Réponse aux questions de l'année

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages

http://www.college-de-france.fr/site/gerard-berry
gerard.berry@college-de-france.fr

Cours 9, 9 avril 2014

COLLÈGE
DE FRANCE
1530

Qualité des spécifications

- Peut-on faire des spécifications vraiment non ambiguës, en particulier dans le monde industriel et dans les applications entremêlant homme et machine ? (exemple de l'accident du Paris-Rio).
- Certainement pas en toute généralité. Mais c'est une bonne idée de travailler autant que possible à partir de composants élémentaires bien spécifiés et bien vérifiés, et surtout avec des outils de spécification et de programmation formellement clairs et efficaces. Il y a de vraies réussites: programmation et preuve de RER et métros en B (J-R. Abrial), programmation d'avions et autres en SCADE, etc, vérification du noyau d'OS SEL4 en Isabelle (G. Heiser), du compilateur C CompCert en Coq, (X. Leroy), d'algorithmes distribués en TLA+ (L. Lamport), etc.
- C'est aussi une bonne idée de faire des analyses critiques fouillées de systèmes et logiciels, avec des équipes formées de gens aux passés et aux connaissances fort différentes. Cela se fait en avionique, nucléaire, et d'autres domaines (sécurité par exemple).

Logique ternaire

- J’ai le souvenir d’une logique ternaire pour les relais : oui, non et indéfini = encours de basculement. Cette logique "floue" a-t-elle été développée ou prise en compte dans vos travaux ?

- Absolument, sous le nom de logique de Scott (ou de Kleene).
- Cf. le cours des 26 mars et 2 avril 2014, où l’indéfini signifie « pas encore stabilisé ».
- Cf. aussi le cours du 9 décembre 2009, où la logique ternaire et plus généralement les logiques ordonnées sont utilisées pour la sémantique dénotationnelle des langages de programmation.
- Cf. enfin les articles et livres du transparent suivant sur les relations fondamentales entre sémantiques dénotationnelles et opérationnelles

Full Abstraction for Sequential Languages: the State of the Art
G. Berry, P-L. Curien et J-J. Lévy. In *Algebraic Methods in Semantics*, Cambridge University Press (1985) 89-132.

Theory and Practice of Sequential Algorithms: the Kernel of the Programming Language CDS
G. Berry et P-L. Curien. In *Algebraic Methods in Semantics*, Cambridge University Press (1985) 35-88.

Sequential Algorithms on Concrete Data Structures
G. Berry et P-L. Curien. *Theoretical Computer Science*, vol. 20 (1982) 265-321.

Tolérance aux fautes en logiciel

- Peut-on construire des logiciels tolérant les fautes comme on le fait pour les matériels ?

- Sujet très débattu, mais encore peu clair. Le matériel est sujet à des pannes aléatoires, mais pas le logiciel. Les tests de cohérence permanents en cours de route peuvent être une solution, mais pas toujours. Cf. bug d'Ariane 501 : son logiciel des gyrolasers a décidé de fermer boutique à cause d'un *overflow* alors que tout était en parfait état de marche.

- Airbus utilise deux chaînes logicielles / matérielles distinctes, avec vote. Mais la notion de vote n'est pas triviale car les calculs doivent être asynchrones pour éviter de dépendre d'une seule horloge. Et des erreurs communes peuvent venir de la spécification commune. Il y a encore des points de vue divers sur le sujet....

Circuits constructifs

- Logique intuitionniste UN : est-elle implémentée en Coq et disponible sur le Web? Je suppose qu'il y a une limite de taille à ce qui peut être traité avec un outil aussi général mais que les exemples du cours ne sont pas trop gros.
- Pas encore, mais ça pourrait se faire dans les années qui viennent ! Coq n'a pas de problème de taille, cf. la preuve de Feyt-Thompson par Gonthier et. al. (250 pages de maths lourdes mises en Coq...)
- Y a t'il une chaîne logicielle "schéma -> solveur SMT" pour les exemples de circuit ? Quelle part est disponible en open source ?
- Oui dans le compilateur Esterel v5 avec le système SIS de UC Berkeley, que j'espère remettre enligne bientôt (binaire libre ou open source ?). Mais il utilise les BDDs, mieux que SAT ici car les BDDs permettent l'optimisation dans la foulée. Voir cependant le travail de Namjoshi et Kurshan pour l'analyse de constructivité en SAT:
- Kedar S. Namjoshi, Robert P. Kurshan: *Efficient Analysis of Cyclic Definitions*. CAV 1999 [PDF]

Validité du modèle synchrone

- Je me pose des questions sur le fondement de votre démarche qui postule que le temps d'exécution est nul ou tout au moins négligeable. Y a-t-il une réflexion conduite pour anticiper les conséquences de cas qui pourraient apparaitre dans des conditions que je ne sais exclure de façon définitive : n'y a-t-il AUCUN cas où cette condition essentielle ne serait pas remplie (boucle logique non détectée ou provoquée par une défaillance – éventuellement fugitive – d'une partie du matériel) ?

- La validité de l'hypothèse synchrone se traite au niveau système, et les cas de non-respect aussi. Ils peuvent être tolérables pour des raisons d'automatique, ou dans des architectures de type LTTA (cf. séminaire d'Albert Benveniste le 5 mars 214), ou intolérables mais détectables par des chiens de gardes avec passage en mode dégradé, etc.

- Dans son domaine d'application, l'hypothèse synchrone simplifie en fait le problème par rapport à d'autres approches de type scheduling dynamique. Elle permet le calcul du WCET (*Worst Case Execution Time*), impossible avec des tâches sur OS préemptif par exemple.

Les drivers, un point faible des systèmes?

- Observant des fonctionnements étonnants de mes liaisons *WiFi* sous Windows XP, je me demande si vos méthodes et outils sont utilisés dans le développement des drivers.

- Cela faisait partie de nos objectifs, car les drivers sont effectivement des points faibles récurrents des systèmes. Mais leur statut dans l'industrie est hélas assez bas, ils sont souvent négligés malgré leur importance. Pas de clients clairement exprimés.

- Cependant, les choses ont commencé à changer quand les constructeurs ont compris le danger des bugs de drivers, y compris pour la sécurité. Voir par exemple les travaux sur la vérification formelle de drivers par Tom Ball *et. al.* (système SLAM) chez Microsoft, et les travaux de Gilles Muller (UPMC) et Julia Lawall (Inria) sur les drivers et APIs Linux et leur maintenance automatique.

Compilation des langages synchrones

- Il existe plusieurs approches pour représenter un système "synchrone" qui deviendra du matériel intégré avec du logiciel exécuté sur une partie de ce matériel. Quels sont les algorithmes / méthodes pour passer d'un modèle type Esterel ou Ptolemy à une description matérielle (synthétisable ou synthétisée) et logicielle ?

- Question hélas pas simple. Pour Esterel, il existe des algorithmes permettant de passer d'un programme soit à une implémentation logicielle, soit à un implémentation matérielle, cf. cours du 15 janvier. Mais passer à un mélange efficace de matériel et de logiciel n'est pas actuellement possible. Malgré les efforts mondiaux, aucune méthode automatique de « Hardware / Software Codesign » n'a encore fait ses preuves. Ptides a cette ambition, voir cours du 2 avril 2014.
- Pour les algorithmes variés de compilation, voir *Compiling Esterel*, D. Potop, S. Edwards et G. Berry. Springer, 2007.

Systèmes et langages

- Pourquoi, alors qu'il existe Ptolemy et Esterel , les architectes du matériel utilisent-ils SystemC ?
- Pour Esterel, sujet traité en grand dans le cours du 22 janvier, cf Web !
  - prééminence de Verilog / VHDL / C / C++, effet “Winner Take All”,
  - réticence des architectes et développeurs à changer de langage, opération jugée trop risquée dans une industrie tendue où toutes les parties du circuit doivent être prêtes en même temps
  - outillage industriel développé sur ces langages, intérêt des fournisseurs à ne faire que des changements incrémentaux
  - difficulté d’embaucher des ingénieurs déjà formés, etc.
- Pour SystemC :
  - ni Esterel ni Ptolemy ne sont vraiment capables de traiter les simulations TLM (Transaction Level Modeling), cf. séminaires de Matthieu Moy et Laurent Maillet-Contoz, 29/01/2014. C’est à cela que sert principalement SystemC.
  - Les compilateurs SystemC→circuits s’améliorent aussi, au moins pour le data-flow

Synchronisation molle

La matière molle a pu gagner ses lettres de noblesse à la fin du XXe siècle ; envisagez-vous qu'un destin similaire soit possible pour la synchronisation molle dans les circuits? D'ailleurs, dans les exemples du cours, même les réseaux GALS finissent par être complètement synchrones, et les synchroniseurs que vous avez présentés semblent toujours actifs.

Hmmm, la comparaison me paraît franchement osée, et je ne la suivrai pas.

Non, les réseaux GALS ne finissent pas complètement synchrones, surtout bien sûr si les horloges de leurs composants sont différentes, ce qui est en général le cas.

Par ailleurs, les synchroniseurs sont loin d'être toujours actifs, car la plupart des composants d'un circuit (par exemple de téléphone) sont éteints dès que possible. Les synchroniseurs ne sont allumés que lorsque l'horloge réceptrice du circuit l'est.

Synchronisations transitoires

A l'inverse, les synchronies enregistrées dans le cerveau (hors sommeil, épilepsie,..) sont extrêmement locales et transitoires. Est-ce qu'une telle synchronie "quasi-fluide", ou polychronie, est utilisée dans des circuits électroniques ?

Tout à fait. La plupart des composants d’un téléphone dorment presque tout le temps: le codec mp3 si on n’écoute pas de musique, l’accélérateur vidéo si on ne regarde pas de films, etc. Les synchronisations sont ainsi transitoires et réduites au minimum pour économiser l’énergie.

Comparaison avec les réseaux neuronaux

Les réseaux neuronaux sont caractérisés par une redondance qui compense les aléas de la biologie. Est-il possible d'utiliser des populations de processeurs asynchrones pour pallier la métastabilité à faible coût énergétique?

Vaste question sur laquelle ma compétence est limitée. Mais attention à plusieurs points rendant la comparaison difficile :

- On ne sait pas encore vraiment comment fonctionnent les réseaux neuronaux (cf. séminaire de R. Brette)
- Contrairement aux neurones, les composants électroniques sont encore d'une très grande fiabilité, donc le problème n'est pas le même
- On utilise déjà des populations de processeurs asynchrones pour les calculs (GALS, multicœurs), mais pas pour des raisons de redondance, et la métastabilité est un grand problème bien résolu maintenant
- Mais les choses pourraient changer à l'avenir, avec des outils de calculs bien différents sur lesquels je ne suis pas compétent (*memristors* par exemple).

Questions asynchrones / synchrones

- Quelle est la relation entre les modèles de comportement de circuit et les articles de Lamport sur la synchronisation par mémoire partagée?
- Lamport travaille essentiellement sur les questions asynchrones, conceptuellement différentes des questions synchrones. On peut coder l'asynchrone en synchrone en mettant des horloges partout ou en utilisant les théorèmes d'expansion, mais on n'y gagne pas forcément.

- Est ce qu'on peut par exemple prouver la faisabilité des hypothèses standard en programmation parallèle en mémoire partagée (cf. par exemple l'article de Lamport *A Fast Mutual Exclusion Algorithm*). Je me pose la même question à propos des multicœurs ou GPU.
- Même réponse molle.... Les multicœurs sont un bon exemple de synchrone / asynchrone où la difficulté principale est dans l'asynchrone, il faut d'autres cours !

- Quelle logique d'ensemble se dégage des travaux récents reposant sur des modèles formels du parallélisme distribué? Comment s'insère le développement en cours TLA+ (Lamport) ? Les résultats classiques (cf. livres de N. Lynch , Raynal) restent ils valables ou doivent ils être revus en regard des résultats de la modélisation en logique formelle?
- Ces questions demanderaient un cours complet sur l'asynchrone, hors de ma portée personnelle !
- Mais les résultats classiques restent valides, et TLA+ est certainement une des logiques permettant de les prouver (cf. aussi *Separation Logic* pour la mémoire partagée, que je connais beaucoup moins...).

Tolérance aux fautes en logiciel

- Peut-on construire des logiciels tolérant les fautes comme on le fait pour les matériels?
- Sujet très débattu, mais peu clair. Le matériel est sujet à des pannes aléatoires, mais pas le logiciel.
- Voir réponses précédentes à une question similaire.

Questions

- Ne faudrait-il pas améliorer la sensibilité aux questions de sûreté logicielle dans l'industrie, cf. cours du 22 janvier ?
- Oh que si, par exemple dans l'automobile (cf exemple Toyota).

- Quid du nombre astronomique de langages différents ? Une synthèse est possible alors que l'on se préoccupe de l'accès des enfants au « codage » ? Pourrait-on à la fois faire acquérir les bases de la programmation (en n'oubliant pas d'initier aux "évènements") et agir en assemblant des modules préexistants?

- Excellente question, mais trop vaste pour aujourd'hui. Les sujets liés aux langages informatiques restent encore étonnamment difficiles, car les problèmes à résoudre sont vastes et les idées nombreuses. On y travaille pour l'éducation, sans solution claire encore.
- L'unification des langues naturelles n'avance pas beaucoup non plus, à part avec la généralisation du mauvais anglais dans la recherche ☺

Et pour toutes les questions concernant la vérification formelle des programmes, rendez-vous l'année prochaine

Cours 2014-2015:

La preuve de programmes: pourquoi, quand, comment

(première saison)