ALGORITHMES, MACHINES ET LANGAGES

GÉRARD BERRY

Membre de l'Institut (Académie des sciences)
et de l'Académie des technologies, professeur au Collège de France

Mots-clés : algorithmes, données, langage, machines, informatique, programmation

La série de cours et séminaires « Structures de données et algorithmes pour la vérification formelle » est disponible, en audio et/ou en vidéo, sur le site internet du Collège de France (http://www.college-de-france.fr/site/gerard-berry/course-2015-2016.htm) ainsi que le colloque « Arts et sciences, de nouveaux domaines pour l'informatique » (http://www.college-de-france.fr/site/gerard-berry/symposium-2015-2016.htm).

ENSEIGNEMENT

COURS ET SÉMINAIRES – STRUCTURES DE DONNÉES ET ALGORITHMES POUR LA VÉRIFICATION FORMELLE

Introduction

Après la présentation générale de la vérification formelle des programmes et circuits dans le cours 2014-2015, le cours 2015-2016 a été consacré aux méthodes automatiques de vérification fondées sur des théories décidables. Ces méthodes, introduites pour certaines dès les années 1960, ont fait des progrès considérables ces dernières années avec l'invention de nouveaux algorithmes remarquablement efficaces et la combinaison de ces algorithmes. Elles ont l'avantage de ne demander que peu de connaissances de leur fonctionnement interne à l'utilisateur, qui peut ainsi se consacrer davantage à sa propre problématique. Elles sont très utilisées industriellement dans le monde des circuits électroniques, soit pour vérifier le comportement, soit pour valider les nombreuses étapes de traitement de la spécification au masque de circuit final. En logiciel, elles sont de plus en plus utilisées comme complément idéal des systèmes de types pour détecter les erreurs dès la compilation, pour la recherche de bugs par analyse statique, et pour la génération automatique de tests dirigés.

Le cours a étudié les principales méthodes automatiques et leurs applications :

– d'abord, les *binary decision diagrams* (BDD), première structure de calcul booléen efficace déjà brièvement présentée et utilisée dans les cours précédents ;

– puis, les algorithmes analysant la satisfiabilité de grandes formules booléennes, ce qu'on appelle le problème SAT. Ils ont fait des progrès énormes ces vingt dernières années et sont devenus des outils standards dans de nombreux domaines de l'informatique, et les algorithmes SMT (satisfiabilité modulo théories), qui permettent de vérifier des formules arithmétiques ou symboliques bien plus riches ;

– ensuite, les algorithmes de vérification d'automates temporisés, adaptés à la validation des systèmes temps-réels ;

– enfin, les algorithmes explicites, qui énumèrent les états et transitions d'un système au lieu de le traiter symboliquement.

Cours 1 – Les BDD (*binary decision diagrams*) et leurs applications

9 mars 2016

Après une présentation générale des cours de l'année 2016 suivie d'un rappel des notions principales de complexité algorithmique et de calcul booléen, le cours a étudié les diagrammes de décisions binaires ou BDD (*binary decision diagrams*). Ce sont les premières structures à avoir révolutionné le calcul booléen, essentiel dans de nombreux domaines applicatifs, mais qu'on a longtemps pensé inutilisable en pratique de par sa complexité : la satisfaction booléenne est le prototype du problème NP-complet.

Les BDD ont été introduits par plusieurs auteurs dans les années 1970, mais ont surtout été systématisés et mis en pratique grâce au travail fondamental de Randy Bryant à Carnegie-Mellon à la fin des années 1980. Ce sont des graphes sans cycles avec partage maximal donnant une forme canonique compacte des expressions booléennes, modulo la donnée d'un ordre fixe des variables. La canonicité implique que l'égalité de fonctions booléennes se décide en temps constant, les opérations logiques de base se faisant elles en temps polynomial d'une façon récursive simple. Le cours a présenté l'implémentation efficace des BDD, montré l'influence décisive de l'ordre des variables sur leur taille, puis décrit les heuristiques permettant d'éviter autant que possible leur explosion en mémoire. À travers des exemples liés au langage Esterel étudié les années précédentes, le cours a expliqué pourquoi les BDD sont bien adaptés à la synthèse, l'optimisation et la vérification de circuits ou de contrôle complexe dans les programmes logiciels. Le cours s'est terminé par la présentation des *zero-suppressed decision diagrams* (ZDD), graphes dont le fonctionnement est identique à celui des BDD, mais avec une sémantique très différente qui concerne le codage de familles d'ensembles. Il a étudié brièvement l'application des ZDD à la recherche de pannes minimales en analyse de défaillance.

Il faut noter que Donald Knuth a qualifié les BDD d'une des structures de données les plus importantes introduites à la fin du XX^{e} siècle, et les ZDD d'une des plus belles structures de données jamais inventées en informatique.

Séminaire 1 – Application des BDD dans la conception de circuits intégrés

Patrick Vuillod (Synopsys) et Jean-Christophe Madre (Mentor Graphics), le 9 mars 2016

La synthèse des circuits numériques est un champ d'application majeur des BDD. L'exposé a présenté le processus de conception de ces circuits numériques, puis décrit deux applications des BDD dans ce processus : tout d'abord, la génération des circuits à partir de leurs descriptions algorithmiques, puis la réduction de la consommation électrique de ces circuits.

Cours 2 – SAT : La satisfaction booléenne

16 mars 2016

En calcul booléen, il y a trois problèmes : le codage efficace de fonctions booléennes, la satisfiabilité d'une formule pour au moins un jeu de valeurs des variables, appelée le problème SAT, et sa validité pour toutes les valeurs des variables (équivalente à la non-satisfiabilité de sa négation). Si les BDD sont devenus le standard pour la représentation de fonctions booléennes, ils sont moins efficaces pour SAT car ils portent trop d'information : un BDD encode toutes les valeurs des variables qui rendent une formule vraie, alors qu'il suffit d'en trouver un seul jeu pour garantir la satisfiabilité. Les premiers algorithmes SAT, nés dans les années 1960, étaient fondés soit sur l'élimination de variables par suite de résolutions logiques de variables, soit sur l'exploration systématique des valeurs des variables, les deux méthodes étant souvent exponentielles en pratique. En 1972, Stephen Cook a introduit la notion de NP-complétude et a montré que SAT est le prototype du problème NP-complet, qu'on pensait alors non soluble en pratique car exponentiel dans le pire cas.

Mais, depuis les années 2000, de nouveaux algorithmes ont considérablement amélioré SAT en mêlant recherche en arbre et apprentissage des causes des échecs. Même si leurs pires cas restent exponentiels, ils permettent de résoudre des problèmes SAT sur des exemples concrets comportant des centaines de milliers de variables, voire des millions. Ils sont utilisés dans l'industrie pour la vérification de circuits et de programmes embarqués et pour la génération de tests. Ils jouent un rôle central dans la vérification de conformité des transformations conduisant de la spécification d'un circuit aux composants électroniques intégrés. Une compétition mondiale annuelle joue un rôle prépondérant dans l'élaboration et le réglage de ces algorithmes, dont on ne sait pas encore vraiment pourquoi ils marchent si bien en pratique.

Après avoir montré pourquoi SAT est bien adapté à la résolution des problèmes en vérification et plus généralement en combinatoire, le cours a détaillé les algorithmes dits CDCL (*conflict-driven clause learning*). Améliorant considérablement les techniques classiques d'exploration en arbre, ils apprennent de nouvelles clauses simplificatrices par analyse des conflits et utilisent des structures de données paresseuses qui suppriment quasiment tout coût administratif pendant l'exploration. Le cours a étudié les différentes options en matière d'ordre des variables, de retour arrière et d'oubli des clauses apprises.

Séminaire 2 – SAT : des victoires contre les problèmes difficiles

Laurent Simon (Laboratoire bordelais de recherche en informatique, université de Bordeaux), le 16 mars 2016

Le séminaire a présenté l'aventure scientifique qui a conduit aux progrès du domaine SAT. Il a exposé les ingrédients essentiels des démonstrateurs SAT actuels, qui ont évolués au fil des compétitions et des applications rencontrées. Certains de ces ingrédients essentiels sont très heuristiques. Par exemple, plutôt que de chercher intelligemment une solution, on préfère foncer vers l'échec et prendre le temps de l'analyser. En se fondant sur les échecs passés, on apprend alors de nouvelles clauses permettant d'accélérer la recherche en évitant de repasser toujours aux mêmes endroits. On utilise aussi des politiques de redémarrage à zéro des recherches extrêmement fréquentes, ainsi que des méthodes d'oubli des clauses apprises de plus en plus agressives. Comme de nombreux problèmes combinatoires difficiles dans des domaines variés peuvent se traduire efficacement en SAT, le champ d'application ne cesse de grandir. Mais les démonstrateurs SAT sont devenus des systèmes de science expérimentale quasiment impossibles à analyser par les méthodes algorithmiques classiques, et l'on bute encore sur une question épineuse : comment utiliser au mieux les architectures parallèles pour accélérer SAT ?

Cours 3 – SMT : la satisfaction modulo théories

23 mars 2016

La satisfaction modulo théories (SMT) s'intéresse aux formules mathématiques mêlant logique propositionnelle et théories mathématiques décidables, ou quelquefois semi-décidables : fonctions non interprétées, théories arithmétiques diverses (programmation linéaire réelle ou entière, inéquations aux différences, arithmétique de Presburger avec addition et comparaisons, etc.), théorie des tableaux, manipulation de bitvecteurs, théories de pointeurs pour la gestion mémoire, etc. Comme les solveurs SAT sur lesquels ils s'appuient d'ailleurs, les solveurs SMT sont en progrès constant, en termes d'efficacité comme en termes d'applications. Ils sont de plus en plus utilisés en analyse statique et vérification de programmes, en programmation par contraintes, en optimisation, etc. En vérification, le couplage fort d'un langage de programmation bien conçu et d'un système SMT puissant permet une détection efficace de bugs dès l'écriture du programme, voire une preuve formelle complète de programmes dans certains cas.

Les formules traitées par les solveurs SMT font interagir la logique et les différentes théories. Les solveurs les décomposent finement pour construire des formules propositionnelles telles que chaque atome booléen est une formule d'une seule théorie, tout en permettant le partage de variables entre atomes. Le cours a montré comment l'analyse de satisfiabilité de ces formules propositionnelles fournie par les solveurs SAT interagit avec les solveurs des théories particulières, puis a détaillé les difficultés de cette interaction. Il a ensuite présenté les théories principales utilisées en pratique, ainsi que les algorithmes associés. Il a enfin montré comment on peut introduire des quantificateurs pour simplifier et généraliser les formules à vérifier, mais en perdant la décidabilité d'une façon contrôlée au plus fin. Le tout a été illustré par des exemples liés à la vérification des programmes.

Séminaire 3 – SMT en pratique : le démonstrateur Alt-Ergo

Sylvain Conchon (Laboratoire de spécification et vérification, université Paris-Sud), le 23 mars 2016

Alt-Ergo est un démonstrateur SMT spécialement conçu pour prouver la validité des formules logiques engendrées par des outils de vérification de programme comme Frama-C, Why3, Spark ou l'Atelier-B. Ces formules contiennent des déclarations de types de données, des axiomes pour définir des structures de données complexes, des modèles de mémoires, des théories spécifiques (par exemple ensembles finis, flottants, vecteurs de bits), et des obligations de preuve provenant de propriétés de programmes à vérifier.

Les choix de conception et d'implémentation d'Alt-Ergo ont tous été guidés par le traitement efficace de ces formules. C'est le seul démonstrateur SMT qui permet de manipuler directement des formules de logique du premier ordre avec des types polymorphes « à la ML ». Il dispose de théories prédéfinies utiles à la preuve de programmes comme la théorie de l'égalité avec symboles non interprétés, l'arithmétique sur les entiers et les réels, la théorie des tableaux fonctionnels polymorphes avec extensionalité, les types énumérés, les enregistrements et les symboles associatifs/commutatifs.

Pour traiter les formules avec quantificateurs, Alt-Ergo repose sur un SAT-solveur « paresseux » original et sur un algorithme de filtrage de termes polymorphes. Son noyau met en œuvre une méthode de combinaison de procédures de décision fondée sur des techniques de réécriture, ce qui lui permet par exemple de traiter les symboles associatifs/commutatifs de manière efficace.

Cours 4 – La vérification formelle des systèmes temporisés

30 mars 2016

Les systèmes temporisés sont ceux pour lesquels les contraintes liées au temps physique sont essentielles pour la correction du fonctionnement : systèmes embarqués temps-réel, protocoles de communication dépendant directement du temps ou soumis à des contraintes de délai de transmission, etc. Un formalisme standard pour les décrire est celui des automates temporisés, dont les transitions peuvent être gouvernées par des variables appelées horloges. Ces variables avancent toutes au même rythme en fonction du temps. Elles conditionnent les transitions, et on peut les remettre individuellement à zéro lors de transitions. En s'appuyant sur des exemples écrits à l'aide du système UPPAAL décrit ensuite dans le séminaire, le cours a présenté le formalisme et la façon d'exprimer les propriétés à vérifier. Il a ensuite décrit les algorithmes permettant de décider efficacement la validité des propriétés des automates temporisés. Ces algorithmes de nature géométrique étudient les contraintes mutuelles que l'automate impose aux horloges, soit de façon exacte, soit par interprétation abstraite. Ils traduisent le problème de vérification initial en ensembles d'inégalités aux différences d'horloges, dont on peut décider la satisfiabilité avec des algorithmes rapides, également utilisés dans les systèmes SMT décrits dans le cours précédent.

Séminaire 4 – *Real-time model-checking of embedded systems*

Kim Larsen (CISS, Center for Embedded Software Systems, Ålborg University), le 30 mars 2016

Kim Larsen, professeur à l'université d'Ålborg (Danemark) et directeur du CISS, est un des acteurs principaux de la vérification formelle de systèmes temps-réel. Il est

l'architecte du système de vérification UPPAAL, développé par les universités d'Ålborg et Uppsala, qui est le système de vérification d'automates temporisés le plus puissant et le plus utilisé actuellement. Après un rappel sur les systèmes embarqués et la vérification formelle qualitative et quantitative, des propriétés logiques et temporelles des systèmes, il a présenté le design d'UPPAAL et ses principales fonctionnalités à travers l'exemple d'un système de signalisation ferroviaire, puis détaillé les principaux algorithmes à l'œuvre. Il a ensuite illustré la puissance de la vérification temporisée par plusieurs exemples industriels : détection et correction d'un très ancien bug qui n'avait jamais pu être caractérisé auparavant dans les systèmes audio/vidéo de Bang & Olufsen, vérification de protocoles chez Philips, applications diverses dans l'automobile, ordonnancement de tâches temporisées, etc.

Cours 5 – La vérification par énumération explicite

6 avril 2016

Toutes les techniques de vérification des cours précédents reposaient sur une représentation implicite des systèmes par des formules booléennes, symboliques ou numériques. Au contraire, la vérification explicite repose sur une exploration systématique du système à étudier par énumération explicite de ses états et transitions. En pratique, la vérification explicite a des propriétés très différentes de celles des vérifications implicites. Par exemple, elle donne de bons résultats sur les systèmes très asynchrones comme les protocoles de communication et les algorithmes distribués en général, souvent moins adaptés à la vérification implicite qui, elle, est très performante pour les systèmes synchrones. D'autre part, elle se parallélise bien, ce qui n'est pas encore le cas pour les algorithmes implicites.

Mais, pour que la vérification explicite soit efficace, il faut impérativement implémenter certaines optimisations délicates : systèmes de caches efficaces et de très grande taille pour la détection des états déjà visités, réductions d'ordres partiels pour éviter d'énumérer dans tous les ordres possibles des suites d'actions indépendantes, interprétation abstraite par hashcodes ne traitant pas les collisions, utilisation systématique de l'aléatoire, etc.

Le cours a étudié les algorithmes et leurs applications à travers deux des principaux systèmes du domaine : SPIN (Simple Promela Interpreter), développé par G. Holtzmann aux Bell Labs, et CADP, développé à l'Inria par H. Garavel, R. Mateescu et leur équipe. SPIN est un système compact fondé sur un langage de spécification issu de C, simple d'usage mais relativement limité, alors que CADP se présente sous la forme d'une riche librairie d'algorithmes combinables entre eux et gérés par un langage de script. CADP prend comme entrée des spécifications écrites dans divers calculs algébriques de processus communicants, et résout des problèmes d'énumération, d'équivalence par bisimulation, de preuves hiérarchiques par réductions compositionnelles, etc. SPIN et CADP ont tous deux des groupes d'utilisateurs académiques et industriels très actifs.

Séminaire 5 – La vérification formelle appliquée aux protocoles cryptographiques

Stéphanie Delaune (LSV, ENS Cachan), le 6 avril 2016

Les protocoles cryptographiques sont les algorithmes et programmes qui permettent d'établir une communication sécurisée. Ils sont fragiles et constituent le principal point d'entrée pour les attaques de sécurité, comme le montreront quelques

exemples. Ils sont aussi très difficiles à analyser. La preuve formelle est de plus en plus vue comme le meilleur moyen, voire le seul moyen, d'assurer le bon fonctionnement de ces protocoles. Elle demande souvent la mise en œuvre de techniques très élaborées.

Cours 6 – Réponses aux questions de l'année

13 avril 2016

Ce cours a été consacré à trois compléments aux cours de l'année, puis aux réponses aux questions reçues par courrier électronique et à celles posées en salle. Le premier complément a concerné la preuve de correction du synchroniseur de l'instruction parallèle d'Esterel, central pour la sémantique et l'implémentation du langage et fondé sur un codage 2-adique. La preuve a été formalisée dans le système de vérification Why-3 développé à Orsay, qui appelle lui-même les deux solveurs SMT CVCC et Alt-Ergo (voir le séminaire 3 du 23 mars 2016). Le deuxième complément a approfondi le formalisme CHAM de la machine chimique développé à la fin des années en 1989 avec Gérard Boudol pour modéliser le parallélisme asynchrone, qui a trouvé de nombreux successeurs. Le troisième complément a présenté la vérification en CADP d'un système de perçage qui sert souvent d'exemple pour la vérification formelle.

Les questions écrites traitées ont été les suivantes :

• Peut-on concevoir des systèmes grandement distribués et sûrs, ou aussi des systèmes « tolérants les fautes », sortes d'hybrides entre preuves et sûreté de fonctionnement ?

• Sur les protocoles : il y a de nombreuses initiatives pour de nouveaux standards. Comment faire pour y exploiter les avancées théoriques présentées dans le cours ?

• Sur les optimisations de circuits : peut-on capitaliser sur les opérations déjà effectuées, par exemple quand on retrouve des sous-expressions logiques vues ailleurs ? Peut-on enregistrer les optimisations déjà réalisées dans des formats adaptés ?

Séminaire 6 – Vers une automatisation sûre et expressive : combiner preuves automatiques et interactives

Chantal Keller (LRI, université Paris-Sud), le13 avril 2016

Les précédents cours et séminaires ont exploré la diversité des prouveurs, chacun ayant son degré d'automatisation, d'expressivité et de sûreté. Le séminaire a présenté diverses approches pour combiner ces outils dans le but de bénéficier des avantages de chacun, en portant une attention particulière à deux familles de prouveurs : d'une part, les assistants de preuve, outils très expressifs et sûrs permettant la vérification de propriétés complexes mais peu automatisés, d'autre part, les prouveurs SMT, très automatisés et généralistes, mais qui ont une expressivité et une sûreté plus réduites. Plusieurs implantations récentes de ces diverses approches ont été présentées : interaction entre prouveurs SMT et Coq grâce à Ergo, vérifieurs SMTCoq et Why-3, tactique automatique Sledgehammer de l'assistant Isabelle, et preuve de programmes semi-automatisée dans le langage F* (cf. séminaire 1).

Cours à l'extérieur – L'importance des langages en informatique

Irisa Rennes, le 4 novembre 2015

Les langages sont omniprésents en informatique, et ce, à de nombreux niveaux : spécification, architecture, programmation, test, documentation, etc. Les plus connus sont les langages de programmation, qui traduisent directement nos idées algorithmiques en instructions d'ordinateurs. Ils ont été développés historiquement en très grand nombre, avec des méthodes de pensée et d'action, des syntaxes et des sémantiques très variées, parfois complémentaires et parfois franchement incompatibles. Les langages les moins bien compris sont les langages de spécification, qui constituent sans doute le point faible principal de la réalisation des systèmes. L'exposé a étudié l'évolution des différents modes de pensée qui ont conduit aux langages actuels, en insistant sur les critères qui favorisent ou empêchent le succès d'un langage donné. Il a montré l'importance des approches formelles et de l'incorporation directe de la vérification formelle dans les langages.

Séminaire – Intégration de la vérification formelle dans les langages de programmation

Thomas Jensen, Inria Rennes, le 4 novembre 2015

La phase de validation d'un logiciel occupe une partie importante du temps consacré à sa conception. Afin d'optimiser ce processus, plusieurs langages proposent d'intégrer des constructions linguistiques qui aident à la vérification formelle de propriétés de sûreté et de sécurité. L'exposé est revenu sur les contrats de logiciels comme ceux utilisés dans le langage Spec#, a comparé les vérifications dynamiques et statiques de contrats de logiciels, et a examiné des propositions plus récentes sur la vérification de propriétés de sécurité en Java issues de spécifications formelles.

Colloque – Arts et sciences, de nouveaux domaines pour l'informatique

Ce colloque a eu pour but de sortir des applications connues de l'informatique pour montrer son action dans deux domaines très différents : les arts et les sciences. R. Teyssier (université de Zurich) a d'abord présenté les grandes simulations de galaxies et d'étoiles en astronomie, puis F. Goussard (Muséum nationale d'histoire naturelle) a présenté l'utilisation de l'analyse/synthèse 3D pour la visualisation et l'étude des fossiles. Pour les arts, A.-C. Worms (Art2M) a présenté la problématique générale de l'art numérique, puis les artistes A. Meunier et C. Bruno ont présenté des approches originales et surprenantes d'exploitation artistique des ressources du Web. Ensuite C. Narteau (Institut de physique du globe de Paris) a montré comment simuler finement les dunes de sable à l'aide d'automates cellulaires, et Y. Rondelez a montré comment effectuer des calculs biochimiques avec des brins d'ADN comme données et des enzymes comme moteurs de calcul. Abordant ensuite l'informatique musicale, G. Assayag (IRCAM) a présenté les principes de la co-improvisation homme/ordinateur, que J. Nika (IRCAM) a précisés avec le système ImproteK qu'il a développé avec M. Chemillier (EHESS). Tout cela a été mis en action dans un concert avec B. Lubat au piano et au chant et M. Chemillier et J. Nika au réglage d'ImproteK. Le concert a été accompagné d'une discussion avec le public.

PUBLICATIONS

BERRY G., « Un géant de création et de simplicité », *in : Lettres à Alan Turing*, réunies par J.-M. Lévy-Leblond, Vincennes, Éditions Thierry Marchaisse, coll. « Lettres à... », 2016, p. 47-62.