Logiciel certifié et théories typées: Récursion et Induction sur des Objets de Type Fonctionnel

Joëlle Despeyroux

INRIA
Sophia Antipolis

13 mai 2002

pré-exposé HDR - Concours DR2

OUTLINE

Sémantique Naturelle et Preuves

Sémantique Naturelle:

Notion proposée par Gilles Kahn, Thierry et Joëlle Despeyroux [CDDK86, Des86] comme une variante de la SOS (Sémantique Opérationelle Structurelle) proposée par Gordon Plotkin.

Exemple: Imp:

$$
\begin{array}{lll}
exp \;\; e ::= & x \mid n \mid e + e' \mid e - e' \mid \\
 & true \mid false \mid e = e' \mid e \wedge e' \mid e \vee e' \\
stmt \; s ::= & skip \mid x := e \mid s; s' \mid \\
 & if \; e \; then \; s \; else \; s' \mid while \; e \; do \; s
\end{array}
$$

Sémantique Naturelle et Preuves *(continued)*

Évaluation en SOS: $par : \dfrac{p \hookrightarrow^{a} p' \quad q \hookrightarrow^{b} q'}{p//q \hookrightarrow^{ab} p'//q'}$

Évaluation en SN:

$$Judgements:$$
$$env \vdash exp \hookrightarrow value; \; env \vdash stmt \hookrightarrow env;$$

$$int : \; \rho \vdash int\ n \hookrightarrow n$$
$$var : \; \rho \vdash var\ x \hookrightarrow v \quad (\rho(x) = v)$$

$$assign : \; \frac{\rho \vdash e \hookrightarrow v}{\rho \vdash x := e \hookrightarrow \rho[x = v]}$$

$$seq : \; \frac{\rho \vdash s \hookrightarrow \rho' \quad \rho' \vdash s' \hookrightarrow \rho''}{\rho \vdash s; s' \hookrightarrow \rho''}$$

$$if_t : \; \frac{\rho \vdash e \hookrightarrow True \quad \rho \vdash s \hookrightarrow \rho'}{\rho \vdash if\ e\ then\ s\ else\ s' \hookrightarrow \rho'}$$

$$if_f : \; \frac{\rho \vdash e \hookrightarrow False \quad \rho \vdash s' \hookrightarrow \rho'}{\rho \vdash if\ e\ then\ s\ else\ s' \hookrightarrow \rho'}$$

Induction mathématique

Principe d'induction mathématique sur $\mathbb{N}$:

$$\begin{array}{l}(P(0)\ \& \\ (\forall m \in nat.\ P(m) \Rightarrow P(S(m))) \\ \Rightarrow \forall n \in nat.\ P(n)\end{array}$$

Principe d'induction structurelle sur exp:

$$\begin{array}{l}(\forall x \in var.\ P(x))\ \& \\ (\forall e, e' \in exp.\ P(e)\ \&\ P(e') \Rightarrow P(e + e'))\ \& \\ (\forall e, e' \in exp.\ P(e)\ \&\ P(e') \Rightarrow P(e * e')) \\ \Rightarrow \forall e \in exp.\ P(e)\end{array}$$

Théorème.
La sémantique dynamique de Imp\while est déterministe:

$$\forall \rho, e, v, v'.\ (\rho \vdash e \hookrightarrow v)\ \&\ (\rho \vdash e \hookrightarrow v') \Rightarrow v = v'$$

$\boxed{\text{Induction mathématique } \textit{(continued)}}$

Induction bien fondée:

Soit $<$ une relation bien fondée sur un ensemble A. Soit P une propriété sur les éléments de A. On a:
$(\forall x \in A.\ (\forall y \in A.\ y < x \Rightarrow P(y)) \Rightarrow P(x))$
$\equiv \forall a \in A.\ P(a)$.

Induction sur la structure d'une preuve:

$$
\begin{array}{l}
(\forall \rho, n.\ P(p : (\rho \vdash int\ n \hookrightarrow n)))\ \& \\
(\forall \rho, x, v\ tq\ \rho(x) = v.\ P(p : (\rho \vdash var\ x \hookrightarrow v)))\ \& \\
(\forall \rho, e, n, m.\ \forall p_1 : (\rho \vdash e \hookrightarrow n). \\
\qquad \forall e', n'.\ \forall p_2 : (\rho \vdash e' \hookrightarrow n'). \\
\qquad P(p_1)\ \&\ P(p_2)\ \&\ (n + n') = m \Rightarrow \\
\qquad P(p : (\rho \vdash e + e' \hookrightarrow m))) \\
\Rightarrow \forall p : (\rho \vdash e \hookrightarrow v).\ P(p)
\end{array}
$$

Théories Typées (TT)

Ex: the Calculus of Inductive Constructions (CIC).

The CIC is a Pure Type System (a typed λ-calculus) with 4 constants (Set, Prop, Type_set, Type), axioms $Set : Type\_set$ and $Prop : Type$, and all dependant products.

Notations:

the abstraction $\lambda x : t.a$ is written [x:t]a,

the dependent product type $\forall x : t.a$ is written (x:t)a,

the non-dependent product type $A \rightarrow B$ is simply written $A \rightarrow B$.

The application of a function f to x is written $(f\ x)$,

the expression $\exists x : t.a$ is written (Ex [x:t]a),

the type of sets is denoted as Set while the type of propositions in denoted as Prop.

Principes d'induction ordinaires en Théories Typées

Curry-Howard iso a logic $\Leftrightarrow$ a typed theory:

1. The inference rule $\dfrac{A_1 \cdots A_n}{B}$ is represented by type $A_1 \to \cdots \to A_n \to B$.

2. A proof of a proposition B is represented by a term having type B.

Inductive types are defined by a list of constructors together with their types. Examples:

```
Inductive nat: Set := 0 : nat | S : nat -> nat.
Inductive list [A:Set] : nat -> Set :=
  nil : (list A 0)
| cons : (n:nat) A -> (list A n) -> (list A (S n)).
```

An inductive definition of a type A generates:

- an inductive principle for A, and

- elimination principles which enable the definition of recursive functions on this type (with case and fixpoint).

Syntaxe abstraite fonctionnelle

Technique de représentation des langages de programmation, proposée par G. Plotkin, R. Harper et F. Honsell (1987), puis F. Pfenning, D. Miller, ...

La *syntaxe abstraite fonctionnelle* ou *syntaxe abstraite d'ordre supérieur* permet de formaliser les notions de variables liées et de substitution d'un langage, en représentant les variables du langage spécifié (le langage objet: ML, Java, ...) par les variables du langage utilisé pour spécifier (le langage méta: CCI/Coq, PVS, ...).

En conséquence, les constructions liantes des langages objets (typiquement les abstractions dans les langages fonctionnels) sont représentées en utilisant l'abstraction du méta-langage.

Cette technique permet également de formaliser les *side conditions* (typiquement: "variable x libre dans les hypothèses") et les changements de contexte dans les règles d'inférence (déchargement d'une hypothèse en Déduction Naturelle).

Elle augmente donc nettement le niveau d'abstraction, et par là le confort et la faisabilité, à la fois de la description des langages de programmation, et des preuves sur ces langages.

Syntaxe abstraite fonctionnelle *(continued)*

Remarque: Dans les développements qui ne suivent pas cette approche (typiquement en de Bruijn): **75 %** des développements concernent des lemmes techniques sans intérêt, parfois difficiles (ex. les preuves de Daniel Hirschkoff: 4 ans de thèse [Hir97]).

Problème: il n'est pas possible de faire un raisonnement par induction, de manière standard, sur une syntaxe abstraite fonctionnelle.

2 solutions:

1) Syntaxe abstraite fonctionnelle non standard dans les théories typées existantes,

2) Proposition de nouvelles théories typées, munis des principes de récursion et d'induction adaptés (notions standards étendues aux termes de type fonctionnel).

Syntaxe abstraite d'ordre supérieur restreint

[DFH95]. Méthode permettant d'utiliser une forme restreinte de la syntaxe abstraite d'ordre deux et de l'implication, conjointement avec l'induction, dans des systèmes comme Coq. de preuves Coq.

Cette syntaxe contient des termes exotiques. On décrit les termes licites par un prédicat *Valid*, qui fournit l'induction désirée.

Les sémantiques des langages objets utilisent un prédicat $subst$, dont l'adéquation doit être démontrée.

Parameter $var : Set.$
Inductive Set L
$= Var : var \to L$
$\mid App : L \to L \to L$
$\mid Lam : (var \to L) \to L.$

$L_n = L \to \cdots \to L \to L$ représente les fonctions de L à n arguments dans L:

Fixpoint $LL\ [L : Set] : nat \to Set :=$
$\lambda n : nat.\ \langle Set\rangle$ Case m of
$(*\ \ 0\ \ *)\ \ L$
$(*\ (S\ n)\ *)\ \lambda n : nat.\ L \to (LL\ n)$ end.

Sémantique fonctionnelle

[DH94]. Idée clef: Les sémantiques sont données sur les termes valides de L_n (*termes fonctionnels*).

Le prédicat $Valid$ fournit exactement l'induction sur les arbres que l'on désire. De plus, ce principe d'induction est plus naturel, et plus général, que les différents principes d'induction connus.

Definition $\mathcal{R}ef = \lambda n : nat.\lambda i \in [0..n\text{-}1].\lambda x_{n\text{-}1}, \ldots, x_0 : var.$
$(Var\ x_i)$
Definition $\mathcal{A}pp = \lambda n : nat.\lambda e, e' : L_n.\lambda x_{n\text{-}1}, \ldots, x_0 : var.$
$(App\ (e\ x_{n\text{-}1}\ \ldots\ x_0)\ (e'\ x_{n\text{-}1}\ \ldots\ x_0))$
Definition $\mathcal{L}am = \lambda n : nat.\lambda e : L_n.\lambda x_{n\text{-}1}, \ldots, x_0 : var.$
$(Lam\ \lambda y : var.\ (e\ y\ x_{n\text{-}1}\ \ldots\ x_0)).$

Inductive Definition $Valid : \forall n : nat.\ L_n \Rightarrow Prop$
$= Valid\_ref : \forall n, i : nat.(i < n) \Rightarrow$
$(Valid\ n\ \mathcal{R}ef_{n,i})$
$\mid Valid\_app : \forall n : nat.\forall e, e' : L_n.$
$(Valid\ n\ e) \Rightarrow (Valid\ n\ e') \Rightarrow (Valid\ n\ (\mathcal{A}pp_n\ e\ e'))$
$\mid Valid\_lam : \forall n : nat.\forall e :\ L_{n+1}.$
$(Valid\ (n+1)\ e) \Rightarrow (Valid\ n\ (\mathcal{L}am_n\ e)).$

Récursion sur des termes de type fonctionnel

Une syntaxe abstraite fonctionnelle, définie de manière naïve, ne forme pas un type inductif. Problème: l'espace des fonctions utilisé pour décrire les arbres de syntaxe abstraite doit être plus restreint que celui (des fonctions primitives récursives) nécessaire à l'utilisation de principes d'induction.

$$\begin{array}{l} L : Set. \\ App : L \to L \to L \\ Lam : (L \to L) \to L. \end{array}$$

D'où la proposition de nouvelles théories typées, basées sur la logique modale (la logique linéaire est aussi possible).

1. [DPS97, DPS01], en collaboration avec Frank Pfenning et Carsten Schürmann (CMU): une extension d'un λ-calcul modal simplement typé par des opérateurs d'itération et de raisonnement par cas, étendus aux objets de type fonctionnel. Preuves de la conservation des types, la forte normalisation et l'extension conservative du système.
2. [DL98, DL00, DL01], dans le cadre d'une thèse (Pierre Leleu, 1998): proposition d'une amélioration de ce système (règles de réduction et non d'évaluation, comprennant l'êta-expansion).

Travaux connexes

- R. Pollack en J. Mc Kinna: Lego en Lego [Pol95].
  Distinction var / par: évite le renommage.
- A. Gordon et T. Melham: 5 axiomes pour l'α-conversion [GM96].
- A. Pitts: Fresh ML [Pitts99-lics].
  Formalisation des preuves "à la main".
- C. Schürmann [Schürmann95].
  1ers pas vers une TT modale avec l'induction - à 2 niveaux.

Induction sur des termes de type fonctionnel

– Prédicats $Valid$.
– Termes d'ordre supérieur:
  $P_{n+1}(e) \Rightarrow P_n(Lam\ e)$
  $P_n(e) \Rightarrow P_n(e') \Rightarrow P_n(App\ e\ e')$

Travaux connexes:

– Raymond McDowell et Dale Miller [MM97].
  Systèmes à 2 niveaux; induction sur les entiers.
– Martin Hofmann [Hof99].
  Preuves de validité de différents principes d'induction.
– Carsten Schürmann. Meta LF.
  Système à 2 niveaux.
– Gordon Plotkin [plo99-lics].
  Models.

Conclusion et Perspectives

Conclusion

– 82 - 87: Sémantique Naturelle et preuves.
– 88 - 91: Système assistant à la preuve [Des90].
– 92 - .: Preuves en Coq et
– Théorie des types: récursion et induction.
– 02 -. : ...

Perspectives

– Programmes sûrs en Java en 2012?
– Bibliothèque de maths certifiées en 2022?
– Les besoins grandissent: cartes à puces, transitions bancaires, (fusées,) avions, voitures, ...
– Moyens humains à l'INRIA: très insuffisants. Thème 2A très petit.
– Labos industriels en forte croissance: CNET, Intel, Microsoft, Trusted Logic, Gemplus...

References

[CDDK86] Dominique Clement, Joëlle Despeyroux, Thierry Despeyroux, and Gilles Kahn. A simple applicative language: Mini-ML. In *proceedings of the conference Lisp and Functional Programming, LFP'86*. ACM Press, August 1986. Also appears as INRIA Research Report RR-529, May 1986.

[Des86] Joëlle Despeyroux. Proof of translation in Natural Semantics. In *proceedings of the first Symp. on Logic In Computer Science, LICS'86*. IEEE Computer Society, June 1986. Also appears as INRIA Research Report RR-514, April 1986.

[Des90] Joëlle Despeyroux. Theo: an interactive proof development system. User's manual. Technical Report RT 116, INRIA, February 1990.

[DFH95] Joëlle Despeyroux, Amy Felty, and André Hirschowitz. Higher-order abstract syntax in Coq. In M. Dezani and G. Plotkin, editors, *proceedings of the TLCA 95 Int. Conference on Typed Lambda Calculi and Applications*, volume 902, pages 124–138. Springer-Verlag LNCS, April 1995.

Références

[DH94] Joëlle Despeyroux and André Hirschowitz. Higher-order syntax and induction in Coq. In *proc. of the LPAR'94 Int. Conference*, volume 822, pages 159–173. Springer-Verlag LNAI, 1994.

[DL98] Joëlle Despeyroux and Pierre Leleu. A modal λ-calcul with iteration and case constructs. In *proceedings of the annual Types for Proofs and Programs seminar*, Springer-Verlag LNCS 1657, March 1998.

[DL00] Joelle Despeyroux and Pierre Leleu. Metatheoretic results for a modal lambda-calculus. *Journal of Functional and Logic Programming (JFLP)*, 2000(1), January 2000.

[DL01] Joelle Despeyroux and Pierre Leleu. Recursion over objects of functional type. *Special issue of MSCS on "Modalities in Type Theory"*, 11(4), August 2001.

[DPS97] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In Philippe de Groote and J. Roger Hindley, editors, *proceedings of the TLCA 97 Int. Conference on Typed Lambda Calculi and Applications, Nancy, France, April 2–4*, pages 147–163. Springer-Verlag LNCS 1210, April 1997.

[DPS01] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. *Theoretical Computer Science (TCS)*, 266(1-2):1–57, September 2001. Extended and revised version of the TLCA'97 paper.

Références

[GM96] Andrew Gordon and Tom Melham. Five axioms of alpha-conversion. In *proceedings of the int. conf. on Theorem Proving in Higher Order Logics, Turku, Finland, August 1996*, Springer-Verlag LNCS ?, 1996.

[Hir97] Daniel Hirschkoff. A full formalization of pi-calculus theory in the Calculus of Constructions. In Elsa Gunter, editor, *proceedings of the International Conference on Theorem Proving in Higher Order Logics*, Murray Hill, New Jersey, August 1997.

[Hof99] Martin Hofmann. Semantical analysis of higher-order abstract syntax. In IEEE Computer Society Press, editor, *Proceedings of the International Conference on Logic In Computer Sciences, LICS*, 1999.

[MM97] Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax: An extended abstract. In *Proc. of LICS'97*, June 1997.

[Pol95] Robert Pollack. A verified typechecker. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 365–380, Edinburgh, Scotland, April 1995. Springer-Verlag LNCS 902.