Algorithmes, machines et langages

M. Gérard BERRY, membre de l'Institut
(Académie des sciences) et de l'Académie des technologies, professeur

ENSEIGNEMENT : PROUVER LES PROGRAMMES : POURQUOI, QUAND, COMMENT[a] ?

Cours 1 : La révolution informatique dans les sciences

Ce cours, disjoint du reste des cours de l'année, a été donné à l'École normale supérieure de Lyon le 28 janvier 2015.

L'informatique sert depuis longtemps de moyen de calcul dans les autres sciences et en mathématiques. Mais un changement profond de vison de son rôle dans les sciences naturelles est en cours : la pensée algorithmique et ses réalisations informatiques apportent désormais un regard nouveau sur la façon d'étudier les phénomènes, en particulier dans les sciences de la vie historiquement peu touchées par les approches mathématiques.

La modélisation et la simulation informatiques, originellement dédiées à la simple imitation du réel, deviennent des outils conceptuels et pratiques fondamentaux pour comprendre les phénomènes. Elles conduisent à une nouvelle vision algorithmique des lois de la nature, où l'information et son calcul servent à représenter de façon uniforme les objets classiques d'étude que sont la matière, les ondes et l'énergie.

Cette vision est fondée sur de nouvelles mathématiques discrètes qui complètent les mathématiques continues habituelles et ouvrent de nouveaux champs d'action et schémas de raisonnement. Le cours a illustré ce propos par des exemples pris en physique, géophysique, astronomie, biologie et médecine. Il a enfin montré comment l'informatique moderne commence à transformer aussi les mathématiques, à travers la possibilité de conduire en machine des preuves vraiment formelles de très grande taille, à l'aide de logiciels implémentant des logiques formelles très puissantes qui sont étudiées dans la suite du cours.

---

a. Cours et séminaires sont disponibles en audio et en vidéo sur le site internet du Collège de France : http://www.college-de-france.fr/site/gerard-berry/course-2014-2015.htm [NdÉ].

Séminaire 1 : L'appareil photo numérique : qu'apporte-t-il de nouveau ?

Henri Maître (Télécom ParisTech), à l'ENS Lyon, le 28 janvier 2015

Le rôle exceptionnel pris par la photographie au sein de la société dans cette dernière décennie nous interpelle : quels sont les éléments techniques qui ont permis cet engouement planétaire qui ne se limite ni à certaines classes d'âge ni à des frontières socio-économiques déterminées ? Bien sûr, la prolifération des capteurs d'image en a été la cause majeure, mais il faut lui associer les facilités de diffusion apportées par les réseaux, qui ont inscrit leur usage dans les pratiques quotidiennes. La présentation a examiné les progrès les plus marquants qui ont permis et accompagné ce développement. Ils concernent bien sûr le capteur numérique, dont l'évolution a été retracée, mais aussi les fonctions ancillaires : mise au point, mesure de la lumière, contrôle des organes fonctionnels permettant une adaptation à la scène et progressivement une automatisation des réglages. Deux éléments sont nouveaux dans l'appareil photo : le (ou les) processeur(s) et le logiciel. La présentation a placé quelques jalons témoignant de leur progrès, en les illustrant abondamment. Elle a montré l'irréversibilité de cette évolution et dégagé des perspectives d'évolution tant pour le matériel que pour les professions qui accompagnent l'industrie de la photographie. Elle a insisté sur le rôle très important que jouent, pour toute l'industrie photographique, les caméras embarquées dans les téléphones mobiles, véritables laboratoires tant techniques que sociologiques des progrès dans ce domaine (*living labs* ?). Elle a mis en valeur une dimension que ces caméras couvrent bien mieux que les appareils professionnels et qui recèle très probablement des marges de progrès considérables pour la photographie future : la capacité de communiquer. C'est ici que se place l'enjeu de la photographie du futur, et les seuls acteurs qui subsisteront seront ceux qui auront anticipé cette évolution d'une photographie inscrite dans le monde connecté dès sa formation.

Cours 2 : De la spécification à la réalisation, au test et à la preuve : les approches formelles

Ce cours a introduit les méthodes formelles de développement et de vérification de programmes, en les reliant aux méthodes de programmation, de test et de validation classiques.

Dès les débuts de l'informatique, la difficulté à faire des programmes justes est apparue comme un problème majeur. Deux approches bien différentes ont été développées : d'une part, le génie logiciel, maintenant bien en place, et, d'autre part, une approche résolument scientifique, introduite par Turing dès 1949. Celle-ci voit un programme comme un objet mathématique sur lequel il faut raisonner mathématiquement. Elle a conduit aux méthodes formelles que nous étudions cette année et la suivante. Celles-ci sont devenues le moyen de choix pour bien spécifier et éviter les bugs, au moins dans les applications où leur coût humain et matériel peut être catastrophique. La vérification formelle a eu une maturation assez lente, d'une part parce qu'elle est intrinsèquement difficile, d'autre part parce que l'industrie ne s'y est vraiment intéressée que récemment. Mais les choses bougent, avec des succès de plus en plus fréquents. Et, ce qui ne gâche rien pour un tel enseignement, les chercheurs français jouent un rôle majeur dans ce domaine en plein essor.

Génie logiciel et vérification formelle partagent deux prérequis essentiels : d'abord, la qualité des spécifications initiales, qui doivent être précises, non contradictoires et complètes, mais sans excès de précision inutile ; ensuite, la qualité des langages de programmation, dont la sémantique doit être précise tout en restant intuitive. Beaucoup de projets industriels échouent encore à cause de leur non-respect de ces contraintes.

Le génie logiciel classique écrit les spécifications en langage habituel et fournit des outils de traçabilité permettant de relier spécifications et réalisation. Pour la validation du résultat, il s'appuie sur des revues de code et surtout sur des tests, ce qui pose deux problèmes majeurs : il est difficile de mesurer ce que les tests vérifient réellement, et une campagne de tests n'apporte aucune information sur ce qui n'est pas testé. Mais des systèmes de génération de tests aléatoires dirigés permettent d'obtenir des résultats étonnants.

À l'opposé, les méthodes formelles écrivent les spécifications en langage mathématique, le seul langage rigoureux dont nous disposions réellement, et font aussi progresser ce langage. Les systèmes modernes de typage des programmes permettent de détecter des erreurs dès les premières passes de compilation. Les meilleurs d'entre eux sont directement issus des recherches en sémantique de la programmation, elles-mêmes directement liées à la vérification formelle. Pour aller plus loin, on cherche à remplacer ou à compléter les tests dynamiques de validation par des preuves statiques là encore mathématiques, aidées par des systèmes de vérification formelle plus ou moins automatisés. Au contraire des tests, la vérification formelle dit tout sur les propriétés à valider : prouvées vraies, elles le sont en toutes circonstances ; détectées comme fausses, les outils permettent souvent de construire des tests les invalidant et de découvrir ainsi la source de l'erreur. Mais la preuve, techniquement plus difficile que le test, demande une formation particulière. Et elle ne constitue pas forcément une panacée car certaines propriétés comme l'arrêt d'un programme ne sont pas prouvables en général (bien que les choses s'améliorent en pratique).

Après avoir détaillé les problématiques ci-dessus, le cours a présenté les styles de méthodes formelles qui sont détaillés dans la suite du cours, ainsi que leurs applications pratiques : assertions, preuves par réécritures fondées sur des sémantiques formelles, interprétation abstraite, vérification logique par assistants de preuve, et vérification automatique de modèles. Un point important de la discussion a été le lien avec la programmation classique. Nous avons vu que trois points de vue assez différents coexistent naturellement, ce qui est une des raisons de la multiplicité des techniques précitées :

1. programmer comme d'habitude et utiliser les outils de vérification formelle pour complémenter les tests et vérifier un certain nombre de propriétés critiques du programme (absence d'erreurs bloquant l'exécution, vérification de prédicats sur les états ou les sorties, etc.). C'est une solution souvent utilisée en conception de circuits électroniques ou en logiciel embarqué ;

2. à l'opposé, abandonner les méthodes classiques en intégrant dans un formalisme logique unique l'ensemble du chemin allant des spécifications formelles au code exécutable, tout en réalisant en permanence des preuves automatiques ou guidées manuellement de la correction de la réalisation par rapport aux spécifications. Cette façon de faire très ambitieuse, généralement fondée sur des assistants de preuve en logique formelle, est utilisée à des degrés divers pour des applications en transports, compilation, systèmes d'exploitation, algorithmes distribués, sécurité informatique, etc. ;

3. entre les deux, utiliser des méthodes intermédiaires comme l'interprétation abstraite et la vérification de modèles, dans lesquelles l'objet vérifié est un modèle plus ou moins abstrait de l'application. On se focalise alors sur le traitement des points difficiles ou dangereux de la spécification ou des programmes en abstrayant leurs détails non pertinents. Cette approche permet de réduire considérablement la taille de la vérification, sans toutefois toujours débusquer le diable qui peut se nicher dans les détails de la réalisation. Elle est souvent privilégiée pour les applications parallèles, distribuées ou temps-réels, ainsi que pour la sécurité informatique.

Séminaire 2 : L'Internet des objets, une révolution à ne pas manquer

Joseph Sifakis (École polytechnique fédérale de Lausanne), le 4 mars 2015

L'Internet des objets (en anglais *Internet of Things* ou *IoT*) est né de la convergence entre les technologies des systèmes embarqués et des réseaux : il intègre des objets informatisés dans le réseau internet existant, éventuellement augmenté de sous-réseaux spécifiques accessibles de façon uniforme à travers les protocoles internet standard. Équipés de capteurs et d'actionneurs, ces objets informatisés communicants peuvent être de types variés : *pacemakers*, biopuces implantées dans le corps, automobiles, équipements de domotique intelligente, etc. Le traitement de l'information se fait de deux façons : d'une part par des microprocesseurs et logiciels embarqués, avec des contraintes fortes de réponse en temps-réel ; d'autre part à travers le réseau, en implémentant dans le nuage des algorithmes de prise de décision permettant de répondre rapidement aux événements, d'effectuer des prédictions sur les comportements à venir et d'optimiser les ressources.

L'exposé a présenté la vision qui sous-tend l'Internet du futur et étudié sa faisabilité réelle. Il a explicité des défis majeurs quant à notre capacité à concevoir et à construire des systèmes couplant matériels et logiciels de façon sûre et optimale, qui ne pourront être résolus que par des techniques rigoureuses de spécifications et de réalisation de systèmes. Les problèmes scientifiques à résoudre sont le lien entre la physique et le calcul, le développement de systèmes par assemblage de composants et l'adaptabilité des systèmes leur permettant de fonctionner même en présence de conditions imprévues.

La vision d'avenir fournie par l'Internet des objets conduira à des impacts sociétaux, techniques et scientifiques considérables. En particulier, elle contribuera à revigorer l'informatique elle-même et à l'enrichir de nouvelles fondations scientifiques.

Cours 3 : Les méthodes générales : assertions, réécriture, interprétation abstraite, logiques et assistants de preuve

Dans sa note visionnaire de 1949, A. Turing a introduit la notion d'*assertion*, qui associe un prédicat logique à un point de contrôle d'un programme, ainsi que l'utilisation d'ordres bien fondés (sans chaîne infinie descendante) pour montrer sa terminaison. L'approche par assertions a été étendue et perfectionnée par Floyd, Hoare, Dijkstra et bien d'autres pour aboutir à une théorie et à une pratique complètes de la définition logique des langages séquentiels impératifs et de la

vérification de leurs programmes. Elle a ensuite été étendue en celle de *contrat* (*assume* / *guarantee*) à respecter par les fonctions ou modules d'un programme.

En 1963, J. McCarthy a introduit l'idée bien différente d'utiliser la réécriture de termes comme outil applicable à la fois pour l'optimisation de programmes et leur vérification formelle. Cette approche a eu une longue descendance dans les systèmes de vérification (Boyer&More, ACL2, PVS, Key, ProVerif, etc.). Elle a eu des succès remarquables en circuits, en avionique, en sécurité, etc.

La troisième approche historique est celle de la sémantique dénotationnelle, introduite par Scott et Strachey vers 1970. Un programme y est interprété comme une fonction dans un espace topologique ordonné par un ordre d'information et les fonctions sont rendues totales par l'ajout explicite d'éléments indéfinis. Une théorie générale du point fixe dans ces espaces permet d'interpréter de façon uniforme les boucles, la récursion et la programmation fonctionnelle d'ordre supérieur. Dès 1970, cette théorie a été à l'origine du premier assistant de preuve de programmes, LCF (Milner *et al.*). Mais le traitement explicite de l'indéfini, trop compliqué, a été ensuite abandonné par les assistants de preuve. Le grand succès pratique de la sémantique dénotationnelle est la vérification par interprétation abstraite créée par P. et R. Cousot en 1977. Ici les assertions portent non plus sur les valeurs exactes, mais sur des abstractions de celles-ci, comme la preuve par 9 le fait pour la multiplication. De nombreux domaines abstraits ont été développés, ainsi que des algorithmes de combinaison de ces domaines et d'accélération des calculs de points fixes. Développée industriellement, l'interprétation abstraite a permis de vérifier des propriétés critiques de gros programmes, comme l'absence d'erreurs à l'exécution dans le code de pilotage de l'Airbus A380. Elle permet aussi l'évaluation du temps de calcul maximal de logiciels embarqués, et est de plus en plus incorporée dans d'autres types de systèmes de vérification.

La quatrième approche traitée dans le cours est la vérification par assistants de preuves logiques. On y traduit le problème de vérification informatique en un problème purement logique, et on fournit une aide à l'organisation de grandes preuves à travers un système de tactiques et d'interactions homme-machine, augmenté d'automatisations partielles pour des sous-domaines spécifiques. Le cours a présenté les logiques de premier ordre, qui utilisent le calcul des prédicats augmenté par la théorie des ensembles (Rodin pour Event-B, etc.) ou par la logique temporelle (TLA+). L'atelier B de J.-R. Abrial a été utilisé pour la spécification, la programmation et la vérification formelle de logiciels critiques pour la conduite de systèmes ferroviaires (la ligne 14 du métro parisien, etc.). TLA+ est une référence pour la preuve des algorithmes parallèles asynchrones des systèmes distribués.

Séminaire 3 : Utilisation des méthodes formelles pour la sécurisation de systèmes complexes : une avancée industrielle

Dominique Bolignano (Prove&Run), le 11 mars 2015

L'équipe de Prove&Run a réalisé la preuve complète d'un micro-noyau, Minix V3, d'une complexité comparable à celle du noyau SeL4 par l'équipe du NICTA – qui était une première mondiale. L'approche a pris en compte de nouvelles contraintes pour la généralisation de ce genre de preuve en environnement industriel. L'objectif est de permettre l'utilisation à grande échelle des méthodes formelles pour le

développement de composants critiques, les méthodes expérimentales actuelles ne suffisant pas pour cela.

Prove&Run a décidé de développer un nouveau langage et une nouvelle approche, en s'appuyant sur l'état de l'art en méthodes formelles, mais en renonçant à certains traits considérés comme des « *must* » dans le monde académique mais trop coûteux en contexte industriel actuel : ordre supérieur, programmation fonctionnelle pure, etc.

La présentation a décrit le travail de preuve et de génération réalisé sur Minix, puis le processus conduisant à la compréhension des besoins spécifiques. Ce dernier s'appuie sur l'expérience de l'orateur lors de ces trois dernières décennies. À titre d'exemple, il a décrit pourquoi il lui semble important de séparer contrôle et donnée. Il a aussi décrit comment permettre aux développeurs classiques habitués aux logiciels de très grande qualité de retrouver leur intuition et leur compréhension intime du fonctionnement de leur logiciel dans ce nouveau contexte. Il a terminé par les perspectives de développement à grande échelle fondé sur les méthodes formelles.

Cours 4 : Prouver les programmes : de l'ordre supérieur à Coq

Ce cours a complété le précédent en présentant les assistants de preuve fondés sur les logiques d'ordre supérieur (où l'on peut aussi quantifier sur les prédicats) et sur le lambda-calcul déjà vu en détail dans le cours 2009-2010. Nous avons d'abord montré pourquoi l'ordre supérieur est plus expressif que l'ordre un. Par exemple, un seul axiome standard suffit pour prouver une propriété par récurrence, puisqu'on peut quantifier universellement le prédicat de travail ; en logique de premier ordre, la récurrence demande une infinité d'axiomes engendrés par des schémas. L'inconvénient de l'ordre supérieur est une plus grande complexité de la logique et de certains algorithmes.

Le cours a brièvement présenté les systèmes HOL de Gordon, HOL-Light de Harrison, PVS de Shankar, Owre et Rushby, et Isabelle de Paulson, ce dernier étant un métasystème logique dans lequel plusieurs logiques peuvent être définies. Ces systèmes ont permis des preuves majeures : celle de la correction de l'arithmétique du Pentium par Harrison (Intel) après le fameux bug de division du Pentium Pro en 1994 ; celle preuve du micronoyau seL4 par Klein *et al.* (NICTA Sydney) avec Isabelle, etc.

Le cours a ensuite présenté avec plus de détails le système français Coq, récipiendaire de grandes récompenses récentes. Coq est fondé sur le calcul des constructions CoC de G. Huet et T. Coquand (1984) et sa version inductive CiC développée par C. Paulin et T. Coquand. Coq intègre complètement calcul et preuve dans un formalisme très riche, développant considérablement des idées de de Bruijn (Automath, 1967), Girard (SystemF, 1972), etc.

Son langage de programmation Gallina est un langage typé d'ordre supérieur, dont les types peuvent eux-mêmes être des fonctions. Le typage est décidable, et il garantit que tous les calculs d'un terme bien typé se terminent sur une forme normale unique. La programmation est facilitée par la couche inductive, qui permet des définitions récursives à terminaison garantie. Grâce à la mise en pratique de la correspondance de Curry-Howard entre calculs et preuves, il n'y a plus vraiment de différence entre calculer et prouver. Un programme a pour type sa spécification

alors qu'une preuve a pour type le théorème qu'elle démontre, de façon absolument uniforme. Les preuves sont elles-mêmes des fonctions standard du calcul. Enfin, une fois un programme écrit et prouvé en Coq, on peut en extraire automatiquement un code objet exécutable écrit en Caml, correct par construction et efficace. Tout cela donne à Coq une richesse considérable à la fois pour spécifier, programmer, prouver et exporter, quatre activités complètement intégrées.

Le cours s'est terminé par la présentation de deux succès majeurs de Coq : le compilateur certifié CompCert de Leroy, seul compilateur C ayant passé le filtre des tests aléatoires présenté dans le cours du 4 mars, et la preuve complètement formelle de grands théorèmes mathématiques par Gonthier *et al.*, avec en particulier le fameux théorème de Feit-Thompson dont la preuve originale n'occupe pas moins de 250 pages de lourdes mathématiques.

Séminaire 4 : Langages et systèmes pour la preuve interactive

Christine Paulin (LRI, Université Paris-Sud), le 18 mars 2015

Vérifier que des systèmes informatiques ont le comportement attendu est une tâche complexe. L'expérience montre en effet que ces systèmes contiennent en général des erreurs plus ou moins critiques pour l'utilisateur et plus ou moins difficiles à corriger pour le concepteur. De nombreux travaux visent à proposer des formalismes spécialisés pour modéliser le système et le comportement souhaité ainsi que des méthodes et algorithmes pour réellement prouver son adéquation.

Plusieurs questions se posent alors sur la correction des méthodes proposées : a-t-on effectivement modélisé le bon programme et le bon comportement ? La méthode et son implantation sont-elles correctes ? Sont-elles suffisamment puissantes ? Ces questions concernent directement les industriels qui développent des codes informatiques critiques et doivent garantir leur bon comportement en apportant suffisamment de justifications à l'organisme de certification.

Notre conviction qu'une méthode est correcte doit reposer sur une interprétation du problème en termes mathématiques et un raisonnement logique. Cependant, l'intrinsèque complexité des modèles mathématiques des programmes rend nécessaire un traitement informatisé de ces modèles.

À côté d'outils spécifiques à l'analyse de programmes s'est développée depuis les années 1970 une classe d'outils qui traitent des théories mathématiques générales. Les langages de description de théories et les formalismes de preuve varient suivant les systèmes. Le séminaire a présenté et justifié plusieurs approches possibles, de la logique du premier ordre à la théorie des types. Via l'exemple de l'assistant de preuve Coq, il a illustré l'état de l'art de ce qui peut être réalisé actuellement en termes de mathématiques formalisées sur ordinateur.

Cours 5 : La vérification de modèles (*model checking*)

Ce cours a terminé la présentation générale des méthodes de vérification formelle par la vérification de modèles, plus connue sous son nom anglais de *model-checking*. Bien différente des précédentes, elle s'intéresse essentiellement aux programmes d'états finis, ceux dont on peut conceptuellement dérouler complètement toutes les exécutions possibles en temps et espace finis, et s'applique très bien aux programmes parallèles. Le parallélisme peut y être synchrone comme dans les circuits ou les

langages synchrones présentés en 2012-2014, ou asynchrone comme dans les protocoles de communication, les réseaux et les algorithmes distribués. Le *model-checking* est né au début des années 1980, quasi-simultanément à deux endroits : à Grenoble (Queille et Sifakis avec le système CESAR et sa logique temporelle) et aux États-Unis (Clarke et Emerson avec la logique temporelle CTL et le système EMC). Le prix Turing 2007 à été attribué à Clarke, Emerson et Sifakis pour ces travaux. Ils s'appuyaient eux-mêmes sur les travaux de Pnueli (prix Turing 1996) sur la logique temporelle. Le *model-checking* constitue certainement la méthode formelle la plus utilisée dans l'industrie, en particulier dans la conception assistée par ordinateurs de circuits électroniques.

L'idée de base est de construire le graphe de toutes les exécutions possibles d'un programme ou circuit, qu'on appelle son modèle. Ce modèle peut prendre la forme d'une structure de Kripke (logicien et philosophe de la logique modale), graphe où les états sont étiquetés par des prédicats élémentaires, ou d'une structure de transitions, où les étiquettes sont portées par les transitions entre états. Une fois construit, le modèle devient indépendant du programme qui l'a engendré. Pour raisonner sur un modèle, on utilise des logiques temporelles qui définissent les propriétés à vérifier et caractérisent l'environnement d'exécution à l'aide de propriétés élémentaires des états ou transitions et de quantificateurs sur les états ou les chemins du modèle. On peut ainsi exprimer et vérifier des propriétés de sûreté (absence de bugs), comme « à aucun moment l'ascenseur ne peut voyager la porte ouverte », d'absence de blocages de l'exécution, ou de vivacité, comme « l'ascenseur finira par répondre à toutes les demandes des passagers », ou encore « chaque processus obtiendra infiniment souvent la ressource partagée s'il la demande infiniment souvent ».

Le cours a d'abord présenté la logique CTL*, qui permet d'imbriquer arbitrairement les quantifications d'états et de chemins sur les structures de Kripke. Mais cette logique très expressive est difficile à utiliser et les calculs y sont d'un coût prohibitif. Deux sous-logiques importantes sont : LTL (*linear temporal logic*), qui ne quantifie pas sur les états et considère seulement des traces linéaires, et CTL, logique arborescente qui quantifie sur les chemins mais avec des restrictions par rapport à CTL*. D'expressivités différentes, ces deux logiques ont leurs avantages et leurs inconvénients. LTL est bien adaptée pour la vérification de propriétés de vivacité, comme le montre Lamport (prix Turing 2014) avec son système TLA+. Mais elle ne permet pas d'exprimer des prédicats impliquant l'existence de calculs particuliers que peut exprimer CTL.

La modélisation par systèmes de transitions, systématisée par R. Milner (prix Turing 1992) dans l'étude des calculs de processus communicants, permet de mieux composer les exécutions de ces processus parallèles. L'équivalence comportementale par bisimulation permet de comparer finement ce que peuvent faire ou ne pas faire les processus. La réduction par bisimulation fournit une alternative très intéressante et intuitive aux logiques temporelles pour la vérification de modèles, en particulier en liaison avec les langages synchrones.

Une dernière façon de conduire la vérification de modèles est de remplacer les formules temporelles par des programmes observateurs, qui prennent en entrée les entrées et les sorties du programme à vérifier et ont pour charge d'envoyer un signal de bug s'ils détectent une anomalie. Cette méthode plus simple que la logique temporelle est idéale pour les langages synchrones Esterel, Lustre et SCADE étudiés les années précédentes.

Séminaire 5 : Prouver la sécurité informatique : la logique à la rescousse

Véronique Cortier (LORIA, CNRS), le 25 mars 2015

Régler un achat, accéder à ses comptes et même voter, toutes ces actions peuvent désormais s'effectuer « en ligne ». Comment s'assurer alors qu'aucun intrus ne viendra perturber ces opérations sensibles ? La sécurité de ce type de transactions est garantie par des programmes informatiques appelés « protocoles de sécurité ». Ces protocoles reposent sur des techniques de cryptographie : chiffrement, signature électronique, hachage, etc. Cependant, la cryptographie ne suffit pas : il peut parfois suffire d'intervertir deux messages pour accéder à un service de manière illégitime. Il est donc nécessaire d'analyser la structure logique des protocoles, combinée à la cryptographie utilisée.

L'analyse formelle des protocoles de sécurité fait appel à de nombreuses techniques de vérification : résolution, unification, théorie équationnelle, systèmes de contraintes. L'objectif de ce séminaire a été de dessiner un panorama des techniques utilisées et de décrire comment plusieurs protocoles ont pu être corrigés grâce à ces techniques d'analyse.

Cours 6 : Vérification et optimisation booléennes d'automates et circuits

Ce dernier cours a introduit les méthodes implicites de manipulation de systèmes de transitions, à travers les méthodes de calcul booléen utilisées à la fois pour la vérification formelle et pour l'optimisation de circuits électroniques et de programmes d'états finis.

Ces méthodes ont révolutionné le domaine en permettant la vérification de systèmes dont le calcul explicite des états et transitions est impossible car la taille des formules manipulées par les méthodes implicites est largement indépendante de celle des systèmes qu'ils décrivent. Le cours a expliqué les codages booléens d'ensembles, de relations et de fonctions, et montré comment calculer l'image directe et l'image inverse de sous-ensembles par des fonctions. Il a ensuite étudié les codages booléens d'automates déterministes et non-déterministes, ainsi que leurs implémentations en circuits électroniques. Le circuit digital canoniquement associé à un automate non-déterministe est, lui, déterministe comme tous les circuits combinatoirement acycliques, ce qui montre clairement que le qualificatif *non-déterministe* est particulièrement mal choisi : en vérification booléenne comme en optimisation de circuits, il est inutile de déterminiser les automates, et c'est souvent nuisible à cause de l'explosion exponentielle que la déterminisation peut produire. Nous avons ensuite montré comment la vérification formelle de propriétés de sûreté définies par des observateurs se réduit au calcul des états accessibles, et comment effectuer ce calcul de manière implicite. Pour finir, nous avons introduit la première structure fondamentale du calcul booléen, les *binary decision diagrams* ou BDD, développés par R. Bryant au milieu des années 1980 (et indépendamment par J.-P. Billion chez Bull en France) et expliqué pourquoi ils permettent de faire les calculs nécessaires au passage à la grande échelle ; mais leurs limitations sont inévitables car le calcul booléen est NP-complet. Les BDD seront étudiés plus en profondeur dans le cours de 2015-2016.

Pour terminer, nous avons montré que le codage booléen permet de réaliser des optimisations très efficaces des circuits engendrés par les programmes Esterel. La structure du langage source et la façon d'y programmer les applications sont essentielles pour la qualité de l'optimisation finale : c'est grâce à l'interaction de la séquence, du parallélisme et de la préemption hiérarchique des comportements que les circuits engendrés par Esterel sont systématiquement meilleurs que ceux programmés et optimisés par les méthodes classiques, au moins en ce qui concerne leurs parties contrôle.

Séminaire 6 : Spécification, construction et vérification de programmes : le parcours d'une pensée scientifique sur une trentaine d'années

Jean-Raymond Abrial, le 1er avril 2015

Cette présentation a été celle d'un chercheur très novateur et influent mais se disant vieillissant, qui a porté un regard historique sur les trente dernières années de son travail. Il y a deux sortes de chercheurs : les prolifiques et les monomaniaques. J.-R. Abrial fait partie de la seconde catégorie, car il a toujours pratiqué le même genre d'investigations, à savoir la spécification et la construction vérifiée de systèmes informatisés.

Ce travail, autant théorique que pratique, s'est concrétisé au cours de toutes ces années dans trois formalismes voisins : Z, B et Event-B (pour lequel il n'est pas, loin de là, le seul contributeur). J.-R. Abrial a tenté d'expliciter comment les idées que contiennent ces formalismes et les outils correspondants ont lentement émergé de façon parfois erratique, tout en précisant les multiples influences qui ont participé à cette évolution. En particulier, il a montré comment plusieurs réalisations industrielles ont permis de progresser dans ce domaine, mais aussi souligné les échecs et parfois les rejets de la part de certaines communautés tant universitaires qu'industrielles. Pour finir, J.-R. Abrial a proposé quelques réflexions et approches pour le financement de recherches telles que celle-ci.

Conférences données à l'extérieur

Cours et conférences du Collège de France à l'étranger

29.04.2014 : Dans le cadre de la collaboration Collège de France / université d'Athènes, « Le temps, de la physique à l'informatique et à la musique ».

29.09.2014 : Dans le même cadre, conférence au lycée franco-hellénique Eugène Delacroix, Athènes, « Pourquoi l'informatique change le monde ».

Conférences académiques invitées

18.09.2014 : Colloque « Digital Intelligence », Nantes, « Why informatics generates mental inversions ».

01.10.2014 : Colloque « La musique en temps réel », université de Strasbourg, « Le temps et les événements en informatique et en musique : comment en parler précisément ? ».

06.11.2014 : Colloque « Le calcul et le temps », université Jean Moulin, Lyon, « Le temps, de la physique à l'informatique et à la musique ».

12.11.2014 : Symposium « Les menaces cybernétiques », École militaire, Paris, « Réflexions sur la sécurité informatique ».

28.11.2014 : Colloque « L'influence des théories scientifiques sur le renouvellement des formes dans la musique contemporaine », CMDC / GREAM / université de Strasbourg, Paris, « Temps informatique, temps musical ».

11.12.2014 : Journée des prix scientifiques en informatique et mathématiques appliquées, Collège de France, « La révolution numérique dans les sciences ».

16.12.2014: IFIP Working Groups 1.9/2.15 Meeting, Collège de France, « Revisiting finite-state systems design and verification ».

10.12.2014 : Colloque « Les scientifiques et l'épistémologie : la rationalité scientifique aujourd'hui », Fondation Del Duca, Paris, « La spécificité des sciences de l'information ».

27.01.2015 : Journées « Avancées récentes autour des neurosciences computationnelles », École normale supérieure de Lyon, « Questions sur les neurosciences ».

29.01.2015 : Journée d'étude « Écriture, technique et pensée », ENSSIB Lyon, « Écrire et mécaniser le raisonnement ».

06.02.2015 : Séminaire à Neurospin, CEA Saclay, « Le modèle de calcul synchrone et ses applications potentielles en neurosciences ».

09.03.2015 : Cérémonie des nouveaux élus de l'Académie des technologies, Palais de la découverte, Paris, « Art et informatique ».

10.03.3015 : Cérémonie des remise des prix de « La main à la pâte », Académie des sciences, « La passion de la science : comprendre et construire ».

11.03.2015 : Evénement « Future@SystemX », IRT SystemX, Saclay, « Comment faire des choses intelligentes avec des machines stupides ».

24.03.2015 : Cérémonie d'accueil des nouveaux entrants du CNRS, Paris, « La passion de la recherche ».

26.03.2015 : Institut des Hautes Études pour la science et la technologie (IHEST), Paris, « De l'informatique temps-réel à l'Internet des objets ».

31.03.2015 : Séminaire de l'Inter-section des applications des sciences, Académie des sciences, Paris, « Informatique et musique ».

09.04.2015 : Séminaire du département de physique de l'École normale supérieure, Paris, « Le nouveau rôle du temps en informatique ».

12.05.2015 : Institut des sciences de la communication, Paris, « Pour une renaissance de la leçon de choses ».

13.05.2015 : Académie des technologies, Paris, « Méthodes formelles de conception et vérification pour les circuits et logiciels embarqués critiques ».

19.05.2015 : Colloque « À la recherche du temps », Académie des sciences, Paris, « Les temps et les événements en informatique ».

28.05.2015 : Institut d'histoire et de philosophie des sciences et techniques (IHPST), « Le traitement du temps et des événements, de la physique à l'informatique ».

30.05.2015 : Colloque de médecine « High-Tech périphérique », Bonifacio, « La révolution informatique ».

11.06.2015 : Conférences aux élèves de 1ère année de l'École polytechnique, Palaiseau, « Le nouveau rôle du temps en informatique ».

16.06.2015 : Colloque CRNS « Innovations globales et universelles », Paris, « L'innovation informatique dans les sciences et technologies ».

18.06.2015 : Séminaire « Code source », UPMC, Paris, « D'Esterel v5 à Esterel v7 ».

25.06.2015 : Colloque dédié à Martin Abadi pour l'attribution du doctorat *honoris causa* de l'École normale supérieure de Cachan, « Determinism, Electricity, and Intuitionism »

Autres conférences

13.09.2014 : Colloque « Art, complexité, cerveau », Mouans-Sartoux, « Calculer, combien ça coûte ? La complexité algorithmique ».

24.09.2015 : Institut des Hautes Études pour la science et la technologie (IHEST), Paris, « Sciences et conscience chez les Shadoks ».

23.10.2014 : Colloque « Pop-mathématiques », semaine de la pop-philosophie, Marseille, « Le temps, de la physique à l'informatique et à la musique ».

29.10.2014 : Haut-commissariat à l'énergie atomique, Paris, « Sciences et conscience chez les Shadoks ».

18.11.2014 : Séminaire de la société Sophiacom, Saint Germain-en-Laye, « Hop and hiphop : Multitier web orchestration ».

17.01.2015 : Conférence à l'association « L'Aphyllante », Aigaliers, « Sciences et conscience chez les Shadoks ».

20.01.2015 : Participation à la table ronde « Faut-il réinventer la sécurité ? », Forum international sur la cyber-sécurité FIC2015, Lille.

19.05.2015 : Conférence à la bibliothèque de Montreuil, « L'informatique, de la révolution technique à la révolution mentale ».

26.05.2015 : Conférence au séminaire directorial de la MACSF, Paris, « Le temps, de la physique à l'informatique ».

09.06-2015 : Conférence à l'université populaire d'Amiens, « Homme et ordinateurs, comment les harmoniser ? ».

26.06.2015 : Séminaires de recherche du laboratoire Servier, Suresnes, « La révolution informatique dans les sciences ».

Émissions de radio

26.09.2014 : « Le club de la tête au Carré », France Inter.

23.10.2014 : Interview à Radio-Grenouille, Marseille.

13.11.2014 : « La tête au carré », France Inter.

19.12.2014 : « Autour de la question », Radio France internationale, « Pourquoi la science informatique ? ».

07.03.2015 : « La conversation scientifique », France Culture.

02.05.2015 : « Les savanturiers », France Culture.

Activités de recherche

Louis Mandel, qui avait rejoint ma chaire en septembre 2013 comme maître de conférences au Collège de France, a quitté le Collège de France en septembre 2014. Ce même mois a été marqué par l'embauche de Lionel Rieg sur un poste d'ATER Collège de France. Formé à la logique formelle et à l'utilisation de Coq, il a travaillé avec moi sur l'implémentation en Coq des sémantiques formelles de mon langage Esterel, décrites en détail dans mon cours de 2012-2013 : la sémantique comportementale, la sémantique constructive et la sémantique par transformation d'états. Tous les théorèmes reliant ces sémantiques ont été prouvés dans Coq. Cette partie majeure du travail de formalisation véritable d'Esterel sera publiée dans un article en préparation. Le travail continuera en 2015-2016 par l'implémentation

d'une sémantique opérationnelle non encore publiée, qui décrit une exécution efficace des programmes, puis de la traduction des programmes Esterel en circuits qui fonde la compilation du langage, et enfin de la preuve d'équivalence de ces nouvelles sémantiques avec la sémantique constructive qui donne la vraie définition du langage mais est opérationnellement inefficace. Cette chaîne justifiera formellement l'ensemble des travaux permettant de passer progressivement d'une sémantique abstraite à une implémentation concrète et efficace.

Tout ceci me permettra de faire paraître enfin mon livre *The Constructive Semantics of Pure Esterel*, disponible en version préliminaire sur le web depuis 1998 (et abondamment cité), mais que je n'ai jamais voulu publier vraiment avant que les preuves des différents théorèmes ne soient vérifiées en machine de façon incontestable. Il est à noter que cela évitera aux relecteurs du livre de vérifier eux-mêmes ces preuves techniques et assez complexes. Nous chercherons aussi à écrire en Coq un vrai compilateur d'Esterel en circuits, puis à extraire automatiquement de sa preuve de correction un code exécutable écrit en Caml. Ceci fournirait la première preuve d'un compilateur de langage parallèle formellement défini, dans l'axe de celle de CompCert par Leroy présentée dans le cours 4 ci-dessus pour C, qui est lui un langage séquentiel conceptuellement plus simple.

Cette année, grâce à l'obtention d'un financement d'un réseau ANR nommé Chronos, j'ai organisé une série de séminaires sur le temps sous tous ses aspects, qui a rassemblé régulièrement une vingtaine de chercheurs de disciplines différentes : circuits, logiciels temps réels, informatique musicale, neurosciences, physique, géosciences, etc. Ce séminaire a pour vocation de discuter les travaux en cours et les sujets mal compris, fort nombreux dans le domaine. À travers une dizaine de séances, il a permis de commencer à rapprocher utilement les points de vue des différentes disciplines.

En 2014, en collaboration avec Manuel Serrano de l'Inria Sophia-Antipolis, j'avais commencé un travail sur l'utilisation de la programmation synchrone pour l'orchestration d'activités Web construites par composition de services Web existants. Ce sujet a stagné à cause de l'abandon d'un thésard commun ; il va redémarrer sur une base nouvelle en 2015-2016 avec un nouveau thésard.

Autres activités

Je suis président du Conseil d'enseignement et de recherches de l'École polytechnique, président du conseil scientifique de l'école d'informatique et électronique ESIEE Paris–Marne-la-Vallée, membre du conseil d'administration et du conseil scientifique de l'Institut pour la recherche et la coordination acoustique et musicale (IRCAM), et membre du bureau de l'association européenne Informatics Europe.

J'ai présidé le jury des grands prix Inria / Académie des sciences, participé au jury du prix Milner délivré par la Royal Society, l'Académie des sciences et la Leopoldina, ainsi qu'à plusieurs jurys de l'Académie des sciences, dont celui des prix Irène Joliot-Curie pour la promotion de la place des femmes dans la science et la technologie et celui des bourses françaises L'Oréal-UNESCO pour les femmes et la science.