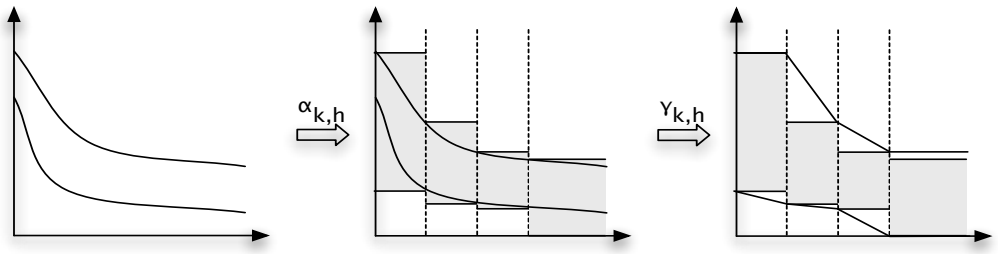

FIG. 4.5 – *Exemple de transformation opérée par $\alpha_{k,h}$ et $\gamma_{k,h}$.*

$\alpha_0(X) = (f^-, f^+)$ tel que $\forall x \in \mathbb{R}^+,\ f^-(x) = \inf\{f(x) \ : \ f \in X\}$ et $f^+(x) = \sup\{f(x) \ : \ f \in X\}$. Pour tout $X \subseteq \mathcal{C}^1_+$, α_0 calcule des fonctions de $\mathcal{C}^1_+$, au moins si, dans $\mathcal{T}_0$, nous nous restreignons à des familles de fonctions F pour lesquelles $\{x \ : \ \exists f \in F,$ la dérivée de f est discontinue en $x\}$ est fini (cette hypothèse est certainement trop restrictive). $\gamma_0(f^-, f^+) = \{f \in \mathcal{C}^1_+ \ : \ f^- \dot\prec f \dot\prec f^+\}$.

Dans un analyseur statique, il serait demandé à l'utilisateur de spécifier le comportement des paramètres physiques à l'aide de valeurs de $\mathcal{T}_1$, c.à.d. à l'aide de paires de fonctions dont l'outil calculerait des approximations numériques. La seconde abstraction, présentée maintenant, précise comment des approximations sûres d'ensembles de fonctions peuvent être obtenus. Intuitivement, f^- et f^+ sont approchées de façon sûre par des fonctions en escalier (voir la figure 4.5). $\mathbb{F}$ étant l'ensemble des nombres flottants, $\mathbb{E}^+$ représente l'ensemble des fonctions en escalier $e \ : \ \mathbb{R}^+ \to \overline{\mathbb{F}}$.

$$\mathcal{T}_1 = \langle (\mathcal{C}^1_+)^2_\bot, \prec, \vee, \wedge, \top_\prec, \bot_\prec \rangle \underset{\alpha_{k,h}}{\overset{\gamma_{k,h}}{\leftrightarrows}} \langle \mathbb{E}_+ \times \mathbb{E}_+, \prec, \vee, \wedge, \top_\prec, \bot_\prec \rangle = \mathcal{T}_2 \tag{4.25}$$

k est un entier positif et h est un nombre flottant positif. Nous avons :

$$\alpha_{k,h}(f^-, f^+) = (\alpha^-_{k,h}(f^-), \alpha^+_{k,h}(f^+))$$

tel que :

$$\alpha^-_{k,h}(f^-) = e^- \ : \ \forall i \ : \ 0 \leq i < k,\ \forall t \ : \ ih \leq t < (i+1)h,\ e^-(t) \leq f^-(t) \tag{4.26}$$

$$\alpha^+_{k,h}(f^+) = e^+ \ : \ \forall i \ : \ 0 \leq i < k,\ \forall t \ : \ ih \leq t < (i+1)h,\ e^+(t) \geq f^+(t) \tag{4.27}$$

$$\forall t \geq kh,\ e^-(t) \leq f^-(t) \text{ et } e^+(t) \geq f^+(t) \tag{4.28}$$

Les équations (4.26) à (4.28) n'indiquent pas comment calculer $\alpha^-_{k,h}$ et $\alpha^+_{k,h}$, mais tout algorithme qui vérifie ces conditions est une abstraction correcte. Un tel algorithme reposant sur une méthode de Runge-Kutta modifiée sera présenté à la section 4.5. Cet algorithme s'avère très performant en comparaison à des algorithmes classiques d'approximation par intervalles [NJ01]. En

outre, pour des fonctions de $\mathcal{T}_1$ définies par un système d'équations différentielles, les k premières marches de e^- et e^+ peuvent être calculées par cette méthode (plus précisément une méthode d'ordre 4 sûre, à pas adaptatif) qui utilise la norme IEEE 754 pour contrôler les erreurs d'arrondi. Pour le dernier pas correspondant aux pas $t \geq kh$, des approximations sûres peuvent aussi être calculées. Par exemple, si f^- et f^+ sont $\mathcal{C}^2_+$, le signe de la dérivée seconde peut permettre de trouver une approximation garantie.

D'autres abstractions intéressantes pourraient être imaginées pour des classes de fonctions particulières comme les fonctions périodiques (par exemple dans l'espace de Fourier). Par exemple, les fonctions trigonométriques correspondent à des entrées réalistes d'algorithmes de traitement digital du signal utilisés dans les systèmes embarqués [Mar04]. Remarquons aussi que, par ailleurs, pour $k' > k$, $\alpha_{k',h}$ est une abstraction plus précise que $\alpha_{k,h}$. Une remarque similaire peut être faite à propos de h.

$\gamma_{k,h}$ doit calculer une paire de fonctions de $\mathcal{C}^1_+$ mais, par exemple, $\vee\{f \in \mathcal{C}^1_+ \ : \ f \prec e^+\} \notin \mathcal{C}^1_+$. Une approximation supplémentaire doit être réalisée et nous choisissons de définir $\gamma^+_{k,h}(e^+)$ comme étant une interpolation linéaire des points $(ih, e^+(ih))$ et $((i+1)h, e^+((i+1)h))$, pour $0 \leq i < k$, comme on peut le voir à la figure 4.5.

4.4.2 Une analyse simple

Revenons maintenant sur les problèmes de précision numérique. Pour des systèmes embarqués critiques, une propriété de sûreté importante devrait consister à montrer qu'une implémentation prend les même décisions qu'un système idéal travaillant avec des nombres réels. Soit p un programme dont les entrées sont données par des capteurs associés à des variables volatiles $x_1, \ldots, x_n$. Pour $1 \leq i \leq n$, $F_i \subseteq \wp(\mathcal{C}^1_+)$ est un ensemble de fonctions concrètes modélisant les comportements possible du capteur associé à x_i.

Soient $[\![p]\!]_{\mathbb{R}}$ et $[\![p]\!]_{\mathbb{F}}$ les sémantiques collectrices de p dans lesquelles les calculs sont effectués en nombres réels et flottants, respectivement. Nous souhaitons comparer les traces de $[\![p]\!]_{\mathbb{R}}(F_1, \ldots, F_n)$ et $[\![p]\!]_{\mathbb{F}}(F_1^\sharp, \ldots, F_n^\sharp)$, où $F_i^\sharp = \alpha_0 \circ \alpha_{k,h}(F_i)$, $1 \leq i \leq n$. On suppose que les boucles sont précisément cadencées, c.à.d. que nous connaissons précisément la durée de chaque itération. Ces informations sont généralement connues, les logiciels embarqués étant des programmes temps réel pour lesquels les concepteurs savent, par exemple, que telle boucle s'exécute à x Hz. Aussi avons-nous au minimum un intervalle de temps précis pour dater chaque point d'une trace d'exécution. Des outils de calcul de pires temps d'exécution peuvent aussi être utilisés pour collecter cette information [FHL$^+$01].

Pour définir notre analyse $[\![.]\!]^\sharp_{\mathbb{EH}}$ pour les systèmes hybrides, nous nous fondons sur une abstraction de la sémantique $[\![.]\!]_{\mathbb{E}}$ de l'erreur globale introduite à la section 2.3 dans laquelle les termes flottants et d'erreurs sont des intervalles (voir l'abstraction de la section 4.2) et nous l'enrichissons d'une nouvelle règle pour les variables volatiles : dans l'intervalle de temps $[t, t']$, la valeur du capteur x_i est :

$$[\![x_i]\!]^\sharp_{\mathbb{EH}} = [a, b]\vec{\varepsilon}_f + [-u, u]\vec{\varepsilon}_e \tag{4.29}$$

avec :

$$F_i^\sharp = (F_i^{-\sharp}, F_i^{+\sharp})$$

$$a = \min(F_i^{-\sharp}([t, t'])) \qquad b = \max(F_i^{+\sharp}([t, t']))$$

$$u = \max(ulp(|a|), ulp(|b|))$$

$[\![.]\!]^\sharp$ permet de détecter des divergences de flot de contrôle entre $[\![.]\!]_{\mathbb{R}}$ et $[\![.]\!]_{\mathbb{F}}$, c.à.d. de détecter les situations dans lesquelles un code embarqué ne prendra pas les mêmes décisions qu'un système idéal qui calculerait dans les réels. Pour une valeur $v = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$ de $[\![.]\!]^\sharp_{\mathbb{EH}}$, ces divergences correspondent aux conditions $cond(v)$ pour lesquelles $[\![cond(v)]\!]_{\mathbb{F}} = cond(f) \neq cond(f+e) = [\![cond(v)]\!]_{\mathbb{R}}$.

Remarquons tout de même que l'analyse que nous venons de présenter est trop simple pour être utile en pratique. En effet, des divergences "faibles" de flot de contrôle entre $[\![.]\!]_{\mathbb{R}}$ et $[\![.]\!]_{\mathbb{F}}$ sont courantes et acceptables en pratique. Néanmoins, $[\![.]\!]^\sharp_{\mathbb{EH}}$ ouvre la voie à la définition d'analyses plus intéressantes. Une propriété de sûreté réaliste pourrait s'exprimer ainsi : "étant donné un programme p qui, dans $[\![.]\!]_{\mathbb{R}}$, atteint un certain point de contrôle entre les instants t_0 et t_1, alors, dans $[\![.]\!]_{\mathbb{F}}$, p atteint ce même point entre les instants t'_0 et t'_1 avec $|t_0 - t'_0| < x$ et $|t_1 - t'_1| < y$. Par exemple si une alarme doit être activée lorsque la valeur d'un paramètre atteint un certain seuil au temps t, alors nous souhaitons que l'implémentation du système active effectivement cette alarme dans un délai acceptable autour du temps t et nous souhaitons borner ce délai. Une analyse permettant de vérifier ce type de propriétés a été récemment définie par O. Bouissou [Bou05].

4.5 Résolution sûre de systèmes d'équations différentielles

Nous avons vu au cours de la section précédente qu'il était nécessaire de connaître un encadrement garanti des valeurs correspondant aux entrées continues d'un système hybride afin de pouvoir analyser sûrement celui-ci. Ces données sont le plus souvent définies par un système d'équations différentielles qu'il est nécessaire de savoir résoudre de façon sûre, c.à.d. en garantissant que les solutions approchées calculées numériquement encadrent en tout point la ou les solutions exactes. Peu de travaux ont été menés dans cette direction, les plus aboutis concernant les logiciels ADIODES [Sta05] et VNODE [NJ01] et qui reposent sur l'arithmétique d'intervalle. En outre, ils s'avèrent lents et peu précis, notamment à cause de l'effet enveloppant. Les travaux présentés ci-dessous sont résumés dans [BM06a].

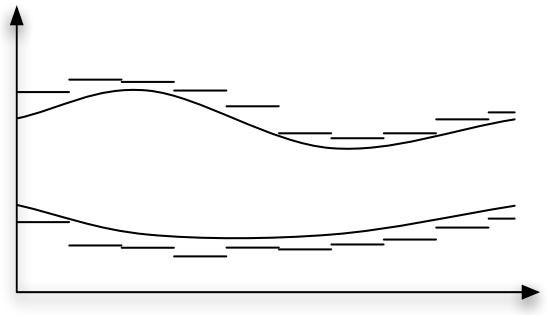

FIG. 4.6 – *Exemple de fonctions en escalier calculées par la méthode de résolution sûre présentée à la section 4.5.*

Nous présentons ici un nouvel algorithme, développé en collaboration avec O. Bouissou, pour encadrer précisément et sûrement, les solutions d'un système d'équations différentielles. Cet algorithme est une adaptation d'une méthode de Runge-Kutta d'ordre 4 [CK90, PTVF92]. Comme illustré à la figure 4.6, il permet de calculer une fonction en escalier à valeurs dans les flottants

strictement supérieure à la solution exacte d'une équation différentielle (une sous-approximation peut être calculée de manière analogue). Formellement, étant donnée une équation de la forme

$$\dot{y} = f(y, x)$$

avec $f : \mathbb{R}^+ \to \mathbb{R}$ une fonction $\mathcal{C}^4$ (continue dérivable quatre fois par morceaux), il permet de trouver une fonction en escalier flottante φ telle que :

$$\forall t \in \mathbb{R}^+,\ y(t) \leq \varphi(t) \tag{4.30}$$

La méthode de Runge-Kutta que nous utilisons permet un contrôle de l'erreur et une modification dynamique du pas d'intégration en fonction d'une tolérance maximale pour chaque pas ϵ fixée par l'utilisateur. Par rapport à l'algorithme original, la fonction de contrôle du pas a été modifiée pour permettre d'obtenir une sur-approximation sûre grâce à une technique de "prédiction-correction"[2] :

1. on obtient, par un calcul très précis, une approximation de la valeur au pas suivant ;
2. on vérifie par un calcul grossier que ce pas est valide.

Etant donnés $\dot{y} = f(x, y)$ et une valeur initiale y_0 au point x_0, rappelons tout d'abord qu'une méthode de Runge-Kutta d'ordre 4 consiste à calculer une approximation y_1 de y en $x_0 + h$ de la façon suivante :

$$\begin{aligned}
k_1 &= f(x_0, y_0) \\
k_2 &= f(x_0 + h/2, y_0 + k_1/2) \\
k_3 &= f(x_0 + h/2, y_0 + k_2/2) \\
k_4 &= f(x_0 + h, y_0 + k_3) \\
y_1 &= y_0 + \frac{h}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big)
\end{aligned}$$

On peut alors exprimer, d'une part, l'erreur commise à chaque pas et, d'autre part, la propagation de l'erreur, en fonction des quatre premières dérivées de f. Ces deux étapes, qui correspondent aux points clés de notre approche, sont détaillées dans les paragraphes suivants. Ils découlent des calculs d'estimation de l'erreur commise par la méthode de Runge-Kutta classique [Bie51, Car58].

Erreur commise en un pas Pour trouver une sur-approximation de l'erreur η commise en un pas (voir la figure 4.7), on suppose que f est quatre fois dérivable par morceaux et que toutes les dérivées sont bornées. Il est alors possible d'exprimer η en fonction de $f^{(i)},\ i = 1, 2, 3, 4$. Pour cela on pose :

$$\begin{aligned}
k_1(x) &= f(x_0, y_0) \\
k_2(x) &= f(x_0 + (x - x_0)/2, y_0 + k_1(x)/2) \\
k_3(x) &= f(x_0 + (x - x_0)/2, y_0 + k_2(x)/2) \\
k_4(x) &= f(x_0 + (x - x_0), y_0 + k_3(x)) \\
\tilde{y}(x) &= y_0 + \frac{(x - x_0)}{6}(k_1(x) + 2k_2(x) + 2k_3(x) + k_4(x))
\end{aligned}$$

[2] L'expression "prédiction-correction" est parfois employée avec une signification différente de celle utilisée ici.

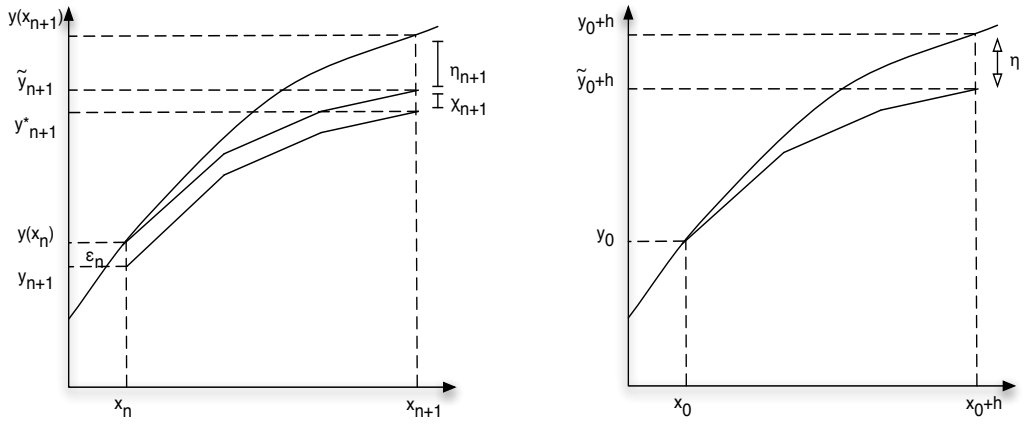

FIG. 4.7 – *Gauche : erreur η commise en un pas par la méthode de Runge-Kutta d'ordre 4. Droite : propagation de l'erreur par la méthode de Runge-Kutta d'ordre 4.*

et l'on cherche à sur-approximer la quantité :

$$\Delta = |y(x_0 + h) - \tilde{y}(x_0 + h)| \tag{4.31}$$

Sachant que $\forall i \in \{0, 1, 2, 3, 4\},\ \frac{d^i(y-\tilde{y})}{dx^i}(x_0) = 0$, par un développement en séries de Taylor, on obtient :

$$\eta = \tilde{y}(x_0 + h) - y(x_0 + h) = \frac{h^5}{5!}\frac{d^5(\tilde{y} - y)}{dx^5}(\xi) \text{ avec } \xi \in [x_0, x_0 + h]$$

Par conséquent,

$$\eta \leq \frac{h^5}{5!}.\left(\left|\frac{d^4 f}{dx^4}(\xi)\right| + \left|\frac{d^5\tilde{y}}{dx^5}(\xi)\right|\right)$$

Ainsi, on a d'un côté

$$\left|\frac{d^5\tilde{y}}{dx^5}(\xi)\right| \leq \frac{d^5\tilde{y}}{dx^5}([x_0, x_0 + h])$$

et, par ailleurs, pour majorer $\frac{d^4 f}{dx^4}(\xi)$, il est nécessaire de disposer d'une borne globale sur $\frac{d^4 f}{dx^4}$. Il semble cependant possible d'alléger cette hypothèse.

Propagation de l'erreur Si l'on suppose maintenant connaître l'erreur ϵ_n au pas n, il reste à déterminer l'erreur au pas $n + 1$ (voir la figure 4.7). Nous avons :

$$|\epsilon_{n+1}| \leq |\tilde{y}_{n+1} - y^*_{n+1}| + |y^*_{n+1} - y(x_{n+1})| \tag{4.32}$$

avec :

$$\tilde{y}_{n+1} = y(x_n) + \epsilon_n + \frac{h_n}{6}\big(k_1(x_{n+1}) + 2k_2(x_{n+1}) + 2k_3(x_{n+1}) + k4(x_{n+1})\big)$$

$$k_1(x_{n+1}) = f(x_n, y_n + \epsilon_n)$$

Par un développement en séries de Taylor autour de (x_n, y_n), on obtient :

$$\tilde{y}_{n+1} = y^*_{n+1} + \epsilon_n\,[1 + \cdots]$$

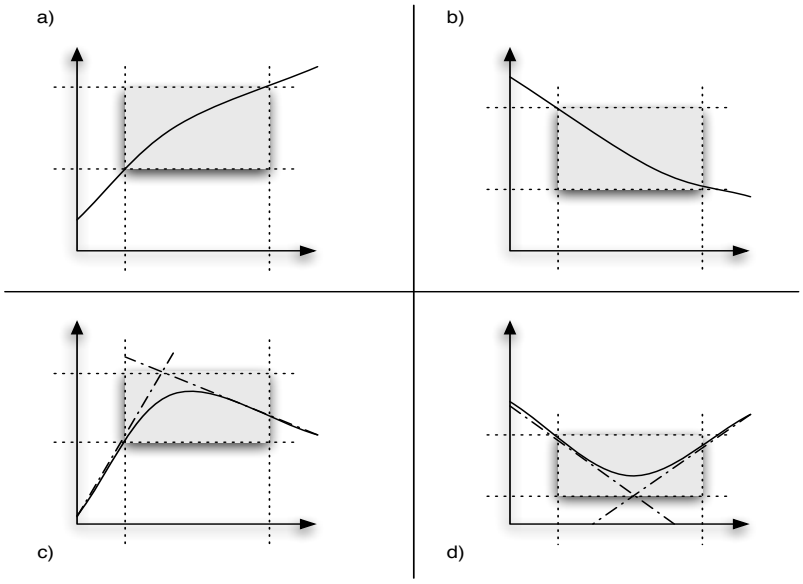

FIG. 4.8 – *Contrôle du pas selon les valeurs de $\dot{y}_n$ et $\dot{y}_{n+1}$.*

et finalement :

$$\chi_{n+1} = \tilde{y}_{n+1} - y^*_{n+1} = \epsilon_n \left[1 + \Phi(x)\right] \tag{4.33}$$

où la fonction Φ s'exprime en fonction de la dérivée première de f. Il est alors possible de majorer cette erreur en calculant $\Phi\big([x_n, x_{n+1}]\big)$. Finalement, l'erreur globale peut être bornée, si on connaît une borne M sur les dérivées de f, par :

$$\forall n,\ |\epsilon_n| \leq K.\frac{e^{(n+1)hM} - 1}{e^{hM} - 1} \tag{4.34}$$

avec

$$K = \frac{h^5}{5!}.C \quad \text{et} \quad C \geq \left(\left|\frac{d^4 f}{dx^4}\right| + \left|\frac{d^5 \tilde{y}}{dx^5}\right|\right)$$

Cependant, ces bornes sont fortement sur-approximées et, dans une implémentation, nous envisageons d'utiliser plutôt les formules précédentes.

Contrôle du pas Soit y_n l'approximation de $y(x_n)$. La méthode décrite précédemment permet de déterminer y_{n+1}, approximation de $y(x_n + h)$, ainsi qu'une majoration de l'erreur e_{n+1} commise au cours de ce calcul. Plusieurs cas doivent être distingués pour valider ce pas :

1. Si $\dot{y}_n \geq 0$ et $\dot{y}_{n+1} \geq 0$ (figure 4.8-a), alors on souhaite utiliser y_{n+1} comme marche courante de la fonction en escalier. La vérification consiste à calculer $I = f\left([x_n, x_{n+1}], [y_n, y_{n+1}]\right)$ et à tester si $I \cap \mathbb{R}^- = \emptyset$. Si oui, le pas est accepté et on prend $y_{n+1} + 2\epsilon$ comme valeur pour la marche supérieure. h est divisé par deux et le calcul de y_{n+1} est recommencé sinon.
2. Le cas $\dot{y}_n \leq 0$ et $\dot{y}_{n+1} \leq 0$ (figure 4.8-b) est symétrique au précédent.
3. Si $\dot{y}_n \geq 0$ et $\dot{y}_{n+1} \leq 0$ (figure 4.8-c), la dérivée $g(y, x) = \ddot{y}$ s'annule au moins en un point du pas. Pour pouvoir utiliser comme marche supérieure une approximation linéaire, on s'assure que la dérivée seconde g est négative sur tout le pas : si $g\left([x_n, x_{n+1}], [y_n, y_{n+1}]\right) \cap \mathbb{R}^+ \neq \emptyset$, alors le pas courant est refusé et l'on recommence avec un pas de taille deux fois plus petite.

Require: $\mathbf{yerr}_{\mathbb{M}}$,$\mathbf{y[n+1]}_{\mathbb{M}}\mathbf{y[n+1]}_{\mathbb{E}}$=yout+$\mathbf{e}_{\mathcal{I}_{\mathbb{M}}}$,eps,F,dydt,x,y
  if $\mathbf{e}_{\mathcal{I}_{\mathbb{M}}}$+$\mathbf{yerr}_{\mathbb{M}} \geq$ eps **then**
    return false
  end if
  $\mathbf{dyout}_{\mathbb{M}} \leftarrow$F(yout,x[n]+$h_n$)
  $\mathbf{I}_{\mathcal{I}_{\mathbb{M}}} \leftarrow \mathbf{F}_{\mathcal{I}_{\mathbb{M}}}$([y[n]-$h_n$/100,yout+$h_n$/100],[x[n],x[n]+$h_n$])
  if $\mathbf{dydt[n]}_{\mathbb{M}} \geq 0$ **then**
    if $\mathbf{dyout}_{\mathbb{M}} \geq 0$ **then**
      if $\mathbf{I}_{\mathcal{I}_{\mathbb{M}}} \cap R_*^- \neq \emptyset$ **then**
        if $|\min(\mathbf{I}_{\mathcal{I}_{\mathbb{M}}}).h_n| \geq$eps **then**
          return false
        end if
      end if
      return true
    else
      $\boldsymbol{\alpha}_{\mathbb{M}} \leftarrow$(y[n+1]-y[n]-dydt[n+1]*x[n+1]+dydt[n]*x[n])/(dydt[n]-dydt[n+1]) ;
      $\mathbf{dI}_{\mathcal{I}_{\mathbb{M}}} \leftarrow$dF$\mathbf{dF}_{\mathcal{I}_{\mathbb{M}}}$([min(y[n],yout),$\alpha$],[x[n],x[n]+$h_n$]) ;
      if $\mathbf{dI}_{\mathcal{I}_{\mathbb{M}}} \cap \mathbb{R}_*^+ \neq \emptyset$ **then**
        return false
      else
        return true
      end if
    end if
  else
    ...
  end if

FIG. 4.9 – *Algorithme récapitulatif de la méthode de Runge-Kutta garantie présentée à la section 4.5.*

4. Le cas $\dot{y}_n \leq 0$ et $\dot{y}_{n+1} \geq 0$ (figure 4.8-d) est symétrique au précédent.

Implémentation Si l'algorithme résultant de la combinaison des trois étapes précédentes était exécuté en utilisant des nombres réels, il serait correct. Cependant, nous souhaitons obtenir des fonctions en escalier flottantes, et les calculs sont réalisés en machine en nombres flottants. Il est donc nécessaire de tenir compte des erreurs dues au calcul numérique lors de cette implémentation et de s'assurer que le résultat flottant sur-approxime bien le résultat réel. Pour cela nous allons utiliser plusieurs domaines numériques présentés au cours du chapitre 2 :

- on utilise des flottants multi-précision (ensemble $\mathbb{M}$) pour les calculs, pour limiter les erreurs de perte de précision ;
- certains calculs sont effectués en utilisant le domaine $\mathcal{I}_{\mathbb{M}}$ des intervalles multi-précision ;
- pour le calcul donné par la méthode de Runge-Kutta, on utilise le domaine $\mathbb{E}$ des flottants avec erreur globale introduit à la section 2.3.

L'algorithme ainsi obtenu est donné sous forme synthétique à la figure 4.9 et une première implémentation (non optimisée) a été réalisée. Elle permet de comparer notre méthode, en termes de

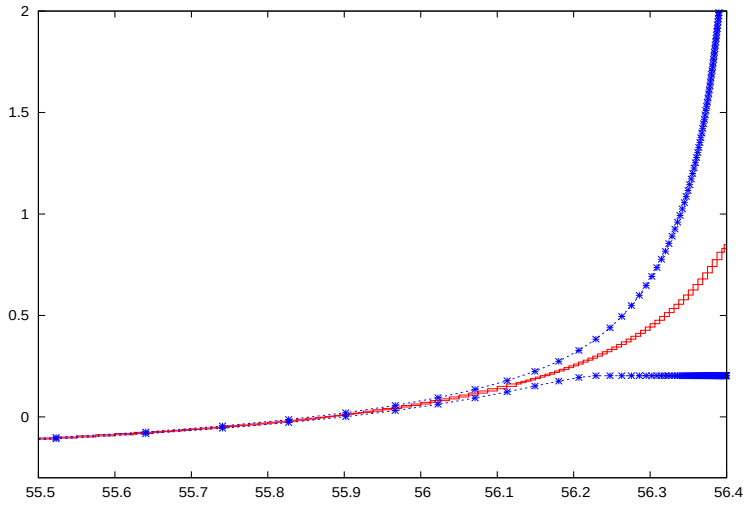

FIG. 4.10 – *Résultats produits par VNODE et la méthode de la section 4.5 pour le problème des deux corps de l'équation (4.35).*

précision, au logiciel VNODE. Considérons, par exemple, le problème des deux corps qui consiste à résoudre le système suivant :

$$\left\{\begin{array}{rcl} f' & = & h \\ g' & = & l \\ h' & = & -1.0.f/((f.f+g.g).sqrt(f.f+g.g)) \\ l' & = & -1.0.g/((f.f+g.g).sqrt(f.f+g.g)) \end{array}\right. \qquad \left\{\begin{array}{rcl} f_0 & = & 0.1 \\ g_0 & = & 0.0 \\ h_0 & = & 0.0 \\ l_0 & = & 4.36 \end{array}\right. \tag{4.35}$$

Les résultats produits par notre méthode ainsi que par VNODE sont donnés à la figure 4.10. En outre, nous pouvons remarquer que tous les points d'approximation calculés par VNODE sont contenus dans notre sur-approximation, sauf en un point où notre méthode est moins précise. De plus l'algorithme que nous venons de présenter s'avère plus stable : VNODE devient près imprécis pour des temps $t \geq 55$.

4.6 Perspectives

De nombreux travaux doivent encore être menés en analyse statique, à partir de la sémantique des séries d'erreurs. Nous avons pu voir, au cours de ce chapitre, que la définition d'heuristiques pour le problème du partitionnement dynamique demeurait un problème ouvert et que l'analyse de systèmes hybrides discrets-continus était un axe de recherche à moyen et long terme. Concernant ce dernier sujet, des avancées intéressantes ont récemment été réalisées par O. Bouissou [Bou05].

L'analyse présentée à la section 4.3 et qui repose sur le calcul d'exposants de Lyapunov abstraits ouvre la voie à une autre axe de recherche, consistant à s'appuyer sur des résultats de la théorie des systèmes dynamiques et des systèmes de fonctions itérées [ASY96, Bea91, Eda95, Eda96] pour définir de nouvelles analyses statiques. Il serait par exemple intéressant de calculer, pour des

fonctions correspondant à des corps de boucles, les bassins d'attraction, afin de déterminer si l'ensemble des données pouvant être utilisées en entrée chevauchent des zones critiques, ou encore d'étudier les bifurcations possibles de ces mêmes fonctions.

Plus généralement, une autre direction de recherche, importante en pratique, concerne la définition de domaines relationnels ou faiblement relationnels. Ces domaines ont montré leur efficacité dans le cadre de la détection d'erreurs à l'exécution [Min05] et permettraient d'améliorer sensiblement la qualité des analyses pour la précision numérique. De premiers travaux dans cette direction ont été menés par E. Goubault et S. Putot [GP05]. A partir de la sémantique des séries d'erreurs, il serait particulièrement intéressant de définir des analyses tenant compte des relations entre certains termes d'erreurs d'une part, et, d'autre part, entre les valeurs flottantes et les termes d'erreurs.

Chapitre 5

Les analyseurs Fluctuat

5.1 Introduction

Deux analyseurs Fluctuat ont été développés au CEA, par E. Goubault, S. Putot et moi-même, l'un pour le langage C et l'autre pour l'assembleur du processeur TMS320. Ces analyseurs sont fondés sur la sémantique des séries d'erreurs vue au chapitre 3. Fluctuat pour assembleur est utilisé par Airbus, à des fins de certification, tandis que Fluctuat C est utilisé notamment par Airbus, l'IRSN et Hispano-Suiza. L'analyseur d'assembleur est présenté à la section 5.2 et Fluctuat C à la section 5.3.

5.2 Fluctuat pour assembleur TMS320

Les processeurs pour le traitement digital du signal (DSP) sont couramment utilisés dans les systèmes embarqués critiques pour effectuer des opérations de bas niveau, souvent critiques, telles que le contrôle de périphériques ou l'acquisition et la transformation de données fournies par des capteurs. Les résultats de ces calculs étant souvent des données critiques pour le bon fonctionnement d'un système, il est nécessaire de les valider pour des raisons de sûreté.

Dans cette section, nous allons présenter l'analyseur statique Fluctuat pour assembleur TMS320 [Mar04], un DSP très répandu dans l'industrie [Tex97]. Cet analyseur a été conçu pour pouvoir valider des codes de taille industrielle et est actuellement utilisé par Airbus. La conception de ce logiciel nous a amené à définir une nouvelle analyse statique par interprétation abstraite pour les programmes écrits en assembleur. Contrairement aux autres approches développées ces dernières années [FHL$^+$01, XRB00, XRB01], nous nous intéressons à des codes relogeables, dans lesquels les adresses sont définies par des étiquettes. Ces codes contiennent des informations utiles à l'analyse concernant les structures de données ou les segments de code ou de données. Une autre originalité de notre analyse est qu'elle ne repose pas sur la construction a priori d'un graphe de flot de contrôle qu'il est généralement difficile de reconstituer pour des programmes assembleurs [The01]. Cela nous permet notamment de traiter précisément les branchements à des adresses calculées dynamiquement. Nous pouvons ainsi, par exemple, analyser efficacement des programmes récursifs. Notons enfin que cette analyse dépend très peu de la spécificité de l'assembleur utilisé et qu'elle semble aisément transposable à d'autres processeurs. Nous reviendrons sur ce point à la fin de cette section.

5.2.1 Interprétation abstraite de programmes écrits en assembleur

Nous allons maintenant présenter la sémantique abstraite utilisée par notre analyseur en utilisant un langage assembleur simplifié, représentatif de celui du TMS320. Comme on peut le voir à la figure 5.1, nous disposons d'opérations arithmétiques entières et flottantes (`ADDI`, `ADDF`, etc.), de l'instruction `MV` pour la recopie de valeurs, des instructions `PUSH` et `POP` pour manipuler la pile, d'une instruction (`CALL`) pour l'appel de procédures et de branchements conditionnels et inconditionnels (`BR`, `BLT` et `BEQ`). `CMPI` et `CMPF` comparent les opérandes src_1 et src_2 et modifie les drapeaux `LT` et `EQ` selon que $src_1 < src_2$ et $src_1 = src_2$, respectivement. `BLT` (resp. `BEQ`) exécute les branchements si `LT` (resp. `EQ`) est vrai. `CALL` empile le registre de pc (pointeur de code) et effectue le branchement.

$$
\begin{array}{rcll}
mnem & ::= & \texttt{ADDI}\ src_1, src_2, dst \mid \texttt{SUBI}\ src_1, src_2, dst \mid \texttt{MULI}\ src_1, src_2, dst & \\
& \mid & \texttt{ADDF}\ src_1, src_2, dst \mid \texttt{SUBF}\ src_1, src_2, dst \mid \texttt{MULF}\ src_1, src_2, dst \mid \texttt{MV}\ src, dst & \\
& \mid & \texttt{BR}\ src \mid \texttt{BLT}\ src \mid \texttt{BEQ}\ src \mid \texttt{CMPI}\ src_1, src_2 \mid \texttt{CMPF}\ src_1, src_2 & \\
& \mid & \texttt{PUSH}\ src \mid \texttt{POP}\ dst \mid \texttt{CALL}\ src & \\
\\
src, dst & ::= & val & (Immediat) \\
& \mid & R_0 \mid R_1 \mid \ldots \mid R_n & (Registre) \\
& \mid & *val & (Direct) \\
& \mid & R_i(R_j) & (Indirect) \\
\\
val & ::= & int \mid float \mid label & \\
\\
decl & ::= & \texttt{label :}\ \texttt{.INT}\ (\texttt{int}\ list) \mid \texttt{label :}\ \texttt{.FLOAT}\ (\texttt{float}\ list) & \\
\\
prog & ::= & (decl\ list)((\texttt{label})?\ mnem)\ list &
\end{array}
$$

FIG. 5.1 – *Syntaxe du langage.*

Les arguments des opérandes sources et destinations des opérations peuvent correspondre à quatre modes d'adressage. Pour simplifier, tous les modes d'adressage sont autorisés pour toutes les opérations, ce qui n'est pas le cas généralement en pratique. Dans le mode immédiat, val représente une valeur pouvant correspondre à un entier, un flottant ou une adresse relogeable définie par une étiquette. Le mode registre permet de lire ou d'écrire dans les registres. On utilise un jeu de registres généraux notés $R_0, \ldots, R_n$. Dans le mode direct $*val$ représente la valeur stockée à l'emplacement mémoire décrit par val. Enfin, en mode indirect, $R_i(R_j)$ représente l'emplacement mémoire $a + b$, où a et b sont les valeurs contenues dans R_i et R_j.

Un programme est composé de deux segments, pour les données et pour le code. Le segment des données est composé de déclarations de zones mémoires initialisées et l'on suppose que dans le segment de code toutes les opérations sont codées sur un seul mot mémoire exactement.

Les domaines abstraits des états mémoire globaux et des valeurs élémentaires sont fortement connectés par le fait que notre langage contient des pointeurs. Nous présentons tout d'abord la façon dont sont abstraits les états mémoire, en supposant que les valeurs élémentaires appartiennent à un certain domaine abstrait $\mathbb{D}^\sharp$ que nous définirons ultérieurement. Un état mémoire est représenté par un quintuplet $\mathcal{M} = (\rho, \mu, \sigma, \varphi, pc)$ dans lequel :

– $\rho : [0, n] \to \mathbb{D}^\sharp$ associe à chaque registre R_i, $1 \leq i \leq n$, une valeur abstraite ;

– μ : $Loc \to \mathbb{D}^\sharp$ associe à chaque emplacement mémoire une valeur abstraite ;
– σ est la pile abstraite que nous définirons plus tard ;
– φ : $flags \to \mathcal{B}^\sharp$ associe à chaque drapeau un booléen abstrait $b \in \mathcal{B}^\sharp$ avec :

$$\mathcal{B}^\sharp = \{\texttt{true}, \texttt{false}, \top, \bot\}$$

– $pc \in \mathbb{D}^\sharp$ est le pointeur d'instruction.

μ mémorise les valeurs stockées dans le segment des données. Un emplacement $\ell \in Loc$ est une paire (lab, i) dans laquelle lab est une étiquette définie dans le segment des données et i est un entier correspondant à un décalage. Ainsi, (lab, i) pointe sur la i-ième valeur définie dans la déclaration de lab. Par exemple, la déclaration `d : .INT 1,2` force $\mu(\texttt{d}, 0) = 1$ et $\mu(\texttt{d}, 1) = 2$. Dans notre implémentation de l'analyseur, si n valeurs sont allouées à l'étiquette lab, alors un accès à $\mu(lab, m)$, pour $m \geq n$, génère une erreur comparable à un dépassement des bornes d'un tableau dans un langage de plus haut niveau. La fonction φ associe aux drapeaux du processeur un booléen abstrait Nous considérons uniquement deux drapeaux, `EQ` et `LT`, qui sont modifiés par les opérations `CMPI` et `CMPF`.

Le domaine $\mathbb{D}^\sharp$ réunit celui des entiers, des flottants et des pointeurs vers les segments des données et du code, respectivement notés $\mathbb{I}$, $\mathbb{E}$, $\mathbb{D}_*$ et $\mathbb{T}_*$. $\mathbb{I}$ est le domaine des intervalles à bornes entières et $\mathbb{E}$ représente un des domaines des séries d'erreurs vus au chapitre 3.

Soit $\mathcal{L}_D$ l'ensemble des étiquettes définies dans le segment des données. Le domaine abstrait des pointeurs de données est

$$\mathbb{D}_* = \wp(\mathcal{L}_D) \times \mathbb{I} \tag{5.1}$$

La concrétisation d'une valeur abstraite $(L, I) \in \mathbb{D}_*$ est :

$$\gamma_{\mathbb{D}_*}(L, I) = \big\{(lab, i) \ : \ lab \in L, \ i \in I\big\} \tag{5.2}$$

Les opérations d'union et d'intersection sur $\mathbb{D}_*$ sont définis composante par composante. Les pointeurs vers le segment de code sont définis de façon analogue, en utilisant l'ensemble $\mathcal{L}_T$ des étiquettes définies dans ce dernier au lieu de $\mathcal{L}_D$. Finalement, le domaine $\mathbb{D}^\sharp$ des valeurs abstraites est défini par :

$$\mathbb{D}^\sharp = \{\bot_{\mathbb{D}^\sharp}, \top_{\mathbb{D}^\sharp}\} \cup \mathbb{I} \cup \mathbb{E} \cup \mathbb{D}_* \cup \mathbb{T}_*$$

Remarquons que notre gestion des pointeurs permet à un analyseur de détecter certaines erreurs à l'exécution spécifiques à l'assembleur telles que la lecture ou l'écriture de données hors du segment correspondant, des branchements hors du segment du code, des erreurs semblables aux dépassements des bornes des tableaux, des références absolues à la mémoire dans des codes en principe relogeables, des erreurs de types ou encore l'utilisation de registres non initialisés. Même si dans certains cas un programmeur peut intentionnellement écrire des programmes effectuant ce type d'opérations, ils sont en général peu recommandables et nous pensons qu'un outil de validation doit les rapporter.

La sémantique abstraite est présentée à la figure 5.2. $\mathcal{M}\langle pc \to mnem\rangle \mapsto \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ correspond à une transition entre l'état mémoire $\mathcal{M}$ dans lequel le pointeur d'instruction pc pointe vers l'opération $mnem$ et un nouvel état $\mathcal{M}_i$ pour $1 \leq i \leq n$. $\mathcal{M}\langle loc\texttt{:=}v\rangle$ représente l'état mémoire $\mathcal{M}$ dans lequel la valeur v est affectée à l'emplacement mémoire loc.

Les opérandes des instructions sont des modes d'adressage dont la sémantique abstraite est définie ci-dessous. Les opérandes sources et destinations sont traitées différemment. La sémantique $[\![a]\!]_s$ d'une source est :

$$[\![v]\!]_s = v \qquad [\![R_i]\!]_s = \rho(R_i) \tag{5.3}$$

$$[\![*v]\!]_s = \bigcup_{\ell \in \gamma_{\mathbb{D}_*}(v)} \mu(\ell) \qquad [\![R_i(R_j)]\!]_s = \bigcup_{\ell \in \gamma_{\mathbb{D}_*}\left(\rho(R_i) +_{\mathtt{int}} \rho(R_j)\right)} \mu(\ell) \tag{5.4}$$

Dans les modes direct et indirect, l'argument est évalué, ce qui produit une étiquette abstraite $\ell^\sharp \in \mathbb{D}_*$ et l'union des valeurs stockées aux emplacements ℓ, pour $\ell \in \gamma_{\mathbb{D}_*}(\ell^\sharp)$, est calculée. $+_{\mathtt{int}}$ est défini dans les équations (5.8) et (5.9). Pour une opérande utilisée en destination, nous avons :

$$[\![v]\!]_d = \text{erreur} \qquad [\![R_i]\!]_d = \{R_i\} \tag{5.5}$$

$$[\![*v]\!]_d = \begin{cases} \left\{\mu(x) \ : \ x \in \gamma_{\mathbb{D}_*}(v)\right\} \text{ si } v \in \mathbb{D}_* \\ \text{erreur sinon} \end{cases} \tag{5.6}$$

$$[\![R_i(R_j)]\!]_d = \left\{\mu(x) \ : \ x \in \gamma_{\mathbb{D}_*}([\![R_i]\!]_s +_{\mathtt{int}} [\![R_j]\!]_s)\right\} \tag{5.7}$$

L'évaluation d'une opérande destination produit un ensemble d'emplacements mémoire ou de registres. $\mathcal{M}\langle [\![m]\!]_d :=v\rangle$ est l'abréviation de $\mathcal{M}\langle d_1 :=v, \ldots, d_n \ :=v\rangle$, pour $d_1, \ldots, d_n \in [\![m]\!]_d$. Dans les modes direct et indirect, l'ensemble des étiquettes étant fini, $\gamma_{\mathbb{D}_*}(m)$ représente toujours un ensemble fini d'emplacements.

La première règle de la figure 5.2 est valable pour toutes les opérations sur les entiers, c.à.d. `ADDI`, `SUBI` et `MULI`. Afin de gérer l'arithmétique des pointeurs, nous autorisons quelques coercions entre les valeurs de $\mathbb{I}$, $\mathbb{D}_*$ et $\mathbb{T}_*$.

$$i_1 \pm_{\mathtt{int}} i_2 = i_1 \pm i_2 \quad \text{si } i_1 \in \mathbb{I} \text{ et } i_2 \in \mathbb{I} \tag{5.8}$$

$$d \pm_{\mathtt{int}} i = (L, I) \in \mathbb{D}_* \quad \text{si } d = (L, i') \in \mathbb{D}_*, \ \ i \in \mathbb{I} \text{ et } I = i \pm i' \tag{5.9}$$

Une règle de coercion similaire à celle de l'équation (5.9) existe aussi pour les pointeurs de code. L'interprétation abstraite d'un branchement inconditionnel `BR` produit un ensemble $S = \left\{\mathcal{M}\langle pc :=pc'\rangle \ : \ pc' \in \gamma_{\mathbb{T}_*}([\![src]\!]_s)\right\}$ d'états mémoire. `BLT` et `BEQ` se comportent comme `BR` lorsque le drapeau correspondant vaut `true`. La règle suivante de la figure 5.2 concerne `CMPI`. Pour cette instruction, lorsque src_1 est comparée à src_2, quatre nouveaux états mémoire combinant les valeurs possibles des drapeaux sont générés. $\mathcal{M}_{|src_1 \leq src_2}$ représente l'état mémoire $\mathcal{M}$ dans lequel les valeurs des opérandes ont été rétrécies aux seuls cas rendant la condition vraie.

Le domaine abstrait que nous utilisons pour la pile a été conçu afin de pouvoir analyser précisément les boucles dans lesquelles les mêmes séquences de `PUSH` et de `POP` sont réalisées à chaque itération. On suppose aussi qu'une opération de type `PUSH` empile toujours des valeurs du même type (sinon on perd en précision).

La pile abstraite est modélisée par un graphe orienté dont les nœuds correspondent aux opérations `PUSH` et `CALL`. Intuitivement, un arc (n_1, n_2) indique que le `PUSH` ou `CALL` correspondant au nœud n_1 peut éventuellement précéder celui correspondant au nœud n_2 dans une trace d'exécution. La valeur abstraite attachée à un nœud n sur-approxime les valeurs abstraites empilées en ce point de contrôle. Finalement, notre pile abstraite contient aussi un ensemble de sommets de pile.

Formellement, une pile abstraite σ est une paire (G, τ) où $G = (V, E, \omega)$ est un graphe orienté et τ un ensemble de nœuds. V contient un nœud par `PUSH` et `CALL`. $E \subseteq V \times V$ est l'ensemble des arcs, $\omega \ : \ V \to \mathbb{D}^\sharp$ associe à chaque nœud une valeur abstraite et $\tau \subseteq V$ est l'ensemble des sommets de pile. Initialement, $\omega(v) = (\bot_{\mathbb{D}^\sharp}, \emptyset)$ pour tout $v \in V$ et $\tau = \emptyset$. La sémantique de `PUSH`, `POP` et `CALL` est donnée à la figure 5.2. Pour `CALL`, $T(pc + 1)$ représente la valeur de $\mathbb{T}_*$ pointant sur la ligne $pc + 1$ dans le segment du programme. Lorsque `PUSH` src apparaît à la

$$\mathcal{M}\langle pc \to \texttt{ADDI}\ src_1, src_2, dst\rangle \mapsto \left\{\mathcal{M}\langle [\![dst]\!]_d := [\![src_1]\!]_s +_{\texttt{int}} [\![src_2]\!]_s,\ pc\ := pc+1\rangle\right\}$$

$$\mathcal{M}\langle pc \to \texttt{ADDF}\ src_1, src_2, dst\rangle \mapsto \left\{\mathcal{M}\langle [\![dst]\!]_d := [\![src_1]\!]_s +_{\texttt{float}} [\![src_2]\!]_s,\ pc\ := pc+1\rangle\right\}$$

$$\mathcal{M}\langle pc \to \texttt{MV}\ src, dst\rangle \mapsto \left\{\mathcal{M}\langle [\![dst]\!]_d := [\![src]\!]_s,\ pc\ := pc+1\rangle\right\}$$

$$\mathcal{M}\langle pc \to \texttt{BR}\ src\rangle \mapsto \left\{\mathcal{M}\langle pc := pc'\rangle\ :\ pc' \in \gamma_{\mathbb{T}_*}([\![src]\!]_s)\right\}$$

$$\mathcal{M}\langle pc \to \texttt{BLT}\ src\rangle \mapsto \left\{\mathcal{M}\langle pc := pc'\rangle\ :\ pc' \in \gamma_{\mathbb{T}_*}([\![src]\!]_s)\right\} \text{ if } \varphi(\texttt{LT}) \sqsupseteq \texttt{true}$$

$$\mathcal{M}\langle pc \to \texttt{BLT}\ src\rangle \mapsto \left\{\mathcal{M}\langle pc := pc+1\rangle\right\} \text{ if } \varphi(\texttt{LT}) \sqsupseteq \texttt{false}$$

$$\begin{array}{c}\mathcal{M}\langle pc \to \texttt{CMPI}\ src_1, src_2\rangle \mapsto \\ \big\{\mathcal{M}_{|src_1 \le src_2}\langle \varphi(\texttt{LT}) := \texttt{true}, \varphi(\texttt{EQ})\ := \texttt{true}, pc\ := pc+1\rangle, \\ \mathcal{M}_{|src_1 < src_2}\langle \varphi(\texttt{LT}) := \texttt{true}, \varphi(\texttt{EQ})\ := \texttt{false}, pc\ := pc+1\rangle, \\ \mathcal{M}_{|src_1 = src_2}\langle \varphi(\texttt{LT}) := \texttt{false}, \varphi(\texttt{EQ})\ := \texttt{true}, pc\ := pc+1\rangle, \\ \mathcal{M}_{|src_1 > src_2}\langle \varphi(\texttt{LT}) := \texttt{false}, \varphi(\texttt{EQ})\ := \texttt{false}, pc\ := pc+1\rangle\big\}\end{array}$$

$$\begin{array}{c}\mathcal{M}\langle pc \to \texttt{PUSH}\ src\rangle \mapsto \\ \mathcal{M}\langle \omega(pc) := \omega(pc) \cup [\![src]\!]_s,\ \forall t \in \tau\ :\ E\ := E \cup (pc,t), \tau\ := \{pc\},\ pc\ := pc+1\rangle\end{array}$$

$$\begin{array}{c}\mathcal{M}\langle pc \to \texttt{POP}\ dst\rangle \mapsto \\ \mathcal{M}\langle [\![dst]\!]_d := \bigcup_{t\in\tau} \omega(t),\ \tau\ := \{t'\ :\ (t,t') \in V \wedge t \in \tau\},\ pc\ := pc+1\rangle\end{array}$$

$$\begin{array}{c}\mathcal{M}\langle pc \to \texttt{CALL}\ src\rangle \mapsto \\ \mathcal{M}\langle \omega(pc) := \omega(pc) \cup T(pc+1),\ \forall t \in \tau\ :\ E\ := E \cup (pc,t), \tau\ := \{pc\},\ pc\ := \gamma_{\mathbb{T}_*}([\![src]\!]_s)\rangle\end{array}$$

FIG. 5.2 – *Sémantique abstraite du langage.*

i-ième ligne du programme, l'argument est évalué, ce qui donne $v = [\![src]\!]_s$, et v est uni à la valeur déjà présente dans $\omega(i)$. En ce qui concerne les sommets de pile, les éléments courants de τ sont ajoutés aux successeurs de i, et τ devient le singleton $\{i\}$. Pour `POP` dst, l'argument est évalué par $[\![dst]\!]_d$ et l'union des valeurs abstraites associées aux sommets de pile potentiels est affectée aux emplacements mémoire donnés par $[\![dst]\!]_d$. Le nouveau sommet de pile abstrait est l'ensemble des successeurs des sommets actuels. Enfin, la sémantique de `CALL` empile la valeur de retour et effectue le branchement. L'union de deux piles $\sigma_1 = (G_1, \tau_1)$ et $\sigma_2 = (G_2, \tau_2)$ est définie par :

$$(G_1, \tau_1) \cup (G_2, \tau_2) = (G, \tau_1 \cup \tau_2)$$

où, dans G, $\forall v \in V,\ \omega(v) = \omega_1(v) \cup \omega_2(v)$ et $E = E_1 \cup E_2$. Cette pile abstraite permet en particulier d'analyser précisément des procédures récursives.

Nous décrivons maintenant comment les élargissements sont effectués et comment les environnements abstraits sont stockés dans l'analyseur. Pour chaque ligne de code i, on suppose que $\mathcal{E}(i)$ mémorise l'environnement pour lequel l'instruction correspondante a été analysée. L'analyse consiste alors à analyser une ligne de code et à mettre à jour $\mathcal{E}$ ainsi qu'une liste de tâches conte-

nant les configurations qui n'ont pas encore été analysées. Ce processus est répété jusqu'à ce que la liste de tâches devienne vide. Une configuration n'est pas analysée à nouveau si elle a déjà été analysée pour un environnement plus grand. Pour décider quand effectuer un élargissement, un compteur est associé à chaque branchement. $B(i)$, le compteur du branchement de la ligne i, est incrémenté à chaque fois que la ligne i est analysée. Un élargissement est déclenché lorsque $B(i)$ dépasse un certain seuil et, dans ce cas, $B(i)$ est réinitialisé à zéro.

Un élargissement $\mathcal{M}_1 \nabla \mathcal{M}_2$ consiste à élargir séparément chaque registre, emplacement mémoire et drapeau. Nous décrirons ultérieurement l'élargissement de la pile. Pour les valeurs élémentaires, les intervalles d'entiers sont élargis de façon classique et pour les pointeurs aucun élargissement n'est effectué : soit $d_1, d_2 \in \mathbb{D}_*$ et $t_1, t_2 \in \mathbb{T}_*$. $d_1 \nabla d_2 = d_1 \cup d_2$ et $t_1 \nabla t_2 = t_1 \cup t_2$. Remarquons que, les ensembles d'emplacements mémoire dans les segments code et données étant finis, cela n'empêche pas l'analyse de terminer.

Soient $\sigma = (G, \tau)$, $\sigma_1 = (G_1, \tau_1)$ et $\sigma_2 = (G_2, \tau_2)$ trois piles telles que $\sigma = \sigma_1 \nabla \sigma_2$, $G_1 = (V_1, E_1, \omega_1)$ et $G_2 = (V_2, E_2, \omega_2)$. Alors σ est défini par $\tau = \tau_1 \cup \tau_2$, $E = E_1 \cup E_2$ et $\forall v \in V$, $\omega(v) = \omega_1(v) \nabla \omega_2(v)$. La pile est élargie nœud par nœud et les ensembles de sommets potentiels sont unis.

Le dernier point que nous souhaitons préciser concerne la gestion des environnements $\mathcal{E}(i)$ contenant les états mémoire de la i-ième ligne de code. Intuitivement, il est trop imprécis de ne mémoriser, pour une ligne i, qu'un seul état mémoire $\mathcal{M}$. Cela est dû au fait que la sémantique abstraite des opérateurs de comparaison `CMPI` et `CMPF` génère plusieurs environnements dans lesquels les valeurs des drapeaux sont différents et dans lesquels les valeurs des opérandes ont été rétrécies selon les résultats de la comparaison. Par exemple, supposons qu'une comparaison soit effectuée à la i-ième ligne du programme. Si $\mathcal{E}(i+1)$ ne contient qu'un seul état mémoire alors tous les états générés par la comparaison de la ligne i seront immédiatement fusionnés, ce qui rendra l'analyse particulièrement imprécise. Aussi l'analyseur conserve-t-il séparément les états mémoires pour lesquels une instruction a été analysée, selon les valeurs des drapeaux. Autrement dit, $\mathcal{E}(i, b_{\mathtt{LT}}, b_{\mathtt{EQ}})$ mémorise les états $\mathcal{M}$ pour lesquels la ligne i a été analysée et tels que, dans $\mathcal{M}$, $\varphi(\mathtt{LT}) = b_{\mathtt{LT}}$ et $\varphi(\mathtt{EQ}) = b_{\mathtt{EQ}}$. Ainsi, quatre environnements sont stockés par ligne de code, notre langage utilisant deux drapeaux. Cette technique est nécessaire pour conserver toutes les informations produites par les tests.

Nous concluons cette section en montrant comment une procédure récursive simple est comprise par notre analyse. On s'intéresse à la fonction `f` définie par :

$$\begin{cases} \mathtt{f}(0) = 0 \\ \mathtt{f}(n) = n - 1 \text{ si } n > 0 \end{cases} \tag{5.10}$$

Dans l'implémentation donnée ci-dessous, le programme principal calcule $\mathtt{f}(500)$.

```
 |1| f:     POP R1      ; beginning of f. Return address in R1
 |2|        POP R0      ; R0 contains the argument
 |3|        CMPI R0,0   ; if R0<>0 then
 |4|        BEQ endf    ;
 |5|        SUBI 1,R0   ;     R0:=R0-1
 |6|        PUSH R1     ;     push return address
 |7|        PUSH R0     ;     push new argument
 |8|        CALL f      ;     recursive call to f
 |9|        POP R1      ;     retrieve the return address
|10| endf:  BR R1       ; return to the caller
```

```
|11| main:  MV 500,R0   ; beginning of main program
|12|        PUSH R0     ; the argument of f is pushed onto the stack
|13|        CALL f      ; call to the recursive function f
|14|        .end
```

Les valeurs finales que l'on obtient pour les registres R_0 et R_1 à chaque ligne de code sont données à la figure 5.3. Examinons tout d'abord R_0. Nous pouvons constater que, dans le programme principal, R_0 vaut exactement $[500, 500]$ avant l'appel de `f` et $[0, 0]$ après l'appel. L'analyseur a détecté que, lorsque le branchement de la ligne 4 est pris, R_0 doit valoir zéro, même si, dans la fonction `f`, R_0 vaut $[0, 500]$.

ligne	R_0	R_1	ligne	R_0	R_1
1	[0,500]	$(\{\|9\|, \|14\|\}, [0, 0])$	8	[0,499]	$(\{\|9\|, \|14\|\}, [0, 0])$
2	[0,500]	$(\{\|9\|, \|14\|\}, [0, 0])$	9	[0,0]	$(\{\|9\|, \|14\|\}, [0, 0])$
3	[0,500]	$(\{\|9\|, \|14\|\}, [0, 0])$	10	[0,0]	$(\{\|9\|, \|14\|\}, [0, 0])$
4	[0,500]	$(\{\|9\|, \|14\|\}, [0, 0])$	11	$\bot_{\mathbb{D}^\sharp}$	$\bot_{\mathbb{D}^\sharp}$
5	[1,500]	$(\{\|9\|, \|14\|\}, [0, 0])$	12	[500,500]	$\bot_{\mathbb{D}^\sharp}$
6	[0,499]	$(\{\|9\|, \|14\|\}, [0, 0])$	13	[500,500]	$\bot_{\mathbb{D}^\sharp}$
7	[0,499]	$(\{\|9\|, \|14\|\}, [0, 0])$	14	[0,0]	$(\{\|9\|, \|14\|\}, [0, 0])$

FIG. 5.3 – *Résultats de l'analyse de l'exemple de la section 5.2.1. Les valeurs finales des registres R_0 et R_1 sont données pour différentes lignes de code.*

Les valeurs de R_1 sont des pointeurs dans le segment du code (domaine $\mathbb{T}_*$). Dans la boucle, R_1 égale $(\{|9|, |14|\}, [0, 0])$ ce qui représente des pointeurs vers la 9-ième ou la 14-ième ligne du programme. Cette valeur a été donnée par l'opération `POP` de la ligne $|1|$. A la première itération, R_1 est évalué à $(\{|14|\}, [0, 0])$ ce qui correspond à l'adresse de retour empilée par le `CALL` de la ligne 13. Ensuite, les appels récursifs de la ligne 8 empilent $(\{|9|\}, [0, 0])$ et cette valeur est unie à celle déjà présente dans R_1. Les valeurs abstraites de la pile sont données à la figure 5.4. Le carré étiquetée i, j donne la valeur de la pile à la j-ième analyse de la ligne i.

5.2.2 Description de l'analyseur

L'analyseur, écrit en OCaml, traite des programmes écrits dans le langage assembleur du processeur TMS320 C3X [Tex97, BV96], un DSP très utilisé pour les systèmes embarqués. L'outil utilise l'interprétation abstraite décrite à la section 5.2.1 et utilise, pour les opérations en nombres flottants, la sémantique des séries d'erreurs vue au chapitre 3 dans laquelle les coefficients f et ω^ℓ sont respectivement abstraits par des intervalles de flottants TMS320 C3X et de flottants multi-précision fournis par la librairie MPFR [HLFZ01], conformément à l'abstraction présentée à la section 4.2.

L'outil détecte les erreurs à l'exécution mentionnées dans la section 5.2.1 mais l'interface est plus particulièrement conçue pour la validation de la précision numérique des calculs en nombres flottants et pour mettre en évidence les principales sources d'erreurs survenant dans ces traitements.

Outre les erreurs d'arrondi introduites par les opérations élémentaires, des termes d'erreurs sont aussi générés par les opérations de recopie de valeurs des registres vers la mémoire. En effet,

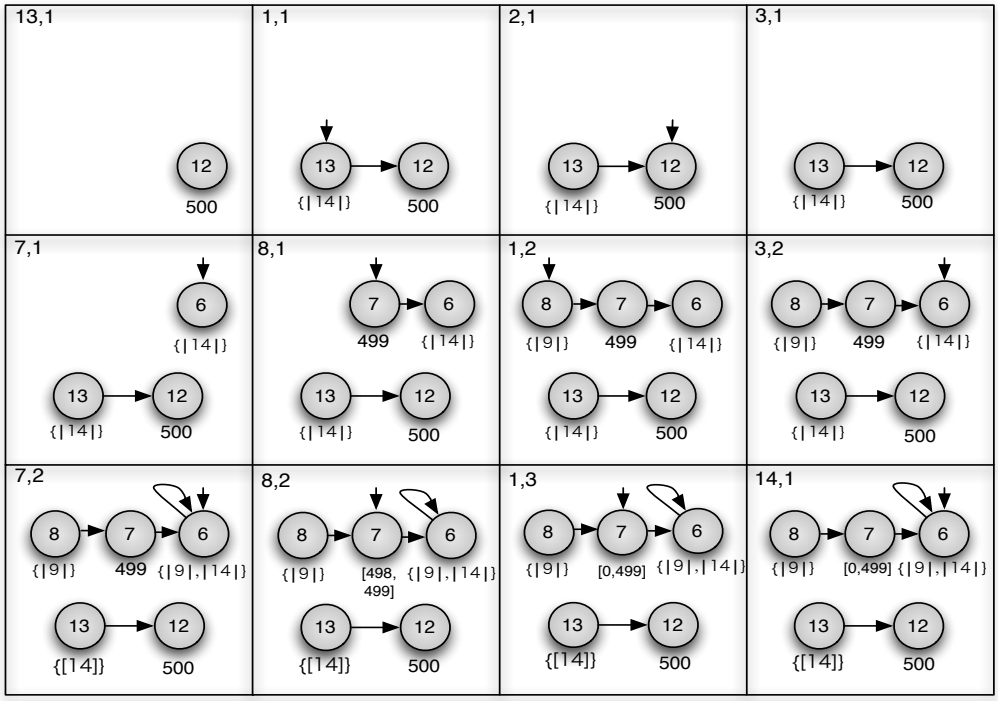

FIG. 5.4 – *Valeur abstraite de la pile pendant l'analyse du programme de la section 5.2.1.*

dans le TMS320, les nombres flottants étant représentés sur 40 bits dans les registres et sur 32 en mémoire, les instructions `STF` et `PUSHF` génèrent des pertes de précision. Par ailleurs, l'utilisateur peut aussi écrire des assertions dans le code pour spécifier les entrées, en utilisant des intervalles pour les flottants et les erreurs qui leurs sont associées. On peut ainsi associer une valeur abstraite initiale à un registre ou à un emplacement mémoire. Par exemple, l'assertion `.float x [0.0,1.0,0.05,0.1]` indique que l'emplacement mémoire `x` est initialisé avec une valeur comprise entre 0 et 1.0 et que l'erreur attachée à cette valeur varie entre 0.05 et 0.1.

Une fois le programme analysé, l'interface graphique de l'analyseur ouvre trois fenêtres que l'on peut voir à la figure 5.5. La fenêtre en bas à gauche affiche une synthèse des résultats de l'analyse. Un carré coloré est dessinée au début de chaque ligne de code pour indiquer l'intensité de l'imprécision numérique la plus importante à cette ligne. Plus précisément, l'analyseur recherche la valeur la plus erronée dans l'état mémoire abstrait correspondant à la ligne de code courante et dessine un carré proportionnellement intense. Une intensité faible correspond à une erreur peu importante (un carré blanc invisible correspond à pas d'erreur du tout) tandis qu'un carré sombre correspond à une erreur très significative. Cette technique permet à l'utilisateur de repérer aisément les principales imprécisions.

La fenêtre la plus à droite de la figure 5.5 affiche les valeurs abstraites des registres et des emplacements mémoire à chaque ligne de code. Pour les valeurs flottantes, deux intervalles sont donnés, un pour le flottant et un pour l'erreur globale (voir section 2.3). De plus l'utilisateur peut ouvrir une nouvelle fenêtre, telle que celle affichée en haut à gauche de la figure 5.5. Cette fenêtre représente par un histogramme la série d'erreurs correspondant à la valeur choisie. En face de chaque ligne de code ℓ une barre proportionnelle au terme d'erreur ω^ℓ de la série est dessi-

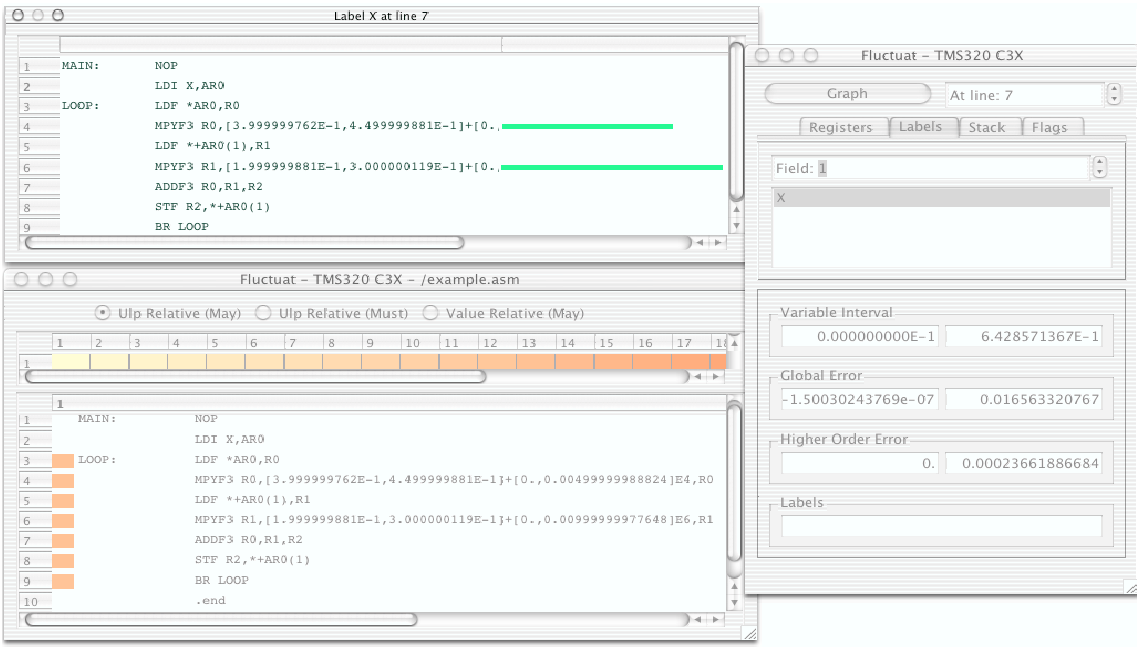

FIG. 5.5 – *Capture d'écran montrant l'interface de l'analyseur Fluctuat pour assembleur.*

née. On peut ainsi voir aisément quels termes d'erreurs contribuent le plus significativement à l'erreur globale sur le résultat d'un calcul. Un programmeur voit ainsi directement quelles opérations doivent être revues pour améliorer la précision. Par exemple, si la perte de précision due à une recopie d'une valeur d'un registre vers la mémoire crée une trop grande imprécision, on peut éventuellement décider de la conserver dans les registres jusqu'à la fin du calcul.

Le programme analysé dans la figure 5.5 implémente un filtre récursif du premier ordre

$$y_n = ax_n + by_{n-1}$$

Les coefficients a et b valent $[0.4, 0.45, 0.0, 0.005]$ et $[0.2, 0.3, 0.0, 0.01]$ ce qui signifie que l'on considère une famille de filtres dont les coefficients sont imprécis. x_n et y_{n-1} sont stockés dans un tableau à deux éléments `x`. Une assertion indique que le signal d'entrée x_n est borné par $0.5 \leq x_n \leq 1.0$ pour tout n. y_n est calculé itérativement, dans une boucle infinie. Les fenêtres à droite et en haut de la figure 5.5 affichent des résultats d'analyse concernant y et stockés en mémoire dans `x[1]`. Nous pouvons voir dans la fenêtre de droite que la sortie reste comprise entre $[0.0, 0.6428571367]$ et que l'erreur attachée ne dépasse jamais 0.016563320767, indépendamment du nombre d'itérations. L'erreur n'est donc jamais beaucoup plus importante que les erreurs initiales sur les coefficients. La fenêtre en haut à gauche nous donne les sources de cette erreur. Comme l'on pouvait s'y attendre dans cet exemple, les principales erreurs proviennent des multiplications dont les opérandes sont imprécises. Nous pouvons voir de plus que le second produit génère une erreur plus importante que le premier.

5.3 Fluctuat pour le langage C

Tout comme l'analyseur d'assembleur, Fluctuat C permet de calculer par interprétation abstraite la propagation des erreurs d'arrondi dans des programmes effectuant des calculs en nombres flottants [GMP02, PGM04]. Il est ainsi possible d'identifier les principales sources d'imprécision. Cet outil a été conçu pour pouvoir traiter des codes industriels de grande taille, notamment des logiciels embarqués qui effectuent en général des calculs plus simples que ceux que l'on trouve dans des codes de calcul intensif. De récents travaux de recherche permettent toutefois d'espérer qu'il sera bientôt possible d'analyser précisément des algorithmes plus numériques [GP05]. Comparé à d'autres outils fondés sur les techniques que nous avons vu au chapitre 2 (arithmétique stochastique et différentiation automatique), Fluctuat présente l'avantage de ne négliger aucun terme d'erreur, ce qui est important pour la validation d'applications critiques.

Fluctuat C utilise la sémantique des séries d'erreurs à l'ordre un, définie au chapitre 3. Comme le montre la figure 5.6, la fenêtre principale de l'analyseur affiche le code du programme analysé, la liste des variables apparaissant dans l'environnement abstrait à la fin de l'analyse, ainsi qu'un histogramme représentant la valeur abstraite de la variable sélectionnée. Des barres de défilement autour de l'histogramme permettent d'effectuer différents types de zooms.

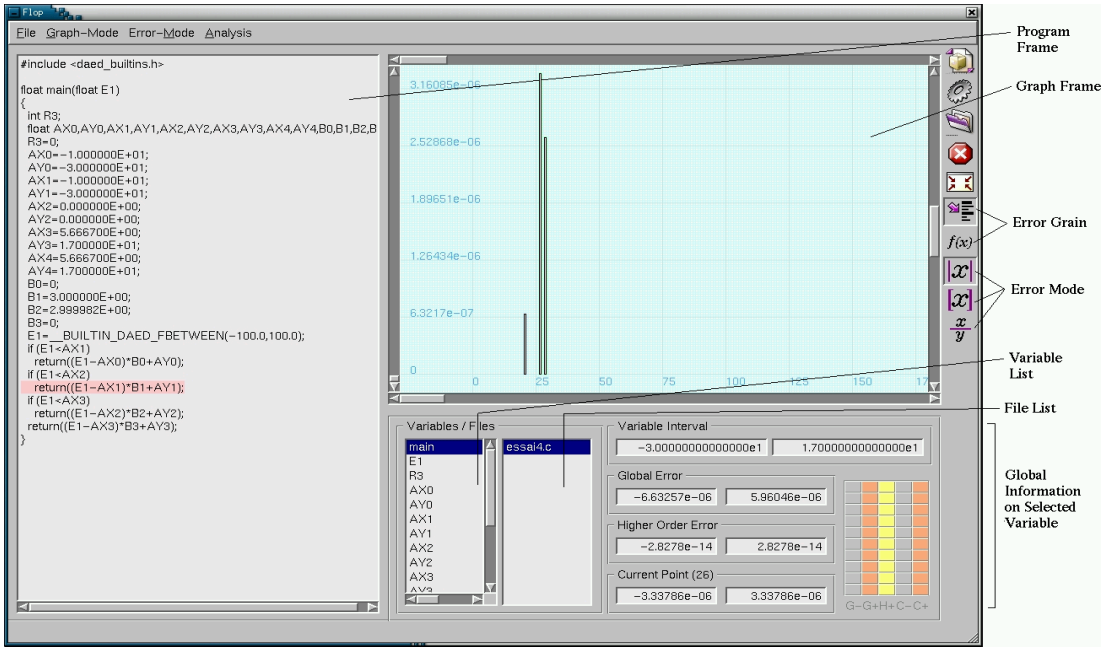

FIG. 5.6 – *Fenêtre principale de l'analyseur Fluctuat pour langage C.*

Le graphe représente la série d'erreur d'une variable v et, par conséquent, donne la contribution des erreurs créées à chaque ligne de code, à l'erreur globale sur v. Les opérations sont identifiées par leur ligne de code qui correspond à l'axe des abscisses.

Dans la figure 5.6, les barres de l'histogramme indiquent la valeur absolue maximale des bornes des intervalles correspondant aux coefficients des séries d'erreurs. Lorsque celles-ci sont

suffisant faibles, le programme peut être considéré comme étant correct vis-à-vis de la précision numérique. Dans le cas contraire, les barres les plus hautes permettent d'identifier aisément les principales sources d'erreurs.

Ce type de graphe est bien adapté pour détecter certaines erreurs, comme les absorptions, mais d'autres, comme les éliminations catastrophiques sont plus facilement détectables à partir d'erreurs relatives. Cependant, comme cela a été discuté à la section 3.6, on ne dispose pas, à l'heure actuelle, d'une sémantique des séries d'erreurs relatives.

L'histogramme et la fenêtre contenant le code du programme sont connectés dans l'interface graphique de telle sorte que lorsque l'on clique sur une barre, la ligne de programme correspondante s'illumine. Cela permet d'identifier rapidement les instructions responsables des pertes de précision sur la variable sélectionnée.

Dans l'exemple de la figure 5.6, un extrait de code typique des logiciels embarqués a été analysé. Il s'agit d'une fonction d'interpolation par paliers. On peut voir, à partir de l'histogramme, que les seules sources d'imprécision sur le résultat final de la fonction principale sont :

- l'erreur d'arrondi due aux nombres flottants sur la constante 2.999982 à la ligne 20 (on retrouve directement cette ligne en cliquant sur la barre de l'histogramme), qui est négligeable ;
- le second `return` de la ligne 26 (la seconde barre du graphe) ;
- le troisième `return` de la ligne 28 (troisième et dernière barre du graphe d'erreur).

Ces deux dernières erreurs sont les plus importantes. Cela s'explique par le fait que nous avons utilisé une assertion `__BUILTIN_DAED_FBETWEEN` pour imposer, durant l'analyse, que `E1` soit considérée à l'entrée de la fonction comme une valeur comprise entre -100 et 100. Ainsi l'analyseur détecte que la fonction n'exécute pas le premier `return`. Il détecte ensuite que la fonction peut exécuter le quatrième et dernier `return`, mais un des termes de la multiplication vaut zéro et la constante renvoyée est exacte. Aussi n'y a-t-il pas de perte de précision dans ce cas. L'utilisateur peut déduire de ces observations que s'il souhaite améliorer la précision de ce calcul, il doit réduire les erreurs commises dans le second et dans le troisième `return`. Une façon simple de résoudre ce problème est d'augmenter la précision des soustractions dans ces deux expressions (en utilisant `double E1` par exemple). En revanche, améliorer la précision de la constante 2.999982 n'apporte pas de gain substantiel. Remarquons aussi que l'analyseur calcule que l'erreur d'ordre supérieur est toujours négligeable pour notre exemple.

Fluctuat C traite l'essentiel du langage ANSI C, effectue une analyse d'alias, traite les tableaux et les structures. Il effectue une analyse interprocédurale utilisant une technique de partitionnement simple [JM82]. Fluctuat C a été construit à partir d'un autre analyseur [GGP+01] développé au CEA. L'intervalle correspondant aux flottants utilise des nombres flottants classiques et une précision plus importante est utilisée pour les termes d'erreur. On utilise la librairie MPFR [HLFZ01], fondée sur GNU MP et qui offre des fonctionnalités supplémentaires, en particulier en ce qui concerne la gestion des modes d'arrondi.

5.4 Perspectives

Concernant l'analyse de code assembleur, le langage que nous avons utilisé à la section 5.2.1 est représentatif de nombreux processeurs mais devrait être adapté dans certain cas. La principale modification peut concerner la gestion de la pile. En effet, nous avons supposé que les accès à celle-ci s'effectuaient à l'aide des instructions `PUSH` et `POP`, comme dans le TMS320. Certains

processeurs, tels que les Sparc ou 680X0 ne fournissent pas ces opérations et laissent le programmeur gérer directement le pointeur de pile SP. Notre pile abstraite devrait être modifiée dans ce cas. Une solution pourrait consister à affecter un intervalle d'entiers à SP et à utiliser un graphe semblable à celui de la section 5.2.1 dans lequel les noeuds correspondraient aux niveaux de pile, c.à.d. aux valeurs entières pouvant être prises par SP.

Par ailleurs, dans notre jeu d'instructions, les opérations de comparaison et de branchement sont disjointes. Certains processeurs possèdent des instructions atomiques pour les branchements conditionnels. Cela simplifie l'analyse et les environnements $\mathcal{E}$ introduits à la section 5.2.1 peuvent être simplifiés (on ne garde qu'un seul état mémoire par ligne de code).

Plus généralement, les analyseurs Fluctuat sont des outils industriellement utilisables mais faisant l'objet de nouveaux développements réguliers. Les analyses mentionnées au chapitre 4 constituent autant de perspectives pour ces outils. D'autres améliorations récentes sont aussi susceptibles d'ouvrir de nouveaux champs d'applications pour l'analyseur de langage C. Par exemple, l'analyse des pertes de précision dues au passage d'une arithmétique à virgule flottante vers une arithmétique à virgule fixe permet d'envisager l'utilisation de Fluctuat dans le domaine des applications embarquées effectuant des traitements d'images ou de sons (téléphones mobiles, etc.) [Mas05].

Chapitre 6

Conclusion et perspectives

6.1 Synthèse

Nous avons vu au cours des chapitres 2 à 5 que les différents travaux présentés dans ce document méritaient tous des approfondissements importants que nous allons brièvement rappeler.

En étudiant les sémantiques dynamiques classiques pour la précision numérique, au chapitre 2, nous avons vu qu'il serait utile de définir de nouvelles arithmétiques, plus précises dans leurs estimations des erreurs et plus pertinentes en terme d'intérêt des propriétés calculées, par exemple par combinaison celles que nous avons présentées. Cette étape est importante car chacune de ces arithmétiques permet de calculer une propriété bien particulière qui doit être judicieusement choisie avant d'élaborer une analyse statique se fondant sur elle. Nous reviendrons sur les critères de sûreté pour la précision numérique à la section 6.2.

Les séries d'erreurs, présentées au chapitre 3, constituent la base principale de nos travaux sur la précision numérique. En identifiant les principales sources d'erreurs responsables d'une perte de précision sur le résultat d'un calcul, cette sémantique fournit au programmeur des informations très utiles pour améliorer la qualité des calculs. Il serait intéressant de développer d'autres sémantiques, sur le même principe, par exemple pour identifier les opérations qui font le plus croître les termes d'erreurs, indépendamment de leur valeur absolue, ou de définir une sémantique des séries d'erreurs relatives. Les séries d'erreurs sont par ailleurs à la base des travaux sur l'analyse statique présentés au chapitre 4 et des analyseurs Fluctuat décrits au chapitre 5.

Nous avons vu, au chapitre 4, que l'analyse statique par interprétation abstraite de la sémantique des séries d'erreurs constituait un thème de recherche particulièrement riche. La définition d'heuristiques efficaces pour le problème NP-complet du partitionnement dynamique des points de contrôle pourrait avoir des conséquences pratiques importantes en réduisant sensiblement les temps de calcul et l'espace mémoire utilisés par les analyseurs. Le calcul d'exposants de Lyapunov abstraits, pour l'étude de la stabilité des calculs peut avoir des prolongements intéressants en ce qui concerne les systèmes hybrides. Plus généralement, d'autres analyses fondées sur les outils mathématiques développés pour l'étude des systèmes dynamiques pourraient ouvrir de nouvelles perspectives, comme par exemple, le calcul de bifurcations ou de bassins d'attraction abstraits. Enfin, l'analyse statique de systèmes hybrides discrets-continus constitue un axe de recherche à long terme, tant en ce qui concerne l'analyse statique proprement dite que sur le thème de la résolution sûre de problème numériques : la méthode de Runge-Kutta sûre présentée à la section 4.5 montre qu'il est possible de résoudre des problèmes numériques précisément et en certifiant les

solutions apportées. Il ouvre la voie à de nombreux autres travaux dans cette direction.

Les analyseurs Fluctuat, pour le langage C et pour l'assembleur du processeur TMS320 sont régulièrement enrichis par de nouveaux développements. Bien qu'étant utilisés pour des cas d'études industriels, par exemple, par Airbus, Hispano-Suiza ou encore l'IRSN, ils restent perfectibles. L'intégration de nouvelles analyses dans ces logiciels permet d'une par d'améliorer leurs performances et, d'autre part, de vérifier l'utilité pratique de nouveaux résultats théoriques. Le développement de ces analyseurs a aussi inspiré d'autres recherches en analyse statique, ne portant pas directement sur la précision numérique, comme les travaux sur le partitionnement dynamique des points de contrôle [Mar03] ou sur les stratégies d'itération sur les politiques [CGG$^+$05].

Nous pouvons ainsi dire, en guise de conclusion, que l'analyse statique pour la précision numérique est un domaine encore jeune, méritant d'être enrichi de nombreux développements supplémentaires, malgré l'existence d'outils déjà utilisables industriellement.

6.2 Sûreté des traitements numériques

La validation de la précision de traitements numériques, pour des questions de sûreté, est une problématique récente, voire naissante, qui n'a certainement pas encore atteint un niveau de maturité suffisant. Indépendamment des techniques utilisées pour les calculer, il n'existe pas à ce jour de consensus autour de critères de correction, ou, autrement dit, autour de propriétés permettant d'affirmer qu'un programme qui s'y conforme est sûr. Nous avons pu entrevoir par ailleurs, au cours du chapitre 2, les subtilités des propriétés les plus couramment calculées.

L'ensemble des travaux présentés dans ce document repose sur une hypothèse : un programme peut être vu à la fois comme un modèle et comme une implémentation. Plus précisément, on suppose qu'un programme implante correctement un modèle réel et l'on étudie les erreurs introduites par l'arithmétique flottante par rapport à ce modèle. Implicitement, on suppose aussi que lorsqu'il développe une application, un programmeur raisonne en utilisant l'arithmétique réelle, qui nous est familière, et non pas l'arithmétique flottante, non-intuitive.

L'estimation des erreurs introduites par l'arithmétique flottante n'est cependant pas un critère de correction absolu : une interprétation des résultats est nécessaire pour déterminer si les erreurs ainsi détectées sont acceptables ou pas. D'autres critères pourraient aussi être envisagés, par exemple à partir du calcul d'erreurs relatives ou d'erreurs inverses (voir la section 2.2) bien que ces dernières soient plus difficiles à calculer par analyse statique. Cependant, ils ne supprimerait pas, a priori, l'étape d'interprétation manuelle des résultats.

Nous avons proposé, à la section 4.4, d'autres propriétés de sûreté qui pallient les inconvénients des critères précédents en supprimant la phase d'analyse des résultats. Celles-ci reposent sur une étude des divergences de flot de contrôle entre les sémantiques réelles et flottantes. Ainsi, un programme sera considéré comme étant correct vis-à-vis de la précision numérique s'il prend les mêmes décisions dans les flottants et dans les réels, ou, autrement dit, si ses exécutions parcourent les mêmes chemins dans le graphe de flot de contrôle, indépendamment de l'arithmétique utilisée. Ce nouveau critère repose aussi sur l'hypothèse qu'un programme serait correct dans l'arithmétique réelle (pas d'erreur de méthode). Il affirme aussi que des pertes de précision même importantes sont acceptables tant qu'elles ne changent pas les décisions prises par le programme. Par exemple, le déclenchement d'une alarme doit avoir lieu dans les mêmes circonstances quelle que soit l'arithmétique utilisée et indépendamment de la précision des calculs qui n'est elle qu'anecdotique.

$$\frac{v = v_1 +_{\mathbb{R}} v_2}{v_1 + v_2 \ \rightarrow \ v} \qquad \frac{v = v_1 \times_{\mathbb{R}} v_2}{v_1 \times v_2 \ \rightarrow \ v}$$

$$\frac{e_1 \ \rightarrow \ e_1'}{e_1 + e_2 \ \rightarrow \ e_1' + e_2} \qquad \frac{e_1 \ \rightarrow \ e_1'}{e_1 \times e_2 \ \rightarrow \ e_1' \times e_2}$$

$$\frac{e \equiv e_1 \quad e_1 \ \rightarrow \ e_1' \quad e_1' \equiv e'}{e \ \rightarrow \ e'}$$

1. $(e_1 + e_2) + e_3 \equiv e_1 + (e_2 + e_3)$
2. $e_1 + e_2 \equiv e_2 + e_1$
3. $e \equiv e + 0$
4. $(e_1 \times e_2) \times e_3 \equiv e_1 \times (e_2 \times e_3)$
5. $e_1 \times e_2 \equiv e_2 \times e_1$
6. $e \equiv e \times 1$
7. $e_1 \times (e_2 + e_3) \equiv e_1 \times e_2 + e_1 \times e_3$

FIG. 6.1 – *Exemple de sémantique de référence des expressions arithmétiques pour la transformation de programmes.*

Ce critère doit cependant pouvoir être assoupli pour être utile en pratique et des divergences de flot de contrôle doivent pouvoir être acceptées dans une certaine mesure. Nous avons vu par exemple, pour l'analyse de systèmes hybrides pour lesquels nous disposons de notions d'actions et de temps, que l'on pouvait requérir que les mêmes actions soient effectuées dans un intervalle de temps acceptable, dans les deux arithmétiques.

La mise en œuvre d'analyses et d'outils permettant de valider ce type de propriétés par analyse statique représente un travail conséquent et ouvre des perspectives à moyen et long terme.

Le postulat exprimé précédemment, selon lequel un programme peut être vu comme un modèle et comme son implémentation, selon que l'on considère son exécution dans une arithmétique réelle ou flottante, nous inspire d'autres prolongements des travaux présentés dans ce documents. Tout d'abord, il serait intéressant d'effectuer des analyses semblables à celles que les outils Fluctuat effectuent pour des langages de bas niveau (langage C et assembleur, voir le chapitre 5) sur des langages de modélisation utilisés pour concevoir des systèmes embarqués (systèmes logiciels et éventuellement physiques) tels que Simulink[1] et Scade[2]. Cela permettrait la détection, au plus tôt dans le cycle de conception d'un système, de problèmes numériques éventuels mais aussi une prise en compte homogène des sous-systèmes physiques (souvent aussi modélisés dans Simulink par exemple). Cette approche pourrait aussi permettre de définir les pertes de précision acceptables dans une implémentation utilisant une arithmétique différente (si un modèle est tolérant à certaines imprécisions, une implémentation qui reste dans cette marge d'erreurs est elle aussi acceptable). Elle ouvre enfin la voie au développement de techniques de transformation d'invariants [Riv04] afin d'inférer plus simplement, sur un modèle de haut niveau, des propriétés numériques et de les vérifier sans inférence sur son implémentation.

Enfin, si l'on suppose qu'un programme fonctionnerait correctement s'il était exécuté en utilisant des nombres réels, il apparaît envisageable de le modifier automatiquement afin de produire un autre programme sémantiquement équivalent et numériquement plus précis (autrement dit, d'améliorer sa précision). Le cadre théorique proposée par P. Cousot et R. Cousot pour définir des transformations sémantiques de programmes [CC02] nous semble très adapté pour cela : si

[1] Simulink : `http ://www.mathworks.com`.
[2] Scade : `http ://www.esterel-technologies.com`.

l'on suppose que le programme source s'exécute correctement dans $\mathbb{R}$, la sémantique des expressions arithmétiques peut être définie par les règles telles que celles données à la figure 6.1. La sémantique $[\![.]\!]_{\mathbb{E}}$ calculant erreur globale, introduite à la section 2.3 indique l'erreur de précision survenant au cours d'une exécution et que l'on peut essayer de minimiser.

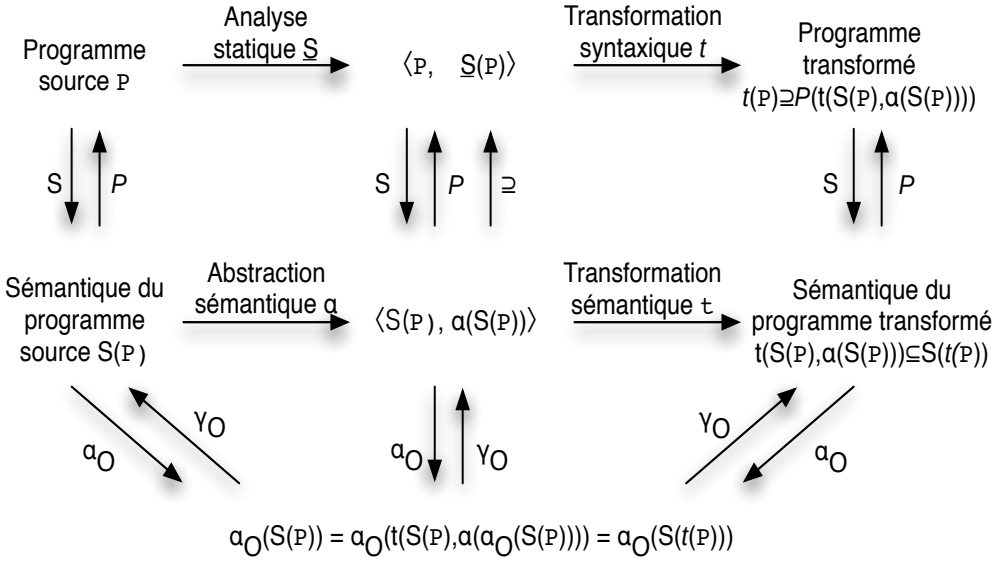

FIG. 6.2 – *Principe de transformation sémantique de programmes pour l'amélioration de la précision [CC02].*

La figure 6.2, reproduite de l'article [CC02]-figure 4, peut ainsi être utilisée pour définir une transformation sémantique de programme ayant pour but d'améliorer la précision numérique d'un calcul. Dans ce cas il semble envisageable de poser :

- $\mathsf{S}(\mathtt{p}) = [\![\mathtt{p}]\!]_{\mathbb{E}}$, c.à.d. que les sémantiques du programme source et du programme transformé sont données par la sémantique de l'erreur globale $[\![.]\!]_{\mathbb{E}}$;
- l'analyse statique $\underline{\mathsf{S}}$ utilisée pour indiquer comment transformer le programme indique l'ordre dans lequel les opérations doivent être évaluées et quelles réécriture des expressions choisir en conformité avec la sémantique idéale donnée à la figure 6.1 ;
- une transformation $\mathtt{t}(\mathtt{P})$ est plus précise qu'une transformation $\mathtt{t'}(\mathtt{P})$ si $[\![\mathtt{t}(\mathtt{P})]\!]_{\mathbb{E}} = f\vec{\varepsilon}_f + e\vec{\varepsilon}_e$, $[\![\mathtt{t'}(\mathtt{P})]\!]_{\mathbb{E}} = f'\vec{\varepsilon}_f + e'\vec{\varepsilon}_e$ tels que $f + e = f' + e' = [\![\mathtt{p}]\!]_{\mathbb{R}}$ et $e \leq e'$, c.à.d. si les deux programmes transformés produisent le même résultat dans $\mathbb{R}$ et si le premier introduit une perte de précision inférieure à celle du second ;
- Enfin, l'abstraction α_O permettant d'observer que la transformation préserve la sémantique de référence peut être définie pour une valeur de $[\![.]\!]_{\mathbb{E}}$ par $\alpha_O(f\vec{\varepsilon}_f + e\vec{\varepsilon}_e) = f + e$ ce qui signifie que le programme source et le programme transformé calculent les mêmes résultats dans la sémantique idéale $[\![.]\!]_{\mathbb{R}}$.

De nombreux travaux doivent cependant encore être menés pour définir entièrement une telle technique de transformation automatique de programmes. Notamment, il semble difficile de définir une analyse statique précise permettant de déterminer quelle transformation, parmi celles compatibles avec les règles de la figure 6.1, améliorera le plus la précision d'un calcul.

$i := 0$
while ($i < imax$)
    $t := i + ecart(i)$
    $a := somme\_machine(a, s(t))$
    $i := i + 1$

FIG. 6.3 – *Exemple de problème ouvert illustrant les difficultés dues au couplage de systèmes hétérogènes.*

6.3 Intégration système

Comme nous avons pu le constater tout au long de ce document, la validation des traitements numériques effectués par un logiciel embarqué s'inscrit au cœur de la problématique d'intégration de systèmes. En effet, celle-ci ne peut être réalisée sans tenir compte des couplages avec l'architecture matérielle sous-jacente, comme illustré par l'analyse de programmes assembleurs du chapitre 5, et sans tenir compte des couplages avec l'environnement physique, l'analyse de systèmes hybrides discrets-continus présentée au chapitre 4 ayant mis ce point en évidence. Les couplages avec le système d'exploitation, que nous n'avons pas abordés, sont aussi susceptibles de perturber la qualité des calculs : les fonctions mathématiques peuvent être implantées dans des librairies système, la préemption d'un processus, dans un système multitâches, implique une recopie des valeurs des registres en mémoire qui peut introduire des pertes de précision lorsque les registres disposent d'un nombre supérieur de bits, et la nécessité de connaître les dates d'exécution des instructions, comme nous avons pu le voir à la section 4.4 pour l'analyse de systèmes hybrides, peut impliquer des couplages avec le système d'opération temps réel dans lequel évolue un logiciel embarqué.

Pour illustrer l'importance et la difficulté des problèmes de couplages survenant en phase d'intégration, examinons par exemple le problème ouvert suivant, mentionné dans [GMP06] et formalisé dans [BDLM06]. On considère la boucle de l'algorithme 6.3 où aucune erreur n'est implicite. On suppose que la fonction $somme\_machine$ introduit une erreur d'arrondi fonction de a et $s(t)$ mais, en revanche, notre modélisation implique que $s(t)$ est représentable exactement en machine avec le format utilisé. Nous supposerons pour simplifier le problème que l'algorithme ne provoque pas de dépassement de capacité. En pratique, les hypothèses suivantes sont souvent vérifiées par les données :

$$\left\{\begin{array}{rcl} |somme\_machine(a, s(t)) - (a + s(t))| & < & 2^m \\ |f(t) - s(t)| & < & 2^m \\ |f(t)| & < & 1 \\ \left|\int_0^t f(t)dt\right| & < & 2^M \end{array}\right.$$

Dans la pratique, on constate qu'elles suffisent à vérifier les conclusions suivantes pour l'itération i entre 0 et $imax$.

$$|a| \lesssim 2^M \tag{6.1}$$

$$\left|a - \int_0^i f(t)dt\right| \lesssim 2^m \tag{6.2}$$

Le symbole $u \lesssim v$ signifie que l'on peut trouver une petite constante entière k telle $u < k \times v$. Dans les pires cas réalistes, les approches statistiques, ne permettent cependant pas de :

1. fixer des bornes au plus près pour l'implantation de l'accumulateur a afin que celui-ci ne provoque jamais de dépassement de capacité (cf. équation (6.1)) ;
2. borner finement l'erreur globale sur le résultat en prenant en compte la précision des données et des calculs (cf. equation (6.2)).

Concernant l'analyse statique, le logiciel Astrée [CCF+05] est à notre connaissance le plus avancé pour ce problème, pour aborder la question 1 (équation (6.1)). Il a notamment déjà permis de montrer qu'aucun dépassement de capacité ne pouvait survenir dans un logiciel de commandes de vol numériques contenant des intégrateurs avec écrêtage pour toute exécution (et en particulier pour tout $s(t)$) sur un processeur conforme à la norme IEEE754 et pour des durées de vol réalistes. Toujours à notre connaissance, Fluctuat est le seul logiciel d'interprétation abstraite permettant d'aborder la question 2 (équation (6.2)). Il permet notamment d'y répondre précisément pour le problème simplifié dans lequel on suppose que l'horloge est parfaite, pour des signaux particuliers échantillonnés de manière particulière (voir [GMP06] pour plus de détails). Cependant ces résultats ne sont pas directement généralisables.

Plus généralement, la validation de logiciels intégrés dans leur environnement d'exécution (milieu physique, architecture matérielle) permet aussi d'envisager la définition de critères de sûreté plus pertinents que ceux que l'on peut définir à partir d'un programme isolé. En ce qui concerne la précision numérique, nous avons vu, à la section précédente, qu'un programme pouvait être validé en fonction de ses interactions avec l'environnement et, par exemple, qu'il était possible d'exiger que celles-ci ne soient que peu affectées par l'arithmétique utilisée, flottante ou réelle.

Il est aussi envisageable de compléter l'étude des erreurs d'implantation (les imprécisions numériques dues à l'utilisation des nombres flottants) par une analyse des erreurs de méthode, dues à la discrétisation d'un modèle continu. Par exemple, il serait intéressant de borner l'erreur de discrétisation en calculant une approximation sûre de l'ensemble des solutions d'un système d'équations aux dérivées partielles et en la comparant à la solution calculée par l'algorithme effectivement implémenté dans un système. Tout comme pour les erreurs numériques, on peut alors ensuite envisager d'analyser l'impact de ces approximations sur les actions effectuées par le système logiciel.

Enfin, lorsqu'un logiciel embarqué est destiné à contrôler un système physique, on peut considérer que les propriétés de sûreté du système doivent porter sur le milieu physique uniquement. Par exemple, on peut vouloir montrer qu'un paramètre physique ne dépasse jamais un certain seuil, la partie logicielle du système n'intervenant que dans la preuve de cette propriété. Dans ce cas, la précision numérique n'est pas la seule source d'erreur et d'autres propriétés qu'il est nécessaire d'établir à l'aide d'analyses diverses (l'absence d'interblocages par exemple) peuvent intervenir dans la preuve de la propriété principale.

Bibliographie

[ACH+95] R. ALUR, C. COURCOUBETIS, N. HALBWACHS, T. A. HENZINGER, P.-H. HO, X. NICOLLIN, A. OLIVERO, J. SIFAKIS et S. YOVINE : The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138, 1995.

[Ami05] P. AMIRANOFF : *Analyse de programmes par instances par le biais des transducteurs*. Thèse de doctorat, Conservatoire National des Arts et Métiers, 2005.

[ANS85] ANSI/IEEE. *IEEE Standard for Binary Floating-point Arithmetic*, std 754-1985 édition, 1985.

[ASY96] K. T. ALLIGOOD, T. D. SAUER et J. A. YORKE : *Chaos, an Introduction to Dynamical Systems*. Springer-Verlag, 1996.

[Bar93] M. F. BARNSLEY : *Fractals Everywhere, Second Edition*. Academic Press, 1993.

[BBK+01] V. BLONDEL, O. BOURNEZ, P. KOIRAN, C. PAPADIMITRIOU et J. TSITSIKLIS : Deciding stability and mortality of piecewise affine dynamical systems. *Theoretical Computer Science*, 255, 2001.

[BD03] S. BOLDO et M. DAUMAS : Representable correcting terms for possibly underflowing floating point operations. *Dans* J.-C. BAJARD et M. SCHULTE, éditeurs : *Symposium on Computer Arithmetic*. IEEE Press, 2003.

[BDLM06] N. BRISEBARRE, M. DAUMAS, P. LANGLOIS et M. MARTEL : Intégration de systèmes calculants : un défi pour la sûreté numérique. Soumis au journal Techniques et Sciences Informatique, 2006.

[BDNN98] C. BODEI, P. DEGANO, F. NIELSON et H.-R. NIELSON : Control flow analysis for the pi-calculus. *Dans Concur'98*, numéro 1466 de Lecture Notes in Computer Science, pages 84–98. Springer-Verlag, 1998.

[Bea91] A. F. BEARDON : *Iteration of Rational Functions*. Numéro 132 de Graduate Texts in Mathematics. Springer-Verlag, 1991.

[Ber05] J. BERTRANE : Static analysis by abstract interpretation of the quasi-synchronous composition of synchronous programs. *Dans VMCAI'05*, numéro 3385 de Lecture Notes in Computer Science, pages 97–112. Springer-Verlag, 2005.

[BHN02] C. BISCHOF, P. D. HOVLAND et B. NORRIS : Implementation of automatic differentiation tools. *Dans Partial Evaluation and Semantics-Based Program Transformations, PEPM'02*. ACM Press, 2002.

[Bie51] L. BIEBERBACH : On the remainder of the runge-kutta formula in the theory of ordinary differential equations. *Z.A.M.P.*, 2, 1951.

[BM06a] O. BOUISSOU et M. MARTEL : A Runge-Kutta method for computing guaranteed solutions of ODEs. *Dans 12th GAMM - IMACS International Symposion on Scientific Computing, Computer Arithmetic and Validated Numerics*, 2006. accepté pour publication.

[BM06b] O. BOUISSOU et M. MARTEL : Static analysis by abstract interpretation of hybrid system. Soumis à Journal of Higher Order and Symbolic Computation, 2006.

[BMP03] H. BRONNIMANN, G. MELQUIOND et S. PION : The boost interval arithmetic library. *Dans Real Numbers and Computers Conference, RNC'5*, 2003.

[Bou92] F. BOURDONCLE : Abstract interpretation by dynamic partitioning. *Journal of Functional Programming*, 2(4):407–435, 1992.

[Bou05] O. BOUISSOU : Validation par interprétation abstraite de systèmes hybrides discrets-continus. Rapport technique, Rapport de stage de seconde année de Master, Université Paris 7, 2005.

[BPR+02] P. BAUDIN, A. PACALET, J. RAGUIDEAU, D. SCHOEN et N. WILLIAMS : Caveat : A tool for software validation. *Dans International Performance and Dependability Symposium*. IEEE Press, 2002.

[BV96] G. BAUDOIN et F. VIROLLEAU : *DSP : Les Processeurs de Traitement du Signal, Famille TMS320C5x*. Dunod, 1996.

[Car58] J. W. CARR : Error bounds for the runge-kutta single-step integration process. *Journal of the ACM*, 5(1), 1958.

[CC77] P. COUSOT et R. COUSOT : Abstract interpretation : A unified lattice model for static analysis of programs by construction of approximations of fixed points. *Dans Principles of Programming Languages 4*, pages 238–252. ACM Press, 1977.

[CC92] P. COUSOT et R. COUSOT : Abstract interpretation frameworks. *Journal of Logic and Symbolic Computation*, 2(4):511–547, 1992.

[CC94] P. COUSOT et R. COUSOT : Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). *Dans International Conference on Computer Languages*, pages 95–112. IEEE Computer Society Press, mai 1994.

[CC02] P. COUSOT et R. COUSOT : Systematic design of program transformation frameworks by abstract interpretation. *Dans Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, 2002. ACM Press, New York, NY.

[CCF96] F. CHAITIN-CHATELIN et V. FRAYSSÉ : *Lectures on Finite Precision Computations*. SIAM, 1996.

[CCF+05] P. COUSOT, R. COUSOT, J. FERET, A. MINÉ, D. MONNIAUX, L. MAUBORGNE et X. RIVAL : The astree analyzer. *Dans European Symposium on Programming, ESOP'05*, numéro 3444 de Lecture Notes in Computer Science, pages 21–30. Springer-Verlag, 2005.

[CD93] C. CONSEL et O. DANVY : Partial evaluation : principles and perspectives. *Dans Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages, POPL'93*, 1993.

[CGG+05] A. COSTAN, S. GAUBERT, E. GOUBAULT, M. MARTEL et S. PUTOT : A policy iteration algorithm for computing fixed points in static analysis of programs. *Dans Computer Aided Verification, CAV'05*, numéro 3565 de Lecture Notes in Computer Science. Springer-Verlag, 2005.

[Cha88] F. CHATELIN : *Valeurs propres des matrices*. Masson, 1988.

[Cha05] A. CHAPOUTOT : Analyse statique pour la précision numérique. Rapport technique, Rapport de stage de seconde année de Master, Université Paris 6, 2005.

[Che95] J.-M. CHESNEAUX : L'arithmétique stochastique et le logiciel CADNA. Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, 1995.

[CK90] J. R. CASH et A. H. KARP : A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides. *ACM Trans. Math. Softw.*, 16(3), 1990.

[CM06] A. CHAPOUTOT et M. MARTEL : Abstract frequency analysis of synchronous systems. *Dans ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, 2006. Poster session proceedings.

[Cou05] P. COUSOT : Integrating physical systems in the static analysis of embedded control software. *Dans* Kwangkeun YI, éditeur : *Third Asian Symposium on Programming Languages and Systems (APLAS)*, numéro 3780 de Lecture Notes in Computer Science, pages 135–138. Springer-Verlag, 2005.

[dDLM05] F. de DINECHIN, C. LAUTER et G. MELQUIOND : Assisted verification of elementary functions. Rapport technique 5683, INRIA, 2005.

[DM97] M. DAUMAS et J.-M. MULLER, éditeurs. *Qualité des Calculs sur Ordinateur*. Masson, 1997.

[DM04] M. DAUMAS et G. MELQUIOND : Generating formally certified bounds on values and round-off errors. *Dans* Vasco BRATTKA, Christiane FROUGNY et Norbert MÜLLER, éditeurs : *Proceedings of the 6th Conference on Real Numbers and Computers*, pages 55–70, Schloß Dagstuhl, Germany, 2004.

[DMM05] M. DAUMAS, Guillaume MELQUIOND et César MUÑOZ : Guaranteed proofs using interval arithmetic. *Dans* Paolo MONTUSCHI et Eric SCHWARZ, éditeurs : *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, pages 188–195, Cape Cod, Massachusetts, USA, 2005.

[DRT01] M. DAUMAS, L. RIDEAU et L. THÉRY : A generic library for floating-point numbers and its application to exact computing. *Dans Theorem Proving and Higher Order Logics, TPHOLs'01*, numéro 2152 de Lecture Notes in Computer Science. Springer-Verlag, 2001.

[Eda95] A. EDALAT : Dynamical systems, measures and fractals via domain theory. *Information and Computation*, 120(1), 1995.

[Eda96] A. EDALAT : Power domains and iterated function systems. *Information and Computation*, 124, 1996.

[Fer04] J. FERET : Static analysis of digital filters. *Dans ESOP'04*, numéro 2986 de Lecture Notes in Computer Science, pages 33–48. Springer-Verlag, 2004.

[Fer05a] J. FERET : *Analyse de systèmes mobiles par interprétation abstraite*. Thèse de doctorat, Ecole Polytechnique, 2005.

[Fer05b] J. FERET : The arithmetic-geometric progression abstract domain. *Dans Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, numéro 3385 de Lecture Notes in Computer Science, pages 42–58. Springer-Verlag, 2005.

[Fer05c] Jérôme FERET : Numerical abstract domains for digital filters. *Dans International workshop on Numerical and Symbolic Abstract Domains (NSAD 2005)*, 2005.

[FHL+01] C. FERDINAND, R. HECKMANN, M. LANGENBACH, F. MARTIN, M. SCHMIDT, H THEILING, S. THESING et R. WILHELM : Reliable and precise wcet determination for a real-life processor. *Dans Embedded Software, EMSOFT'2001*, numéro 2211 de Lecture Notes in Computer Science. Springer-Verlag, 2001.

[Fut71] Y. FUTAMURA : Partial evaluation of computation process - an approach to a compiler-compiler. *Systems, Computers, Controls*, 2(5):49–50, 1971.

[GGP+01] E. GOUBAULT, D. GUILBAUD, A. PACALET, B. STARYNKÉVITCH et F. VÉDRINE : A simple abstract interpreter for threat detection and test case generation. *Dans Proceedings of WAPATV'01 (ICSE'01)*, mai 2001.

[GJ79] M. R. GAREY et D. S. JOHNSON : *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. ISBN 0-7167-1045-5.

[GM97] M. GENGLER et M. MARTEL : Self-applicable partial evaluation for the pi-calculus. *Dans Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*, pages 36–46, 1997.

[GMP01] E. GOUBAULT, M. MARTEL et S. PUTOT : Concrete and abstract semantics of floating-point operations. Rapport technique DRT/LIST/DTSI/SLA/LSL/01-058, CEA, 2001.

[GMP02] E. GOUBAULT, M. MARTEL et S. PUTOT : Asserting the precision of floating-point computations : a simple abstract interpreter. *Dans* 11^{th} *European Symposium on Programming, ESOP'02*, numéro 2305 de Lecture Notes in Computer Science, pages 209–212, 2002.

[GMP06] E. GOUBAULT, M. MARTEL et S. PUTOT : Some future challenges in the validation of control systems. *Dans Proceedings of the European Congress on Embedded Real Time Software (ERTS'06)*, 2006.

[Gol91] D. GOLDBERG : What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[Gou01] E. GOUBAULT : Static analyses of the precision of floating-point operations. *Dans Static Analysis Symposium, SAS'01*, numéro 2126 de Lecture Notes in Computer Science, pages 234–259. Springer-Verlag, 2001.

[GP05] E. GOUBAULT et S. PUTOT : Weakly relational domains for floating-point computation analysis. *Dans First International Workshop on Numerical and Symbolic Abstract Domains*, 2005.

[GPR03] M. GRIMMER, K. PETRAS et N. REVOL : Multiple precision interval packages : Comparing different approaches. *Dans Dagstuhl Seminar on Numerical Software with Result Verification*, numéro 2991 de Lecture Notes in Computer Science. Springer-Verlag, 2003.

[Gri00] A. GRIEWANK : *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. Frontiers in Applied Mathematics. SIAM (Publisher), 2000. ISBN 0-89871-451-6.

[GVL90] G. H. GOLUB et C. F. VAN LOAN : *Matrix Computations*. The Johns Hopkins University Press, 2d edition, 1990.

[Har99] J. HARRISON : A machine-checked theory of floating point arithmetic. *Dans Theorem Proving and Higher Order Logics, TPHOLs'99*, numéro 1690 de Lecture Notes in Computer Science. Springer-Verlag, 1999.

[Hau96] J. R. HAUSER : Handling floating-point exceptions in numeric programs. *ACM Transactions on Programming Languages and Systems*, 18(2):139–174, 1996.

[Hen96] T. HENZINGER : The theory of hybrid automata. *Dans Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996.

[HHWT98] T. HENZINGER, P.-H. HO et H. WONG-TOI : Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[Hig97] N. J. HIGHAM : Testing linear algebra software. *Dans* R. F. BOISVERT, éditeur : *Quality of Numerical Software : Assessment and Enhancement*. Chapman and Hall, 1997.

[HLFZ01] G. HANROT, V. LEFEVRE, Rouillier F. et P. ZIMMERMANN : The MPFR library. Institut de Recherche en Informatique et Automatique, 2001.

[How60] R. HOWARD : *Dynamic Programming and Markov Processes*. Wiley, 1960.

[HRP94] N. HALBWACHS, P RAYMOND et Y.-E. PROY : Verification of linear hybrid systemss by means of convex approximations. *Dans SAS'94*, Lecture Notes in Computer Science. Springer Verlag, 1994.

[HS91] S. HUNT et D. SANDS : Binding time analysis : A new PERspective. *Dans ACM-SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulations, PEPM'91*, pages 154–165, 1991.

[HT98] M. HANDJIEVA et S. TZOLOVSKI : Refining static analyses by trace-based partitioning using control flow. *Dans Static Analysis Symposium, SAS'98*, numéro 1503 de Lecture Notes in Computer Science. Springer-Verlag, 1998.

[IEE87] IEEE. *IEEE Standard for Radix-Independent Floating-Point Arithmetic*, std 854-1987 édition, 1987.

[ISO89] International Electrotechnical Commission. *Binary floating-point arithmetic for microprocessor systems*, std 60559 édition, 1989.

[Jea02] B. JEANNET : Dynamic partitioning in linear relation analysis. *Formal Methods in System Design*, 23(1):5–37, 2002.

[JGS93] N. JONES, C. GOMARD et P. SESTOFT : *Partial Evaluation and Automatic Program Generation*. Prentice Hall International, International Series in Computer Science, 1993.

[JKDW01] L. JAULIN, M. KIEFFER, O. DIDRIT et E. WALTER : *Applied interval analysis*. Springer, 2001.

[JM82] N. D. JONES et S. S. MUCHNICK : A flexible approach to interprocedural flow analysis and programs with recursive data structures. *Dans Proceedings of the 9th ACM Symposium on Principles of Programming Languages*, 1982.

[KM06] D. KROB et M. MARTEL : Le master professionnel "ingénierie des systèmes industriels complexes" : une formation d'architecte système délivrée par l'ecole polytechnique, l'institut national des sciences et techniques nucléaires et l'université paris sud 11. *Dans Actes de la 4-ième Conférence AFIS (Association Française d'Ingénierie Système)*, 2006.

[Knu97] D. KNUTH : *The Art of Computer Programming - Seminumerical Algorithms*. Addison Wesley, third édition, 1997. Chapter 4, ISBN 0-201-89684-2.

[Lan01] P. LANGLOIS : Automatic linear correction of rounding errors. *BIT, Numerical Mathematics*, 41(3):515–539, 2001.

[Lau91] J. LAUNCHBURY : *Projection Factorisations in Partial Evaluation*. Cambridge University Press, 1991.

[LFW02] M. LANGENBACH, C. FERDINAND et R. WILHELM : Worst case execution time prediction. *Dans WCET workshop of EUROMICRO conf. on Real-Time Systems*, 2002.

[LMT98] V. LEFEVRE, J.M. MULLER et A. TISSERAND : Toward correctly rounded transcendentals. *IEEE Transactions on Computers*, 47(11):1235–1243, 1998.

[LN97] P. LANGLOIS et F. NATIVEL : Improving automatic reduction of round-off errors. *Dans IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*, volume 2, 1997.

[LPV74] M. LA PORTE et J. VIGNES : Error analysis in computing. *Information Processing 74*, 1974.

[MA00] B. MARRE et A. ARNOULD : Test sequences generation from lustre descriptions : Gatel. *Dans In Fifteenth IEEE Int. Conf. on Automated Software Engineering (ASE 2000)*, pages 229–237. IEEE Press, 2000.

[Mar96] M. MARTEL : Evaluation partielle et analyse statique de langages fonctionnels typés d'ordre supérieur. Rapport technique, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 1996. Rapport de stage de DEA.

[Mar00] M. MARTEL : *Analyse Statique et Evaluation Partielle de Systèmes de Processus Mobiles*. Thèse de doctorat, Université de la Méditerranée, Marseille, France, 2000.

[Mar02a] M. MARTEL : Propagation of roundoff errors in finite precision computations : a semantics approach. *Dans* 11^{th} *European Symposium on Programming, ESOP'02*, numéro 2305 de Lecture Notes in Computer Science, pages 194–208. Springer-Verlag, 2002.

[Mar02b] M. MARTEL : Static analysis of the numerical stability of loops. *Dans Static Analysis Symposium, SAS'02*, numéro 2477 de Lecture Notes in Computer Science, pages 133–150. Springer-Verlag, 2002.

[Mar03] M. MARTEL : Improving the static analysis of loops by dynamic partitioning techniques. *Dans Third IEEE International Workshop on Source Code Analysis and Manipulation, SCAM'03*, pages 13–21. IEEE Press, 2003.

[Mar04] M. MARTEL : Validation of assembler programs for dsps : a static analyzer. *Dans Program analysis for software tools and engineering, PASTE'04*, pages 8–13. ACM Press, 2004.

[Mar05a] M. MARTEL : An overview of semantics for the validation of numerical programs. *Dans Verification, Model Checking and Abstract Interpretation, VMCAI'05*, numéro 3385 de Lecture Notes in Computer Science, pages 59–77. Springer-Verlag, 2005.

[Mar05b] M. MARTEL : Towards an abstraction of the physical environment of embedded systems. *Dans First International Workshop on Numerical and Symbolic Abstract Domains*, 2005.

[Mar06] M. MARTEL : Semantics of roundoff error propagation in finite precision calculations. *Journal of Higher Order and Symbolic Computation*, 19:7–30, 2006.

[Mas05] J. MASCUNAN : Analyse statique de calculs en virgule fixe. Rapport technique, Rapport de stage de seconde année de Master, Université Paris 6, 2005.

[MD92] Information MANAGEMENT et Technology DIVISION : Patriot missile defense : Software problem led to system failure in Dhahran, saudi arabia. Rapport technique B-247-094, United State General Accounting Office, 1992.

[Mei98] J.-P. MEINADIER : *Ingénierie et intégration de systèmes*. Hermès-Lavoisier, 1998.

[Mei02] J.-P. MEINADIER : *Le métier d'intégration de systèmes*. Hermès-Lavoisier, 2002.

[MG98] M. MARTEL et M. GENGLER : Des étages en Concurrent ML. *Dans Rencontres Francophones du Parallélisme, Renpar'10*, 1998.

[MG00a] M. MARTEL et M. GENGLER : Analyse statique pour la sûreté des applications distribuées. *Dans Rencontres Francophones du Parallélisme, Renpar'12*, 2000.

[MG00b] M. MARTEL et Marc GENGLER : Communication topology analysis for concurrent programs. *Dans SPIN Model Checking and Program Verification*, volume 1885 de *Lecture Notes in Computer Science*. Springer-Verlag, août 2000.

[MG01] M. MARTEL et M. GENGLER : Partial evaluation of concurrent programs. *Dans 7th International Euro-Par Conference*, volume 2150 de *Lecture Notes in Computer Science*. Springer-Verlag, août 2001.

[Mil93] R. MILNER : The polyadic π-calculus : a tutorial. *Dans Logic and Compositionality*. Springer, 1993.

[Mil99] R. MILNER : *Communicating and Mobile Systems : the pi-Calculus*. Cambridge University Press, mai 1999.

[Min01] A. MINÉ : The octagon abstract domain. *Dans AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001.

[Min04] A. MINÉ : Relational abstract domains for the detection of floating-point run-time errors. *Dans ESOP'04*, volume 2986 de *Lecture Notes in Computer Science*, pages 3–17. Springer, 2004.

[Min05] A. MINÉ : *Domaines numériques abstraits faiblement relationnels*. Thèse de doctorat, Ecole Normale Supérieure, 2005.

[Mog92] T. Æ. MOGENSEN : Self-applicable partial evaluation for pure lambda calculus. *Dans Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'92*, pages 116–121, 1992.

[Mog94] T. Æ. MOGENSEN : Efficient self-interpretation in lambda calculus. *Journal of Functional Programming*, 2(3):345–364, July 1994.

[Mon01a] D. MONNIAUX : An abstract Monte-Carlo method for the analysis of probabilistic programs. *Dans Principles of Programming Languages, POPL'01*. ACM Press, 2001.

[Mon01b] D. MONNIAUX : *Analyse de programmes probabilistes par interprétation abstraite*. Thèse de doctorat, Université Paris IX Dauphine, 2001.

[Moo63] R. E. MOORE : *Interval Analysis*. Prentice-Hall, Englewood Cliffs, 1963.

[Moo92] F. C. MOON : *Chaotic and Fractal Dynamics*. Wiley-Interscience, 1992.

[Mos99] P. J. MOSTERMAN : An overview of hybrid simulation phenomena and their support by simulation packages. *Dans Hybrid Systems : Computation and Control*, Lecture Notes in Computer Science, pages 165–177. Springer Verlag, 1999.

[MRL01] C. MICHEL, M. RUEHER et Y. LEBBAH : Solving constraints over floating-point numbers. *Dans CP'2001, Seventh International Conference on Principles and Practice of Constraint Programming*, numéro 2239 de Lecture Notes in Computer Science, pages 524–538. Springer-Verlag, 2001.

[MTHM97] R. MILNER, M. TOFTE, R. HARPER et D. MACQUEEN : *The Definition of Standard ML (Revised)*. MIT-Press, 1997.

[MVM97] G. MULLER, E.-N. VOLANSCHI et R. MARLET : Scaling up partial evaluation for optimizing the sun commercial rpc protocol. *Dans Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and semantic based program manipulations, PEPM'97, Amsterdam*, pages 101–111. ACM, 1997.

[NJ01] N. S. NEDIALKOV et K. R. JACKSON : Some recent advances in validated methods for IVPs for ODEs. *Dans Proceedings of the International Minisymposium on Mathematical Modelling and Scientific Computations*, 2001.

[NN99] F. NIELSON et H. R. NIELSON, éditeurs. *ML with Concurrency*. Monograph in Computer Science. Springer, 1999.

[PEE97] P. J. POTTS, A. EDALAT et H. M. ESCARDÓ : Semantics of exact real arithmetic. *Dans Procs of Logic in Computer Science*. IEEE Computer Society Press, 1997.

[PGM04] S. PUTOT, E. GOUBAULT et M. MARTEL : Static analysis-based validation of floating-point computations. *Dans Follow-up of the seminary on Numerical Software with Result Verification, at Dagstuhl, Germany*, numéro 2991 de Lecture Notes in Computer Science. Springer-Verlag, 2004.

[Pri91] M. PRIEST : Algorithms for arbitrary precision floating point arithmetic. *Dans* P. KORNERUP et D. MATULA, éditeurs : *Symposium on Computer Arithmetic*, pages 132–144. IEEE Computer Society Press, 1991.

[PTVF92] W. H. PRESS, S. A. TEUKOLSKY, W. T. VETTERLING et B. P. FLANNERY : *Numerical Recipes in C*. Cambridge University Press, 1992.

[Put94] M. L. PUTERMAN : *Markov decision processes : discrete stochastic dynamic programming*. Wiley series in Probability and Mathematical Statistics : Applied Probability and Statistics. John Wiley and Sons Inc., New York, 1994.

[Rep99] John H. REPPY : *Concurrent Programming in ML*. Cambridge University Press, 1999.

[Riv04] X. RIVAL : Symbolic transfer function-based approaches to certified compilation. *Dans Principles of Programming Languages, POPL*, ACM Press, 2004.

[RR02] N. REVOL et F. ROUILLIER : Motivations for an arbitrary precision interval arithmetic and the MPFI library. Rapport technique RR-200227, Laboratoire de l'Informatique du Parallélisme, ENS-Lyon, France, 2002.

[Rum99] S. M. RUMP : INTLAB - INTerval LABoratory. *Dans* T. CSENDES, éditeur : *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, 1999.

[SBM00] T. SHEARD, Z. BENAISSA et M. MARTEL : Introduction to multi-stage programming using MetaML. Rapport technique, Oregon Graduate Institute of Science and Technology, 2000.

[SNN97] K. L. SOLBERG, F.. NIELSON et H. R. NIELSON : Systematic realisation of control flow analyses for CML. *Dans Proceedings of the ACM-SIGPLAN International Conference on Functional Programming, ICFP'97*, pages 38–51. ACM, 1997.

[Sta05] O. STAUNING : *Analyse de programmes par instances par le biais des transducteurs*. Thèse de doctorat, Conservatoire National des Arts et Métiers, 2005.

[Tex97] Texas Instruments. *TMS320C3x User's Guide*, 1997.

[Tex98] Texas Instruments. *TMS320C3x/C4x Assembly Language Tools User's Guide*, 1998.

[The01] H. THEILING : Ilp-based interprocedural path analysis. *Dans Embedded Software, EMSOFT'2001*, numéro 2491 de Lecture Notes in Computer Science. Springer-Verlag, 2001.

[TS97] W. TAHA et T. SHEARD : Multi-stage programming with explicit annotations. *Dans Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and Semantic Based Program Manipulations, PEPM'97*, pages 203–217. ACM, 1997.

[Ven02] A. VENET : Nonuniform alias analysis of recursive data structures and arrays. *Dans Static Analysis Symposium, SAS'02*, numéro 2477 de Lecture Notes in Computer Science. Springer-Verlag, 2002.

[Vig93] J. VIGNES : A stochastic arithmetic for reliable scientific computation. *Mathematics and Computers in Simulation*, 35(3):233–261, 1993.

[Wil63] J. H. WILKINSON : *Rounding errors in algebraic processes*. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[XRB00] Z. XU, T. REPS et Miller B. : Safety checking of machine code. *Dans Programming Language Design and Implementation, PLDI*. ACM, 2000.

[XRB01] Z. XU, T. REPS et Miller B. : Typestate checking of machine code. *Dans European Symposium on Programming, ESOP'01*, numéro 2028 de Lecture Notes in Computer Science. Springer-Verlag, 2001.

Résumé

La validation de logiciels embarqués effectuant des traitements numériques est un thème de recherche situé à l'intersection des principaux problèmes d'intégration système. D'une part, les calculs réalisés par ces programmes permettent de contrôler des processus physiques, et, par conséquent, dépendent de l'environnement physique dans lequel un système embarqué évolue. D'autre part, ces calculs dépendent de l'arithmétique de l'ordinateur utilisé pour les réaliser et ne peuvent donc être analysés qu'en tenant compte de l'architecture matérielle sur laquelle les programmes sont exécutés.
Après avoir comparé les propriétés calculées par les principales analyses dynamiques existantes pour la précision numérique, ce document présente la sémantique des séries d'erreurs qui permet d'identifier les opérations responsables des principales pertes de précision au cours d'un calcul. Ces informations sont précieuses pour comprendre l'origine des erreurs et aident à améliorer la qualité des traitements numériques.
Ensuite, plusieurs interprétations abstraites de la sémantique des séries d'erreurs sont décrites. Elles permettent notamment de regrouper statiquement ou dynamiquement certain termes d'erreurs, pour des questions de performances, et de déterminer la stabilité des calculs effectués dans une boucle, par le calcul d'exposants de Lyapunov abstraits. D'autres abstractions sont ensuite proposées, pour intégrer certains couplages avec l'environnement des programmes analysés : ceux avec l'environnement physique, via l'analyse de systèmes hybrides, et ceux avec le processeur, via l'analyse de codes assembleur. Enfin, une description des analyseurs statiques Fluctuat est donnée. Ces analyseurs permettent de valider la précision numérique de programmes écrits en langage C et en assembleur.

Abstract

The validation of embedded softwares which perform numerical processings illustrates most of system integration difficulties. First, the computations performed by these programs control physical processes and, consequently, depend on the physical environment in which the embedded system runs. Second, these computations depend on the computer arithmetic and cannot be carrefully analyzed if the hardware used to execute them is neglected.
This document starts by comparing the properties computed by the main existing dynamic analysis for numerical precision. Then, it introduces the semantics of error series, which enables one to identify the operations responsible of the main precision losts arising during a finite precision calculation. This information is useful to understand the origin of errors and helps the programmer to improve the quality of numerical processings. Next, a few abstract interpretations of the semantics of error series are introduced. A first abstraction makes it possible to statically or dynamically merge some error terms in order to speedup the analyses and a second abstraction is used to determine the stability of the calculations performed in loops by computing abstract Lyapunov exponents. Other abstractions also are presented, to include some couplings between programs and their environment : couplings with the physical environment and couplings with the processor by means of the analysis of assembler codes. Finally, the Fluctuat analyzers are described. These tools enable one to validate the numerical precision of C and assembler programs.