Langages de Preuve

FMF 2014 - Preuve de modèle, preuve de programmes

Christine Paulin-Mohring

Université Paris Sud & INRIA Saclay - Île-de-France

4 février 2014

Plan

Motivations

Faire des systèmes qui “marchent”

- Expliciter le sens de “çà marche !” / “çà marche pas”
  - modéliser (spécifier)
- Comment faire pour que cela marche ?
  - comprendre
  - mettre en œuvre
- Expliquer pourquoi cela marche
  - se convaincre
  - convaincre les autres (documentation, certification)

Logique

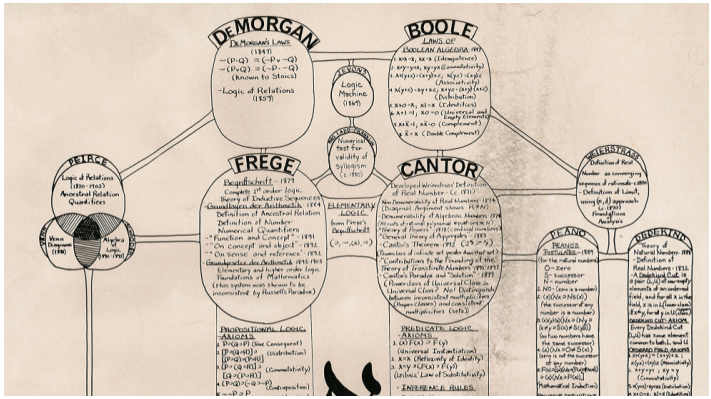
- Mécanismes “universels” pour décrire le raisonnement, c’est-à-dire des enchainements qui préservent ce qui est vrai
- Langage
  - `vrai`, `faux`,
  - `non`, `et`, `ou`, `implique`, `équivalent...`
  - `pour tout`, `il existe`
- Des hypothèses (modèle, environnement) et des conclusions
- Schémas de raisonnement inattaquables
  - de $A \Rightarrow B$ et A on déduit B
  - de $\neg\neg A$ on déduit A

La logique appliquée aux systèmes

- la logique pour modéliser les systèmes
  - le programme
  - les composants du système
  - l'environnement
  - le comportement attendu
- utilisation de formalismes ad-hoc
  - langages spécialisés (circuits, logique temporelle,. . .)
  - des équations différentielles
  - des probabilités
- socle commun : comprendre et faire communiquer les niveaux

$$
\begin{array}{lcl}
M, V \models Xv & \Leftrightarrow & M, (V_2, V_3, \ldots) \models v \\
M, V \models Fv & \Leftrightarrow & \exists i \geq 1, M, (V_i, V_{i+1}, \ldots) \models v \\
M, V \models Gv & \Leftrightarrow & \forall i \geq 1, M, (V_i, V_{i+1}, \ldots) \models v
\end{array}
$$

Objectif de la présentation



- nombreux formalismes différents
  - comme pour les langages de programmation. . .
- des propriétés et philosophies différentes
- des outils variés répondant à des objectifs différents
- importance du langage
  - il ne suffit pas de prouver
  - il faut aussi que ce que l'on prouve soit en adéquation avec ce que l'on veut

comprendre, positionner, choisir

Plan

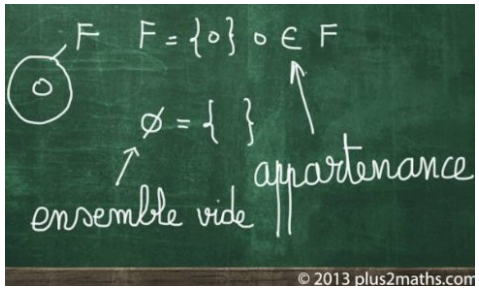
Logique du premier ordre : langage

- symboles pour représenter les objets : c, $\mathtt{f}(t_1, \ldots, t_n)$
- prédicats pour représenter les propriétés : $\mathtt{P}(t_1, \ldots, t_n)$
- connecteurs logiques $\neg A$, $A \vee B$, $A \wedge B$, $A \Rightarrow B$
- quantifications sur les objets $\exists x, P$, $\forall x, P$

Logique du premier ordre : propriétés

- un cadre bien étudié
  - modèles, démonstration,
  - complétude, incomplétude
- validité en général indécidable
  - quelques fragments décidables
    - calcul propositionnel
    - arithmétique linéaire, réels
  - des outils opérationnels : Prolog, Vampire, Zenon, . . .
- théorie : ensemble de formules qui décrit une classe de modèles
  - difficile de cerner 1 modèle, voir impossible en général
  - on peut ajouter de nouvelles propriétés
  - comment préserver la cohérence ?

La théorie des ensembles



- cadre officiel des mathématiciens
- ensemble relativement limité d'axiomes (schémas d'axiome)
- bien étudiée
- l'univers mathématique se reconstruit à partir des objets de base
  - le chemin peut être long...
  - de nombreux niveaux d'abstraction pour cacher la complexité
- ensembles : structurent les objets,
  l'appartenance est une propriété logique (preuve)
  $$\forall x \in \mathbb{R}, \exists y \in \mathbb{N}, x \leq y < x + 1$$
- outils qui s'appuient sur ce cadre
  - Atelier B, Mizar, Isabelle/ZF, ...

Théories typées

- ensembles assez simples pour que l'appartenance soit syntaxique
- structurer les objets :
  - base : $\texttt{bool}$, $\texttt{nat}$, $\texttt{int}$
  - constructeurs : $\alpha \times \beta$, $\texttt{list}\,\alpha$
  - fonctions : $\alpha \to \beta$
- limites de ce que l'on peut vérifier et inférer
  - $t : \mathbb{R}^*$?
  - polymorphisme : $1 : \texttt{nat}$, $1 : \texttt{int}$, $1 : \mathbb{R}$, $[] : \texttt{list}\,\alpha$
  - analyse plus fine
    - interprétation abstraite : outils sémantiques
    - approche déductive

Des objets fonctionnels

- opérations essentielles pour les entiers : $+1$, $+$, $*$
  permet de coder des suites d'entiers
- relation fonctionnelle : $R(\vec{x}, y)$ (laborieux)
- fonction primitive récursive $f_p(\vec{x}, n)$

$$f_p(\vec{x}, 0) = h_0(\vec{x}) \qquad f_p(\vec{x}, n+1) = h_1(\vec{x}, n, f_p(\vec{x}, n))$$

- peut s'étendre à d'autres structures de données
  - spécifications algébriques
- les axiomes introduits cassent-ils la cohérence ?

Des constructions de programmes pour les objets

- internaliser la construction (Système T de Gödel) :
  - f_p est caractérisé par h_0 et h_1
  - nouvel opérateur R
  - $R(h_0, h_1, n)$ nouveau terme
  - passage à des objets fonctionnels
  - possibilité d'un codage au premier ordre (combinateurs)
- vers un (mini)-langage de programmation pour les objets

$$f_p(\vec{x}, 0) = h_0(\vec{x}) \qquad f_p(\vec{x}, n) = \textbf{if } n = 0 \textbf{ then } h_0(\vec{x})$$
$$f_p(\vec{x}, n+1) = h_1(\vec{x}, n, f_p(\vec{x}, n)) \qquad \textbf{else } h_1(\vec{x}, n, f_p(\vec{x}, n-1))$$

- exemples
  - $x + y = \textbf{if } y = 0 \textbf{ then } x \textbf{ else } (x + (y-1))+1$
  - $x \times y = \textbf{if } y = 0 \textbf{ then } 0 \textbf{ else } x + (x \times (y-1))$
  - $x^y = \textbf{if } y = 0 \textbf{ then } 1 \textbf{ else } x \times x^{(y-1)}$
- peut s'étendre à d'autres structures de données

Des programmes comme objets de première classe

- problème de terminaison
  - $\forall n, n = 0 \vee n \neq 0$
  - $0 \neq 1$
  - $f(x) = \textbf{if } f(x) = 0 \textbf{ then } 1 \textbf{ else } 0$
  - $f(x) = 0 \Leftrightarrow f(x) \neq 0$
- solutions possibles
  - domaine avec un objet indéterminé $\perp$
  - logique de termes partiels (quantifications sur les objets finis)
  - langages typés, récursivité restreinte (pas Turing complet)
- objets fonctionnels *purs*
  - $p = p$ non valide si p a des effets
  - $(x{+}{+} == x{+}{+})$, `random == random`
- traduire les constructions impératives (expliciter les effets)

Lien entre raisonnement et calcul

- raisonnement équationnel

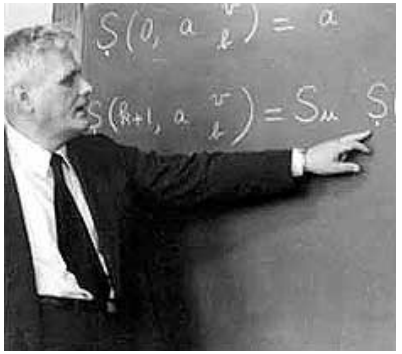
$$\frac{f(t) = u \qquad C[u]}{C[f(t)]}$$

- la preuve de $f(t) = u$ peut être complexe
- la remplacer si possible par du calcul
  - réecriture
  - réduction
  - compilation
- le calcul sert de justification
  - $2 + 2 = 4$
  - $\texttt{fact}(6) = 720$

Premier ordre versus ordre supérieur

- modèle dans lequel on veut raisonner
  - symboles (fonctions, prédicats)
  - propriétés
- au premier ordre : théorie axiomatique
  - théorie puissante (ensembles) et reconstruction (complexe)
  - théorie ad-hoc (cohérence, adéquation)
- prédicats et objets sont deux catégories différentes

Logique d'ordre supérieur (HOL)



- les prédicats deviennent des objets de la logique
  - possibilité de les manipuler explicitement
  - quantification sur les prédicats

$$\mathbb{N}(n) \equiv \forall P, P(0) \wedge (\forall x, P(x) \Rightarrow P(x+1)) \Rightarrow P(n)$$

- constructions de base élémentaires
  - univers infini
  - fonctions (types simples)
  - implication, quantification universelle
- possibilité de reconstruction de notions complexes
- outils : HOL, PVS, Coq, ...

Définition inductive

- plus petite relation verifiant certaines propriétés
- exemples
  - propriétés élémentaires : $\dfrac{}{0 \in \mathbb{N}} \quad \dfrac{x \in \mathbb{N}}{x+1 \in \mathbb{N}} \qquad \dfrac{}{x \leq x} \quad \dfrac{x \leq y}{x \leq y+1}$
  - transitions : $\dfrac{}{\mathtt{pos}(0,0)} \quad \dfrac{\mathtt{pos}(x,y)}{\mathtt{pos}(x+1,y+2)} \quad \dfrac{\mathtt{pos}(x,y)}{\mathtt{pos}(x+2,y+1)}$
  - descriptions sémantiques, systèmes logiques...
- mécanisme déclaratif
  - propriétés de la relation à définir (positivité) : constructeurs
  - existence d'une plus petite petite relation : principe d'induction
- permet de définir des relations partielles, non-déterministes...
- ne met pas en cause la cohérence de la théorie

Théorie des types versus HOL

Des approches similaires pour de nombreux développements

- Théorie des types (Coq)
  - le langage des objets permet de décrire des algorithmes complexes
  - le calcul est intégré à la théorie
  - $p : \texttt{bool}$ peut être évalué en vrai ou faux
  - $P : \texttt{nat} \to \texttt{bool}$ est une propriété décidable
  - cadre logique exploratoire
- HOL
  - le calcul est limité
  - les fonctions complexes sont représentées par leur graphe
  - $p : \texttt{bool}$ est une propriété quelconque qui ne peut pas forcément être évaluée
  - cadre logique stable

Récapitulatif

	1er ordre	ZF	arithmétique	HOL
nouveaux objets	symboles	codés	programmés	définis
nouveaux prédicats	symboles	définis	programmés	définis
étendre la théorie	axiomes	inutile	peu utile	inutile
automatisation	possible	limitée	calcul	limitée
interaction	limitée			essentielle

Plan

Systèmes de preuve

- langage pour les propositions
  - constructions de haut niveau
  - typage, vérification
- système logique $\Gamma \vdash P$
- environnement de preuve
  - vérification de cohérence
  - calcul
  - automatisation
  - interaction
  - bibliothèques

Démonstration automatique premier ordre

Langages et théories du premier ordre

- SAT-solver : variables booléennes
- SMT-solver : prédicats sur des objets interprétés
  - entiers, bit vecteurs, tableau, listes . . .
  - alt-ergo, Barcelogic, CVC, VeriT, Z3, . . .
- premier ordre (quantificateurs)
  - Prover9, Vampire, SPASS, Metis . . .

Caractéristiques

- automatique
- génération de traces de preuve
- back-end pour résoudre des problèmes logiques purs
  - model-checking symbolique
- traite des formules encodées (formes clausales)
- enjeu : gestion efficace des quantificateurs et des théories

Assistants de preuve

- environnement correspondant à une théorie générale
  - arithmétique (ACL2)
  - théorie des ensembles (Isabelle/ZF, Mizar)
  - logique d'ordre supérieur (HOL, Isabelle/HOL, PVS)
  - théorie des types (Coq, Agda, . . .)
- introduction de définitions (langage avancé)
- énoncé de théorèmes, preuves
  - raisonnement automatique par étapes (ACL2, Mizar)
  - raisonnement interactif par tactiques

Assistants de preuve (caractéristiques)

- traite des énoncés logiques de haut niveau
- noyau de confiance raisonnablement limité
- outil de conception et d'étude de systèmes complexes
- mathématiques avancées :
  - 4 couleurs, Feit-Thompson, ...
  - réels, équations différentielles
- enjeux
  - intégrer plus d'automatisation (tactiques, bibliothèques, ...)
  - constructions de haut-niveau (implicite)

Lier preuves automatiques et interactives

- les démonstrateurs automatiques produisent des traces qui seront vérifiées efficacement dans l'assistant
- implantation dans les assistants de stratégies automatiques
  - méthodes par réflexion
  - méthodes par trace (réduire la complexité)
- partage de théories entre systèmes logiques différents
- construction de modèles

Preuves de programmes

- produire un code exécutable spécifié et prouvé correct
- l'utilisateur écrit un programme
  - langage de description de calculs
  - outils de génération de code efficace
- spécifications
  - refléter les programmes dans la logique
- preuves
  - conditions pour que le programme satisfasse la spécification

Différentes approches

- logique : mini-langage de programmation fonctionnel pur

$$\forall x, P(x) \Rightarrow Q(x, f(x))$$

- sémantique axiomatique
  - programme impératif
  - génération d'obligations de preuve
    - logique de Hoare, wp, logique dynamique...
    - lié à la sémantique des programmes (partage)
- raffinement
  - les programmes abstraits comme spécification à des programmes concrets
- annotations
  - implicites : erreurs d'exécution
  - génération automatique

Programmation fonctionnelle

- ACL2, Coq, (HOL,PVS) intègrent des langages de programmation fonctionnels purs
- les programmes sont des objets de la logique
- possibilité d'exécuter ces programmes
- traits impératifs
  - se codent de manière fonctionnelle (monades)
  - optimisations à l'exécution
- utilisation pour la certification d'outils symboliques
  - sémantique : (JavaCard, bytecode verifier)
  - Coq en Coq
  - compilateurs : CompCert
  - SMT-solver : alt-ergo
  - interprétation abstraite
  - générateurs d'obligation de preuves

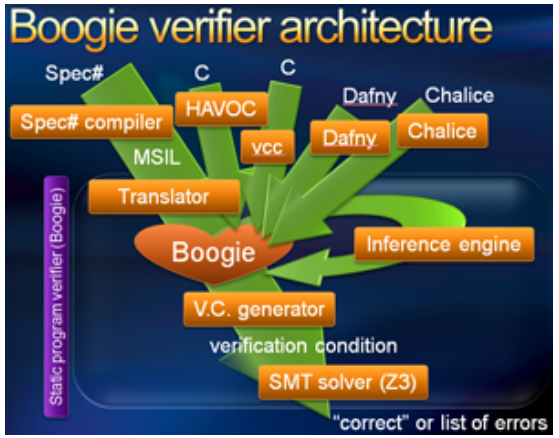
Atelier B

- formules logiques en théorie des ensembles
- programmes décrits sous forme de machines plus ou moins abstraites
- développement par raffinement et combinaison de machines
- génération de code implicite
- preuve d'obligations par application de règles
- utilisé pour des applications industrielles critiques (Meteor)
- cadre logique et discipline de développement logiciel

Langages de programmation annotés

- ESC/Java2 : Extended Static Checker, simplify, JML
- Key tool : JML et OCL, preuves automatiques et interactives
- Spec# : préservation d'invariants pour C#
- Frama-C
  - programmes C annotés
  - langage de spécification ACSL
  - différents modules d'analyse, dont la vérification déductive
  - explicitation de la sémantique
    - différents niveaux de modèle mémoire

Plateformes de preuve de programme



- Boogie
- Why3 `http://why3.lri.fr`
  - descriptions de théories modulaires
  - multi-sortes, fonctions, types algébriques, définitions inductives ...
  - traduction vers des prouveurs variés
    - solveurs spécialisées (gappa)
    - prouveurs automatiques (SMT/TPTP)
    - prouveurs interactifs (Coq, Isabelle, PVS)
  - langage de programmation avec annotations
    - fonctionnel, références, exceptions
    - utile pour décrire la sémantique de langages plus complexes
    - génération d'obligations

Plan

Conclusion

- une large gamme d'outils performants pour faire des preuves sur machine
- utilisation directe ou en back-end d'autres outils
- des approches collaboratives entre outils

Un peu de pub !

Mondes numériques

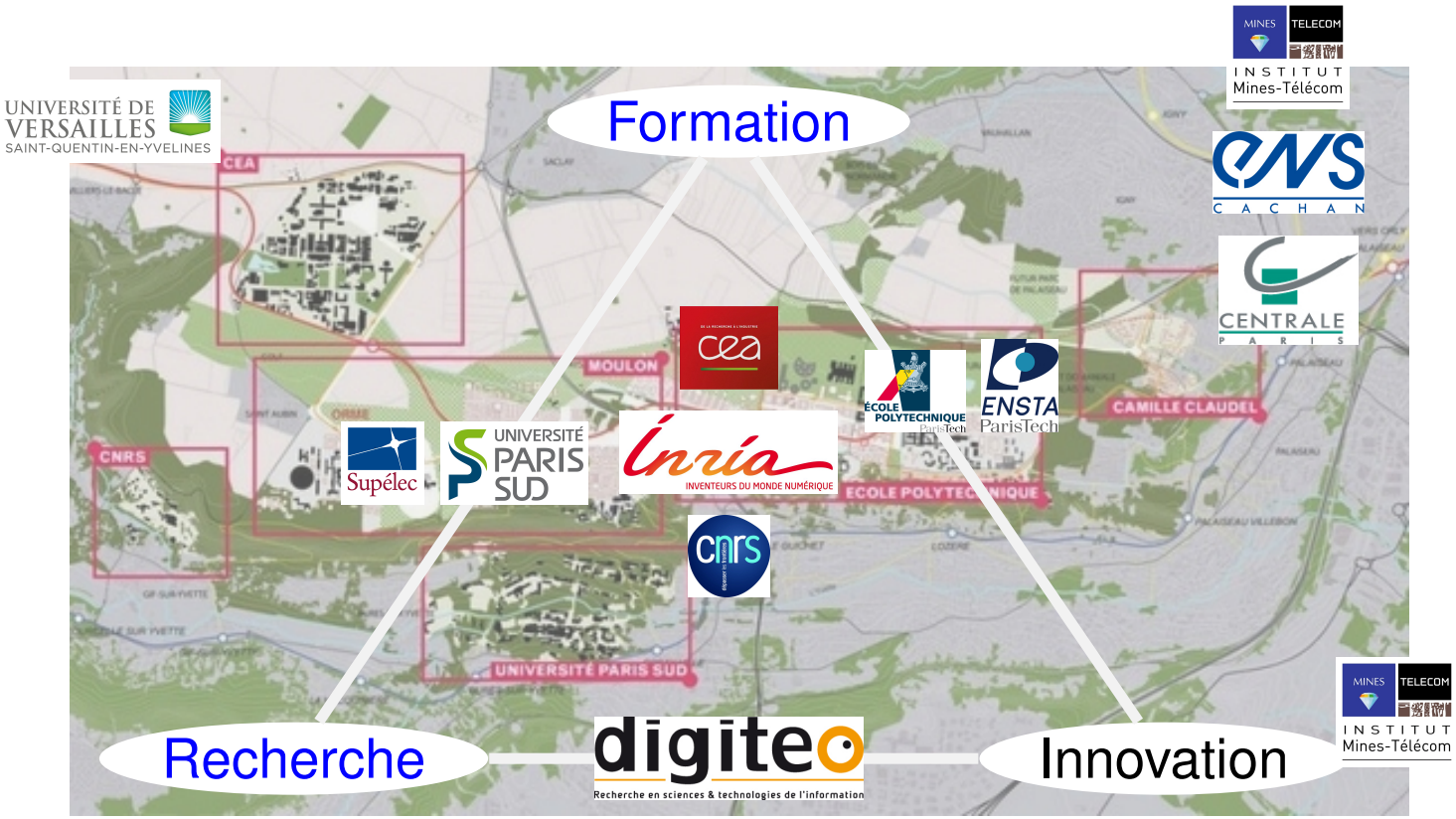
programmes, données et architectures distribués

http://labex-digicosme.fr

- Laboratoire d'excellence 

9 M€ pour 8 ans (900k€/ an)
- Participe à l'Initiative d'excellence (Idex) Paris-Saclay pour construire la future Université Paris–Saclay

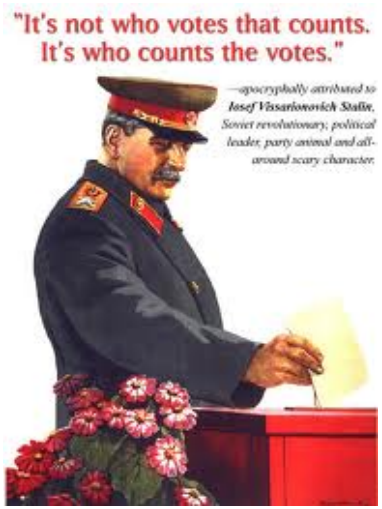
Objectifs



- 14 laboratoires informatique & communications à Paris-Saclay
- environ 340 chercheurs permanents
- 3 axes de recherche principaux

Activités de recherche

- SciLex (Ph. Schnoebelen) : explosion des communications dans des environnements incertains (attaques et pannes)
  1. safe and reusable distributed programs : security, mobility
  2. continuous versus discrete systems : models and verification
  3. from high-level to low-level certification : combination and cooperation of verification techniques



Activités de recherche

- DATASENSE (Michèle Sebag) : méthodes innovantes pour traiter les données : interaction avec les utilisateurs, les décideurs, les ingénieurs et les scientifiques pour réaliser des objectifs stratégiques
  1. scalable, expressive and secure tools for large-scale data
  2. making sense of complex, heterogeneous data and their usages
  3. machine learning : meta-learning and multi-task
  4. distributed decision making
  5. interaction and visualization (EquipeX Digiscope)

Activités de recherche

- COMEX (Pierre Duhamel) : organiser de très grands réseaux hétérogènes, concevoir des systèmes de communications efficaces.
  1. network information theory and coding
  2. network centric design of distributed architectures
  3. terminal centric design of distributed networks

Questions ?