Systèmes d'aide à la démonstration

Pierre Castéran, LaBRI (Méthodes Formelles)

26 octobre 2006

« En informatique (ou en mathématiques assistées par informatique), un assistant de preuve est un logiciel permettant l'écriture et la vérification de preuves mathématiques, soit sur des théorèmes au sens usuel des mathématiques, soit sur des assertions relatives à l'exécution de programmes informatiques. »(Wikipedia)
Exemples : LCF, HOL, Coq, Isabelle, Nqthm, etc.

Contenu

- Généralités
  - Quels théorèmes ? Quelles preuves ?
  - Quelle confiance leur accorder ?
  - Quelle [non-]automaticité ?
- *Coq*
  - Programmes, théorèmes, et programmes certifiés
  - Faut-il avoir peur des axiomes ?
- Un exemple : Problèmes de terminaison
  - Suites de Goodstein et batailles d'hydres
  - Ordinaux dénombrables
- Conclusions et perspectives

Généralités

- Quels théorèmes ? Quelles preuves ?
- Quelle confiance leur accorder ?
- Quelle [non-]automaticité ?

Quels théorèmes ? Quelles preuves ?

- Écrire la preuve complète d'un programme ou d'un théorème est un travail assez long, mais à ne faire qu'une fois (en principe.)
- Cet effort peut être justifié dans les cas suivants :
  - Logiciel critique (énergie, transport, toute forme de sécurité)
  - Langages de description : automates, grammaires, langages de programmation, protocoles de télécommunication, protocoles cryptographiques,etc.
  - Démonstrations comportant un très grand nombre de cas à étudier, calculs à effectuer (quatre couleurs, conjecture de Kepler),
  - Intérêt pour la notion de démonstration en soi, enseignement de la logique.

Vérification de programmes

- Model-checking : entièrement automatique, mais langage restreint,
- Vérification statique de programmes : (*polyspace, esc/java*) : langage non restreint, mais vérification non complète en général,
- assistants de preuves :
  - premier ordre, (*B*)
  - ordre supérieur (*Coq, Isabelle, HOL*), propriétés plus complexes, (sécurité,...), moins d'automaticité.

Vérification de programmes

- On part d'un modèle formel d'un langage de programmation :
  - syntaxe
  - sémantique (machine abstraite ou sémantique axiomatique)
- on construit et valide un système de règles pour prouver la correction (totale) : *cf* Floyd, Hoare, Dijkstra.
- preuves de programmes, de schémas plus ou moins génériques.

L'outil Why (projet Proval)

- Langage impératif avec annotations (pre/post conditions, invariants/variants de boucles)
- Génération d'*obligations de preuves* en logique du premier ordre + arithmétique
- La correction de cette génération peut être prouvée par *Coq*. La preuve de ces obligations peut être déléguée à divers outils (automatiques ou non, selon leur difficulté)
- Applications :
  - *Krakatoa* : validation de programmes *Java*
  - *Caduceus* : langage *C* (y compris l'arithmétique de pointeurs) : exemple : Modélisation du système de gestion de la flash + vérification de l'implémentation à l'aide de Caducéus (June Andronick)

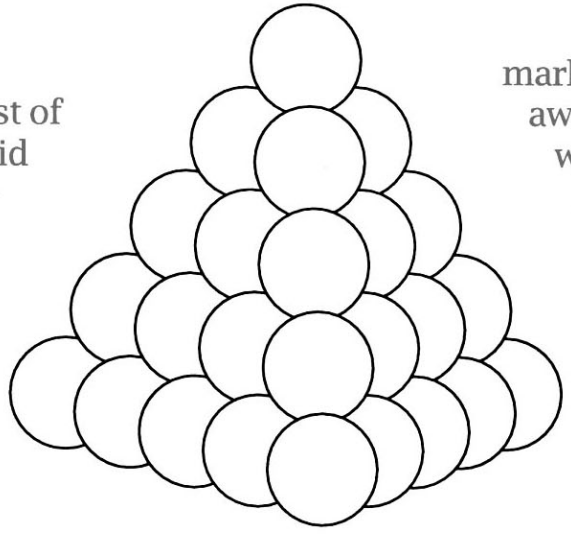
Autres exemples d'applications informatiques

- Preuve de confidentialité de la machine virtuelle *Java* (Olivier Ly.)
- Protocoles de communication : mise au point d'une classe d'automates temporisés, formalisation de leur comportement (traces infinies), vérification de propriétés (Davy Rouillard.)
- Théorie des langages : Formalisation de classes de grammaires catégorielles (multimodales) : (Houda Anoun.)
- Diverses logiques modales : épistémiques, temporelles, etc.
- Voir les sites des divers assistants.

Conjecture de Kepler

- Conjecture de Kepler (1611) : *Il n'existe pas de façon de ranger des sphères de même diamètre dont la densité soit supérieure à celle du rangement cubique à faces centrées (empilements d'oranges ou de boulets de canon) :* $\frac{\pi}{\sqrt{18}}$
- Contributions de Gauss, Thue (en 2D), Fejes Tóth.

When Hilbert introduced his famous list of 23 problems, he said a test of the perfection of a mathematical problem is whether it can be explained to the first person in the street. Even after a full century, Hilbert's problems have never been thoroughly tested. Who has ever chatted with a telemarketer about the Riemann hypothesis or discussed general reciprocity laws with the family physician?



market. "We need you down here right away. We can stack the oranges, but we're having trouble with the artichokes."

To me as a discrete geometer there is a serious question behind the flippancy. Why is the gulf so large between intuition and proof? Geometry taunts and defies us. For example, what about stacking tin cans? Can anyone doubt that parallel rows of upright cans give the best arrangement? Could some disordered heap of cans

- Preuve par Thomas Hales (1998), dont certaines parties utilisent des calculs sur machine :
  - Énumération de graphes (planaires) pouvant être des contre-exemples à la conjecture. Ces contre-exemples éventuels sont caractérisés par 8 contraintes portant sur les cycles, degrés des sommets, affectation de poids aux faces. Le programme *Java* de Hales énumère 5128 graphes satisfaisant ces contraintes.
  - Programmation linéaire, destinée à vérifier qu'aucun de ces graphes ne constitue un réel contre-exemple.

Le projet Flyspeck

- Vérification de la preuve de Hales par une équipe de 12 arbitres, et conférence consacrée à la preuve : résultat : certitude à 99%,
- Projet Flyspeck : construire une version formelle de la preuve de 1998, principalement à l'aide principalement de *HOL/Light* : mais aussi en *Isabelle/HOL* et *Coq*.

Contribution de Bauer et Nipkow (2005)

- Preuve de la complétude de l'énumération des graphes par le programme de Hales,
- Détection de redondances (2771 graphes au lieu des 5128 énumérés par le programme *Java* de Hales,)
- Détection d'une contrainte absente de la preuve de 98, mais traitée dans le programme *Java*,
- Bilan : la mécanisation de cette preuve a permis de nombreuses simplification dans le calcul de l'énumération des graphes,
- 17000 lignes d'*Isabelle*, 165 minutes de vérification, 2300000 graphes engendrés durant la preuve
- Reste à faire : vérifier que les 2771 graphes ne sont pas des contre-exemples réels à la conjecture de Kepler.

Quelle confiance accorder aux théorèmes prouvés sur machine ?

Les assistants de preuves sont des programmes assez gros ; par exemple le source de *Coq* mesure 12Mo et *Isabelle* 25Mo (non compris le langage d'implantation (ML). Peut-on s'y fier ?
Le *critère de de Bruijn* permet de remplacer l'impossible confiance absolue par une réduction de la partie de ces programmes dont on peut se méfier.
Cette partie critique concerne la vérification des preuves. L'aide à la construction de ces preuves n'a pas besoin d'être validée.

En *Isabelle*, *HOL*, un théorème est un objet du langage *ML*, d'un type appelé `thm`.
Les seules possibilités de construction de valeurs de ce type sont :

- les axiomes de la logique considérée,
- les axiomes obéissant au principe de définition
- les règles d'inférence

La méfiance éventuelle peut s'exercer uniquement sur ces trois constructions, et sur le système de type du langage *ML*.

En *Coq*, un système de types propre à ce formalisme est à la base de la vérification des théorèmes.
La preuve d'un théorème d'énoncé A est une expression t dont le type est A dans le *calcul des constructions inductives*.
La confiance en un théorème produit avec *Coq* se réduit à la confiance en ce formalisme et en l'implantation de l'algorithme de vérification de types.

Quelle automaticité ?

- Les domaines d'application usuels des assistants de preuve n'autorisent pas une automaticité totale
- Une notion de *tactique* s'y substitue : transformation d'une obligation de preuve (*but*) $\Gamma \vdash A$ en plusieurs obligations $\Gamma_1 \vdash A_1 \ldots \Gamma_n \vdash A_n$.
- Exemple : preuve par récurrence : on associe au but $\Gamma \vdash \forall n : \mathbb{N}, P(n)$ les deux buts $\Gamma \vdash P(0)$, et $\Gamma \vdash \forall n : \mathbb{N}, P(n) \Rightarrow P(n+1)$
- autres exemples : preuves par cas, inductions structurelle et transfinie, co-induction, réécritures,
- Automatismes limités : tautologies, arithmétique de Presburger, simplification dans des anneaux, etc.
- Programmation de tactiques composées, outils externes.

Coq

- Programmes, théorèmes, et programmes certifiés
- Faut-il avoir peur des axiomes ?

Preuves, programmes et programmes certifiés

Coq distingue deux grandes classes de types : les *propositions* et les *spécifications* :
Une proposition est un énoncé susceptible d'être prouvé : une *preuve* de A est un (λ-)terme de type A. Le type des propositions est appelé `Prop`.
Exemples de propositions : $1 < 3$,
$(\forall P : \texttt{Prop}, \neg\neg P \to P) \to (\forall P : \texttt{Prop}, P \vee \neg P)$

Note : Le symbole $\to$ est utilisé à la fois pour l'implication et les types fonctionnels.

Un exemple simple

```
Theorem double_neg_exm : (∀ P, ¬¬P → P) →
                         (∀ Q, Q ∨ ¬Q).
```

(en logique intuitionniste)

Un exemple simple

$$(\forall\ P, \neg\neg P \to P) \to (\forall\ Q, Q \vee \neg Q)$$

```
intros H Q.
```

$H : \forall\ P, \neg\neg P \to P$
$Q : Prop$

$$Q \vee \neg Q$$

Un exemple simple

$H : \forall\ P, \neg\neg P \to P$
$Q : Prop$
$=========================$
$Q \vee \neg Q$

```
apply H.
```

$H : \forall\ P, \neg\neg P \to P$
$Q : Prop$
$=========================$
$\neg\neg(Q \vee \neg Q)$

Un exemple simple

$H : \forall\ P, \neg\neg P \rightarrow P$
$Q : Prop$
$==========$
$\neg\neg(Q \vee \neg Q)$

```
firstorder.
```

Proof Completed

```
Qed.
```

Un exemple simple

Le terme suivant a été construit par l'application des tactiques. Seul le diagnostic de typage est important.

```
(λH:(∀ P, ¬¬P → P).
   (λQ:Prop .
       (H _ (λH0:¬(Q ∨ ¬Q).
            (H0  (or_intror Q
                 (λq:Q.(H0 (or_introl (¬Q) q))))))))) :

: (∀ P, ¬¬P → P) → (∀ Q, Q ∨ ¬Q).
```

Un exemple simple

H: *Supposons* $\forall\, P,\ \neg\neg P \longrightarrow P$
Soit Q *une proposition*
    H0: *Supposons* $\neg(Q \vee \neg Q)$
        q: *Supposons* Q
        Alors $Q \vee \neg Q$
        Contradiction avec H0
    On en conclut $\neg Q$
    Puis $Q \vee \neg Q$
    Contradiction avec H0
Donc $\neg\neg(Q \vee \neg Q)$
Par H, $Q \vee \neg Q$

Programmes et types

```
Fixpoint rev (A:Type)(l:list A) : list A :=
match l with
| nil ⇒ nil
| a::l' => rev l' :: (a::nil)
end
```

```
rev : ∀A:Type, list A → list A
```

```
rev (1::2::3::nil) ⇝ 3::2::1::nil
```

Preuves de programmes

Utilisation d'induction, de simplification, de réécritures, etc.

```
Lemma rev_of_append : ∀ l l', rev(l++l')=rev l' ++ rev l.
Proof.
  ...
Qed.

Lemma rev_involutive : ∀ l : list A, rev (rev l) =l.
Proof.
...
Qed.
```

Programmes certifiés

Programme retournant un résultat accompagné d'une preuve de validité de ce résultat par rapport à une spécification :
Par exemple, un programme de tri certifié prend en paramètre :

- un type A
- un ordre total décidable sur A,
- une liste l d'éléments de A

et retourne :

- une liste l' d'éléments de A,
- une preuve que l' est ordonnée et est une permutation de l.

Intérêt des programmes certifiés

- Expression simple de problèmes de décision : Construire un terme du type suivant :

  $\forall$ `x, {`$P\ x$`} + {`$\neg P\ x$`}`

  équivaut à associer à tout x un booléen et une preuve :
  - `vrai`, et une preuve de $P(x)$, ou
  - `faux`, et une preuve de $\neg(P(x))$.
- *Extraction* vers des langages de programmation (*ML*, *Haskell*) (en effaçant les preuves).
- Il est plus facile de développer un programme et sa preuve simultanément à partir d'une spécification que de prouver un programme déjà écrit.

Faut-il avoir peur des axiomes ?

Définition technique : un axiome affirmant une proposition A est la déclaration d'une constante de type A.
Par exemple, le tiers-exclu n'est pas prouvable en *Coq*, mais on peut travailler en logique classique en déclarant l'axiome suivant :

```
Axiom classic : ∀P:Prop, P ∨ ¬P.
```

Cette déclaration n'introduit pas d'incohérence.

Faut-il avoir peur des axiomes ?

L'axiome d' *extensionnalité* affirme que deux fonctions égales point par point sont identiques :

Si, pour tout x, $f(x) = g(x)$, *alors* $f = g$

Cet axiome ne pose pas de problème de cohérence. Notons qu'il permet d'inférer l'égalité : `quicksort = bubblesort`. *On rappelle que le lambda-calcul n'est pas un formalisme approprié pour la complexité.*

Opérateur de description (*epsilon de Hilbert*)

« Un entier p tel que $p^2 = n$, s'il existe, sinon n'importe quel entier »

L'adjonction à Coq de ce type d'expression permet de transcrire de raisonnements mathématiques (ensembles + axiome du choix, fonctions partielles), en en respectant la structure, et trouve des applications en linguistique (traitement des noms communs).

L'adjonction de l'opérateur de description à *Coq* a quelques conséquences lourdes :

- On peut construire des termes *non effectifs* « réalisant » une spécification.
  Exemple :

  t `: ∀ n, {halts (machine n)} + {¬ halts (machine n)}`

- Incohérence avec le système de typage de *Coq* d'avant 2004 (Hurkens, Geuvers)

Conséquences : utilisation d'`epsilon` limitée aux modules dédiés aux propriétés mathématiques [des composants informatiques.]

Suites de Goodstein, hydres, et ordinaux

- Suites de Goodstein et batailles d'hydres
- Ordinaux dénombrables

Suites de Goodstein, hydres, et ordinaux

Les suites de Goodstein et problèmes similaires (batailles d'Hydre) présentent les caractéristiques suivantes :

- Propriétés faciles à énoncer, parfois assez contre-intuitives :
- Les preuves de ces propriétés mêlent inférences et calculs

Suites de Goodstein

$$
\begin{aligned}
8 &= 2^{2+1} \\
80 &= 3^{3+1} - 1 \\
&= 3^3 \times 2 + 3^2 \times 2 + 3 \times 2 + 2 \\
169 &= 4^4 \times 2 + 4^2 \times 2 + 4 \times 2 + 1 \\
6310 &= 5^5 \times 2 + 5^2 \times 2 + 5 \times 2 + 0 \\
93395 &= 6^6 \times 2 + 6^2 \times 2 + 6 + 5 \\
&\dots \\
570623341475 &= 11^{11} \times 2 + 11^2 \times 2 + 11 \\
&\dots \\
4,176 \times 10^{31} &\sim 23^{23} \times 2 + 23^2 \times 2
\end{aligned}
$$

Brève Histoire de $G_{8,2}$ (suite)

$$b^b \times 2 + b^2 \qquad (b = 3 \times 2^{27} - 1)$$

$$\cdots$$

$$b^b \times 2 \qquad (b = N = 3 \times 2^{402653211} - 1)$$

$$b^b + \sum_{i=N}^{0} b^i \times N \qquad (b = N + 1)$$

$$\cdots$$

$$b^b \qquad (b = M)$$

$$\sum_{i=M}^{0} b^i \times M \qquad (b = M + 1)$$

Brève Histoire de $G_{8,2}$ (suite et fin)

$$
\begin{array}{rl}
b & (b = P) \\
P & (b > P) \\
\ldots & \\
4 & \\
3 & \\
2 & \\
1 & \\
0 &
\end{array}
$$

Résultats (Kirby et Paris)

1. La suite de Goodstein issue de 4 en base 2 atteint 0 après $3 \times 2^{402653211} - 1$ étapes,
2. Toute suite de Goodstein finit par atteindre 0,
3. Ce résultat ne peut pas être montré dans l'arithmétique de Peano (premier ordre + récurrence.)

Calcul de la longueur de G_4 en *Coq*

Choix d'une structure de donnée appropriée :

$$
\begin{aligned}
4 &= 2^2 \\
&\bullet \quad 3^2 \times 2 + 3 \times 2 + 2 \\
&\bullet \quad 4^2 \times 2 + 4 \times 2 + 1 \\
&\bullet \quad 5^2 \times 2 + 5 \times 2 + 0 \\
&\bullet \quad 6^2 \times 2 + 6 \times 1 + 5 \\
&\ldots
\end{aligned}
$$

Un item de la suite sera représenté par un quadruplet (b, a_2, a_1, a_0)

Calcul de quelques items de la suite (parmi les premiers)

```
nth_item 2 ⇝ (5 2 2 0)
nth_item 3 ⇝ (6 2 1 5)
nth_item 8 ⇝ (11 2 1 0)
nth_item 20 ⇝ (23 2 0 0)
nth_item 21 ⇝ (24 1 23 23)
nth_item 44 ⇝ (47 1 23 0)
nth_item 92 ⇝ (95 1 22 0)
nth_item 188 ⇝ (191 1 21 0)
```

Détection de régularités $(3 \times 2^n - 1, i, j, 0)$.

Preuve d'invariants (par récurrence)

$$
\begin{aligned}
(3 \times 2^n - 1, i, j+1, 0) &\xrightarrow{*} (3 \times 2^{n+1} - 1, i, j, 0) \\
(3 \times 2^n - 1, i, j, 0) &\xrightarrow{*} (3 \times 2^{n+j} - 1, i, 0, 0) \\
(3 \times 2^n - 1, i+1, 0, 0) &\xrightarrow{*} (3 \times 2^{n+3\times 2^n} - 1, i, 0, 0)
\end{aligned}
$$

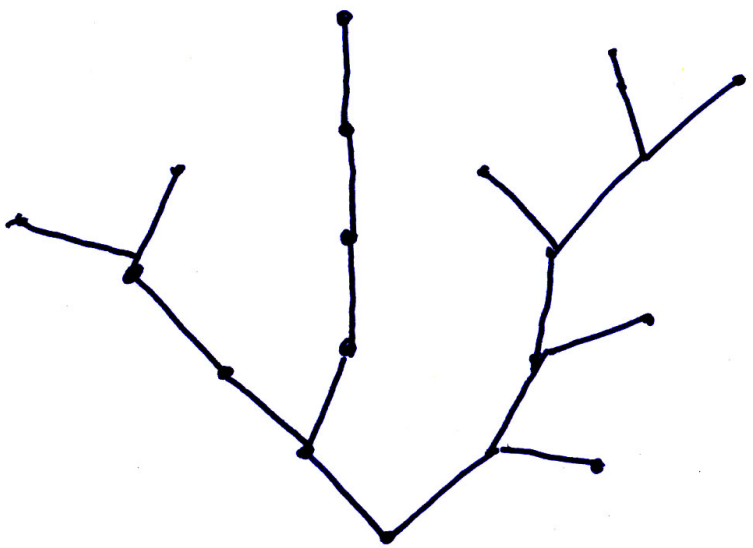
En utilisant ces lemmes et quelques calculs, on obtient :

$$
\begin{aligned}
(3 \times 2^1 - 1, 2, 2, 0) &\xrightarrow{*} (3 \times 2^3 - 1, 2, 0, 0) \\
&\xrightarrow{*} (3 \times 2^{27} - 1, 1, 0, 0) \\
&\xrightarrow{*} (3 \times 2^{27+3\times 2^{27}} - 1, 0, 0, 0)
\end{aligned}
$$

Preuves de terminaison

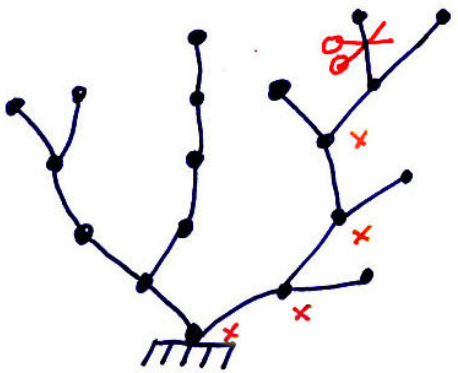
- La technique classique pour prouver la terminaison d'un processus est d'associer à tout état une *mesure*. Si cette mesure décroît à chaque étape dans un ordre bien fondé, la terminaison est assurée.
- Les *ordinaux* fournissent un "catalogue" d'ensembles bien fondés (totalement ordonnés).
- Nous illustrons cette technique sur un problème similaire aux suites de Goodstein : les *batailles d'Hydre*

Batailles d'Hydre

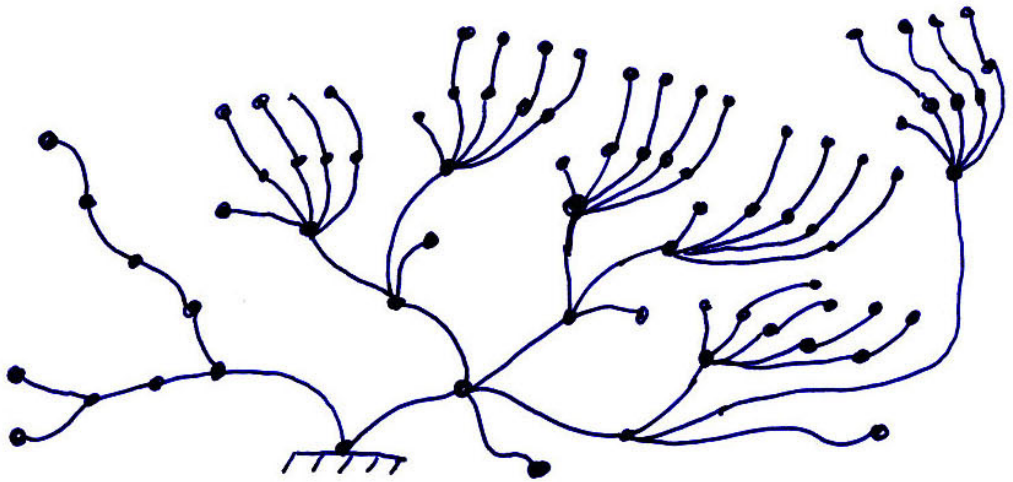


Définition : Une *hydre* est un animal mythique en forme d'arbre fini ; ses feuilles sont appelées *têtes*.

Batailles d'Hydre



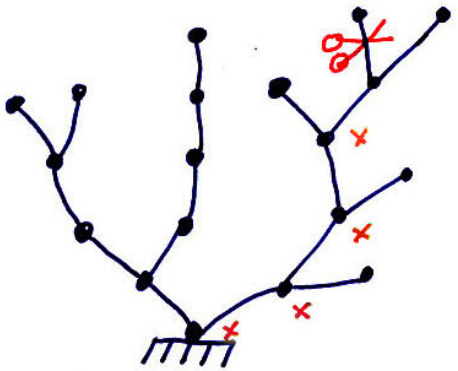
1



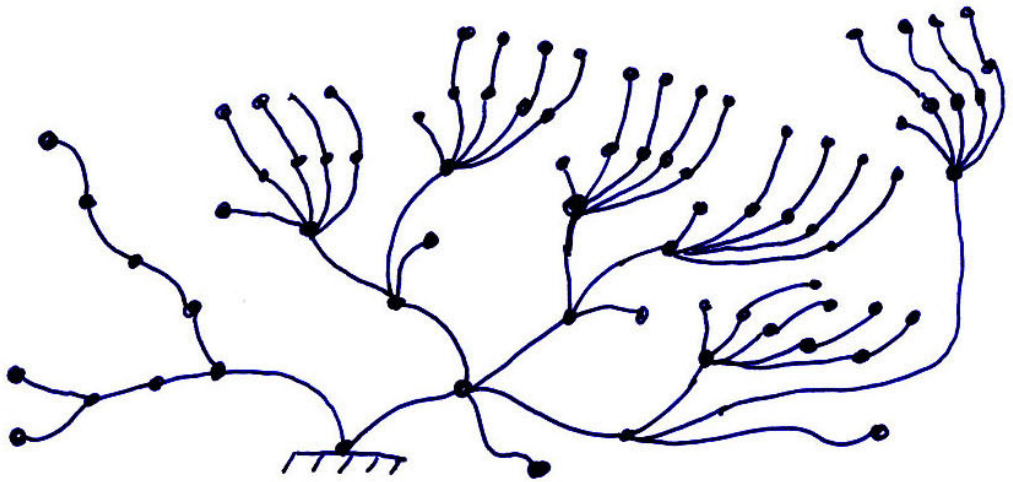
2

- À chaque tour, Hercule choisit une tête et la coupe
- L'hydre réagit en se répliquant autour des sommets marqués X un nombre fini (quelconque) de fois.

Batailles d'Hydre



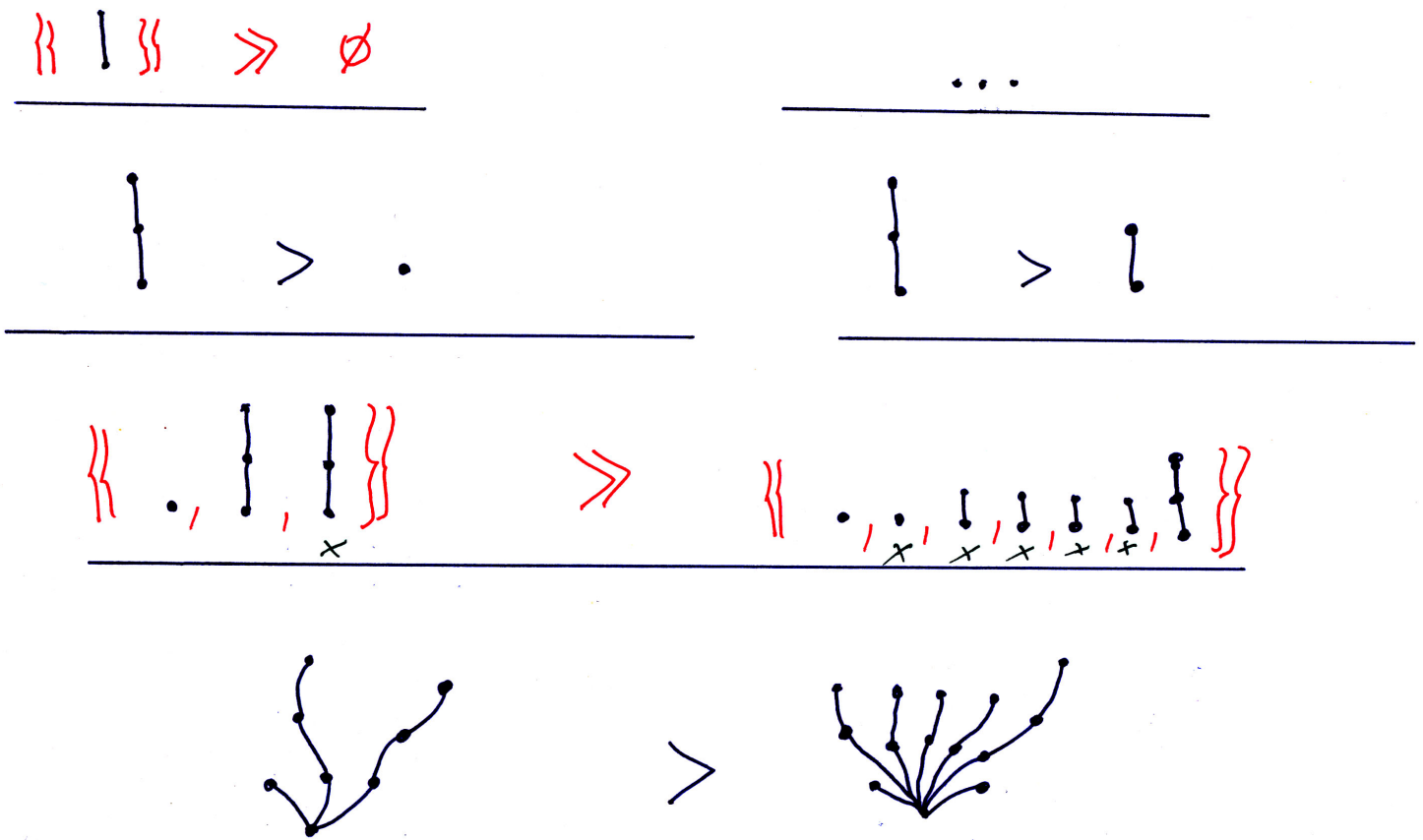
1



2

Théorème : Quelque soient les stratégies (récursives) d'Hercule (choix d'une tête) et de l'Hydre (nombre de réplications), Hercule finit par gagner (voir Kirby et Paris).

Hydres et multiensembles



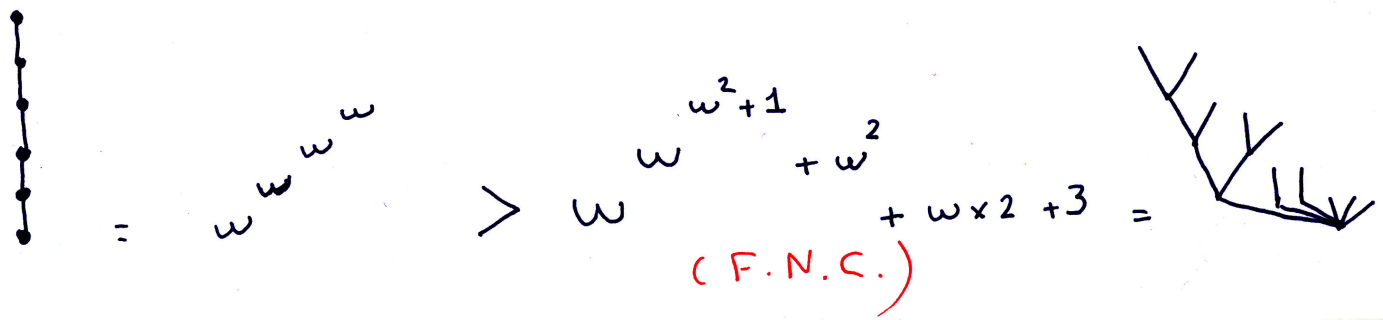
Hydres et ordinaux

$$(H, <) \simeq (MS(H), \ll)$$

$$\downarrow \qquad\qquad \downarrow$$

$$\alpha = \omega^{\alpha}$$

$$\alpha \geqslant \varepsilon_0 = \bigsqcup \{ \omega, \omega^{\omega}, \omega^{\omega^{\omega}}, \omega^{\omega^{\omega^{\omega}}} \ldots \}$$



Une implantation effective des ordinaux

- Bibliothèques sur les ordinaux (permettant de représenter ϵ_0, Γ_0), contenant :
  - Une structure de donnée appropriée,
  - Un ordre linéaire $<$
  - Des capacités de calcul : arithmétique, procédures de décision
  - Une preuve de bonne fondation de $<$.
  - « Validation » par une théorie axiomatique des ordinaux dénombrables (Schütte).

On peut alors prouver que la suite des ordinaux associés aux termes d'une suite de Goodstein est strictement décroissante. De même pour les batailles d'hydre.

Preuve de bonne fondation (difficulté)

$$\omega^{\omega^{\omega^{\omega}}} \times 2 > \omega^{\omega^{\omega^{\omega}}} + \omega^{\omega^{\omega}} \times 42 + \omega^{\omega^3} + \omega^{\omega} + 33$$

- L'induction structurelle ne peut suffire
- L'induction sur la hauteur (en ω) non plus

Une simple analogie ?

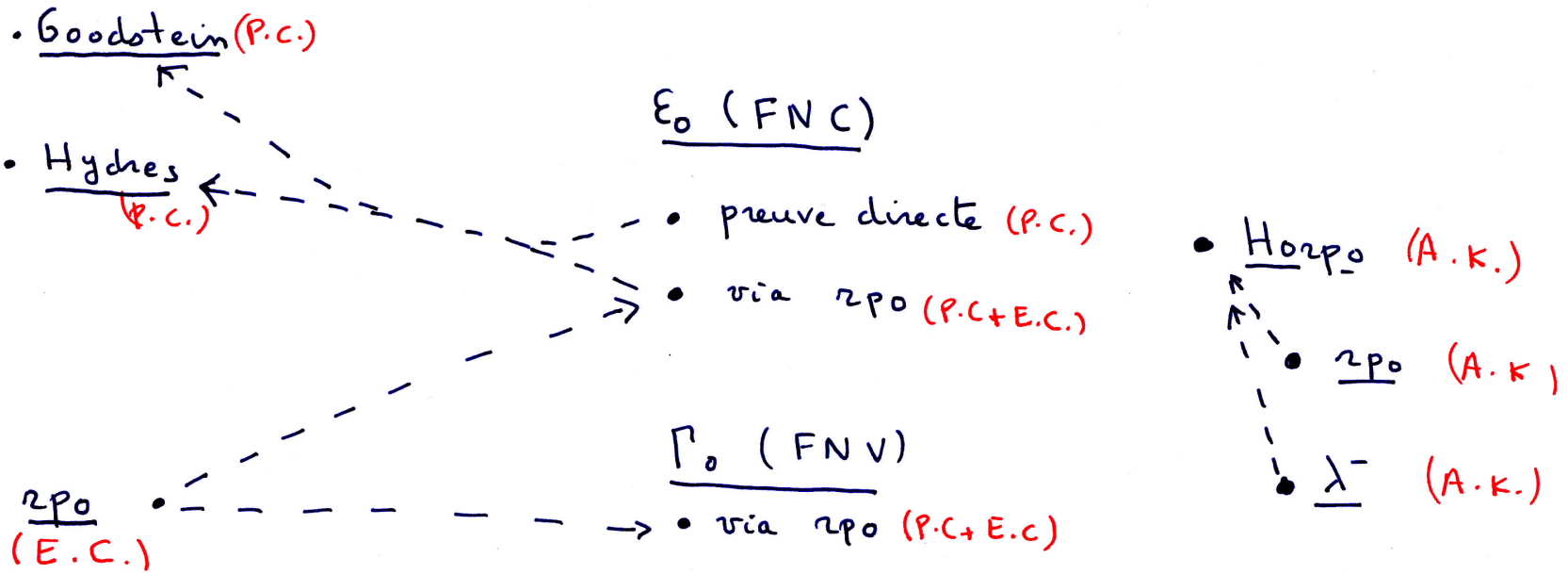
$$\omega^{\alpha} + \beta \quad > \quad \omega^{\alpha'} + \omega^{\beta_1} + \omega^{\beta_2} + \ldots \omega^{\beta_n}$$
$$(\alpha > \alpha' \geq \beta_1 \geq \ldots \beta_n)$$

$$(\lambda\, x.\, t)\; u \;\leadsto_{\beta}\; t\,[\ldots u \ldots u \ldots u \ldots]$$

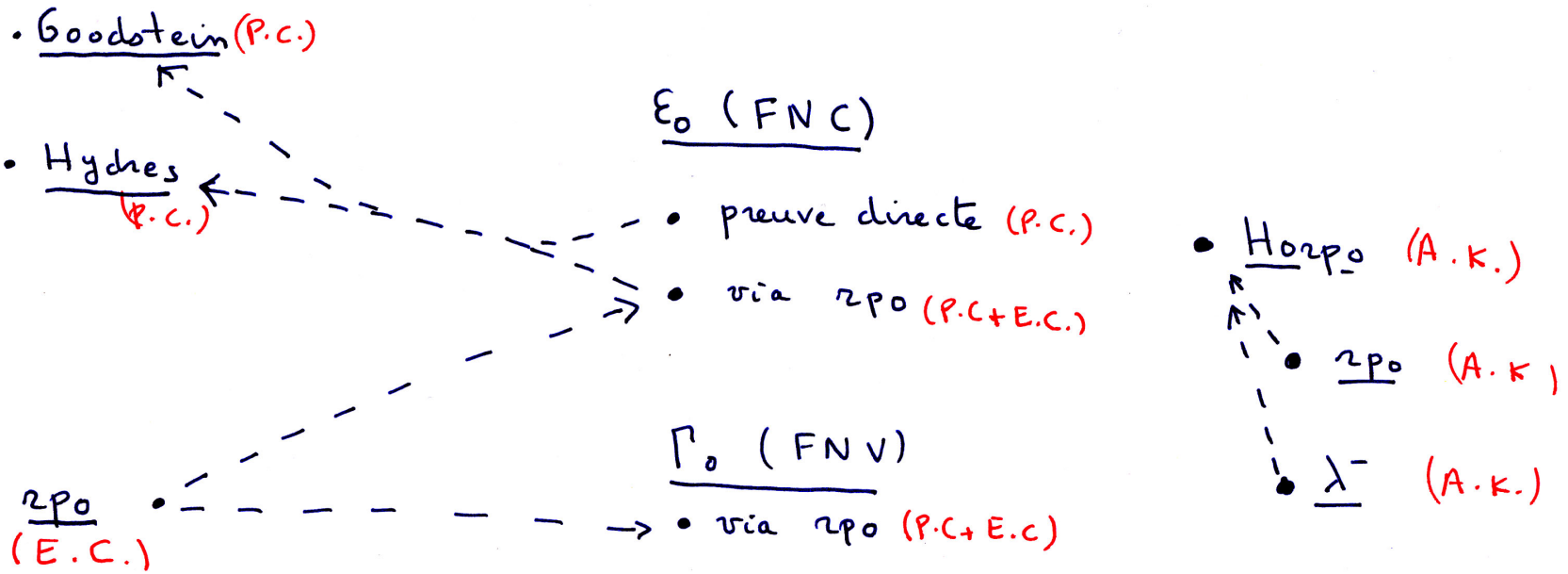
Adaptation de la preuve de Tait :

- triple induction enchevêtrée : hauteur des termes, largeur, induction bien fondée (limitée)
- $+$ induction structurelle

Tait partout ?



Tait partout ?



À faire : une preuve « générique » de Tait, dont les preuves ci-dessus deviendraient des instanciations.

Perspectives

À faire

- Évolution de *Coq* : interface avec le langage mathématique, et les autres formalismes (*HOL, Isabelle, etc.*)
- Travail sur les ordinaux et ordres bien fondés, preuves par réflexion
- Troisième résultat de Kirby et Paris,
- Isomorphismes de structures de données,
- Collaborations : Altarica, Phoenix, etc. ?