Systèmes de Preuves

Ecole Polytechnique - INF 565
Cours n. 1 – 8 janvier 2010

Introduction
+
Les objets de Coq

Benjamin Werner

Chercheur (INRIA) au LIX

Bureau 15, aîle 0, Rdc.

Benjamin.Werner@polytechnique.fr

C'est quoi les maths ?

Tous les hommes sont mortels, Socrate est un homme, donc Socrate est mortel.

C'est quoi les maths ?

Tous les hommes sont mortels, Socrate est un homme, donc Socrate est mortel.

C'est quoi les maths ?

Tous les hommes sont mortels, Socrate est un homme, donc Socrate est mortel.

correction : critère *syntaxique*

C'est quoi les maths ?

Tous les hommes sont mortels, Socrate est un homme, donc Socrate est mortel.

correction : critère *syntaxique*

$$\frac{\vdash A \Rightarrow B \qquad \vdash A}{\vdash B}$$

Les briques du raisonnement mathématique

C'est quoi les maths ?

Tous les hommes sont mortels, Socrate est un homme, donc Socrate est mortel.

correction : critère *syntaxique*

$$\frac{\vdash A \Rightarrow B \qquad \vdash A}{\vdash B}$$

Les briques du raisonnement mathématique

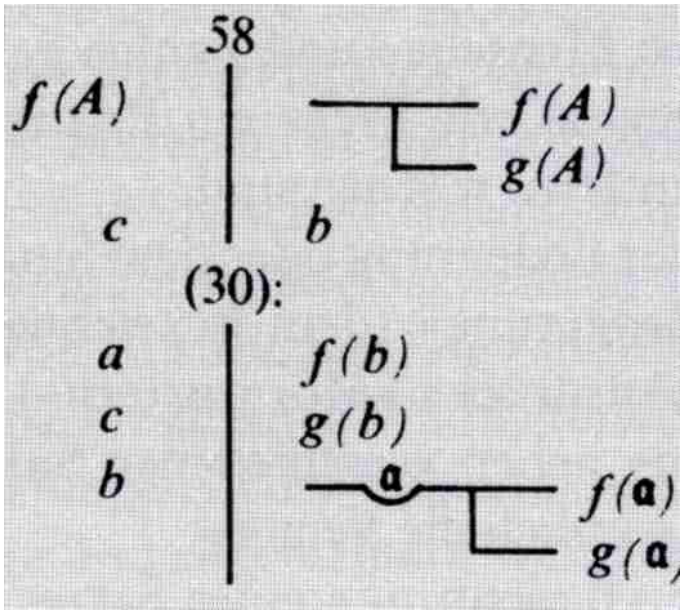
$$\frac{\dfrac{\vdash \forall x.H(x) \Rightarrow M(x)}{\vdash H(s) \Rightarrow M(S)} \qquad \vdash H(S)}{\vdash M(S)}$$

Une démonstration mathématique est une *construction*

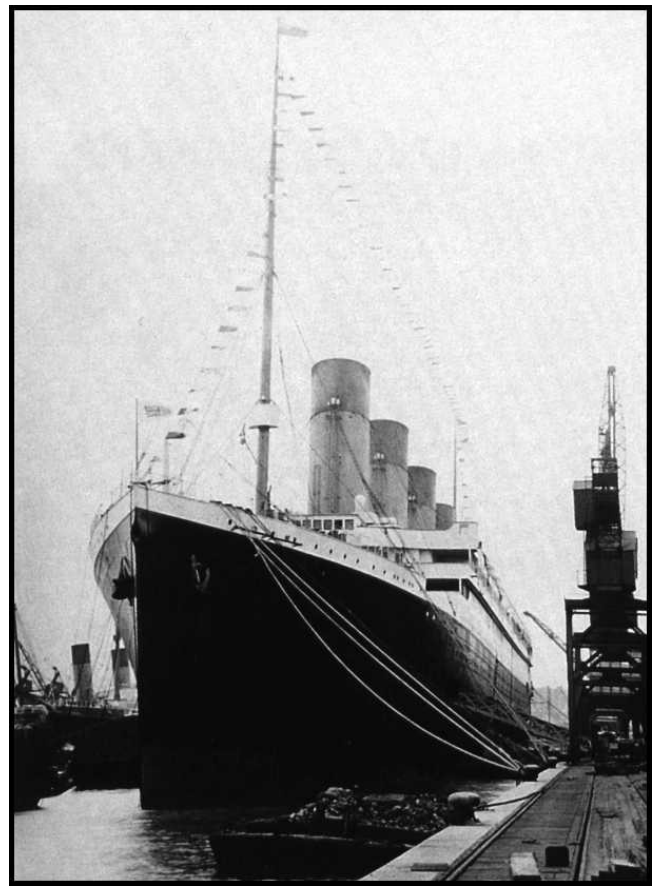
Naissance de la logique mathématique moderne

L'idée de la vérité mathématique définie de manière totalement *objective* grâce à des règles *syntaxiques*.

1872 : la *Begriffsschrift* de Frege



preuve= structure de donnée arborescente



vérification mécanique

Un siècle plus tard

La vérification mécanique du raisonnement devient réalité.

Premier système de preuves : Automath (1968)

N. G. de Bruijn

Les preuves formelles sont construites *en fait*

Un siècle plus tard

La vérification mécanique du raisonnement devient réalité.

Premier système de preuves : Automath (1968)

N. G. de Bruijn

Les preuves formelles sont construites *en fait*

Aujourd'hui

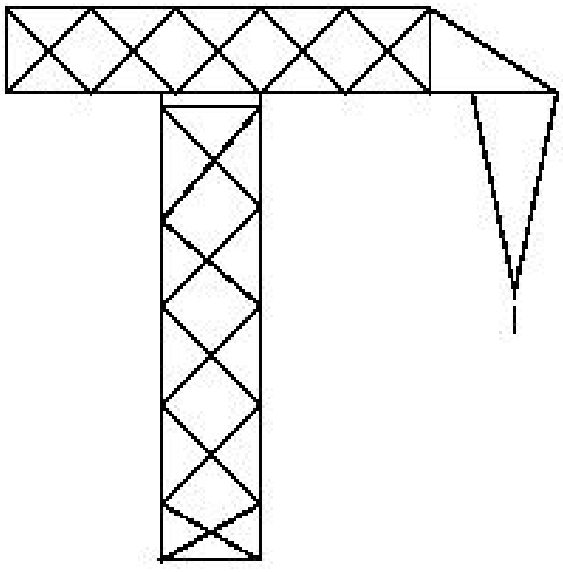
Un système de preuves moderne : Coq

- Même principe
- Formalisme un peu plus moderne

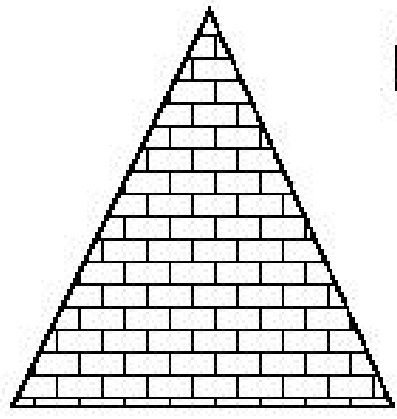
Mini-démo

Good cop, bad cop

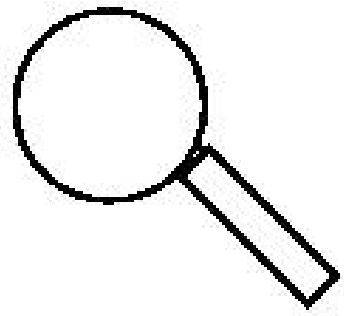
Architecture du système de preuves :



Aide à la preuve



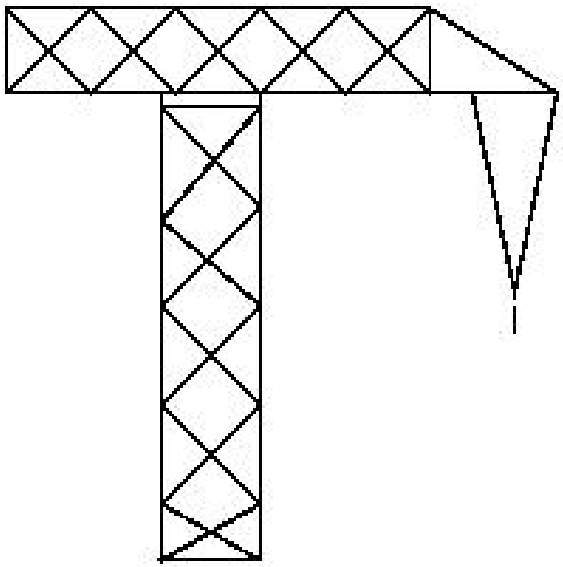
Preuve



Vérificateur
(noyau)

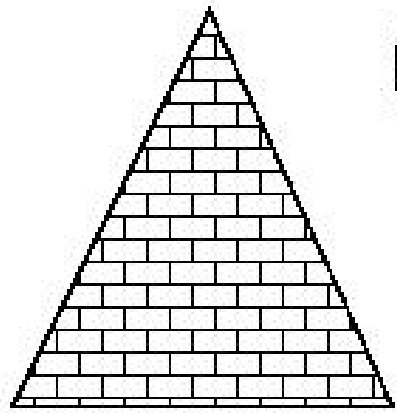
Good cop, bad cop

Architecture du système de preuves :

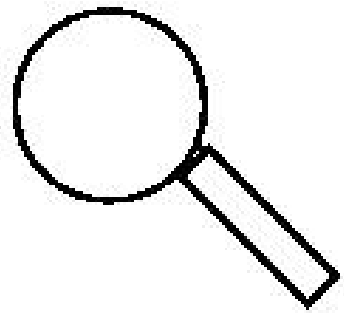


Aide à la preuve

compliqué si nécessaire



Preuve

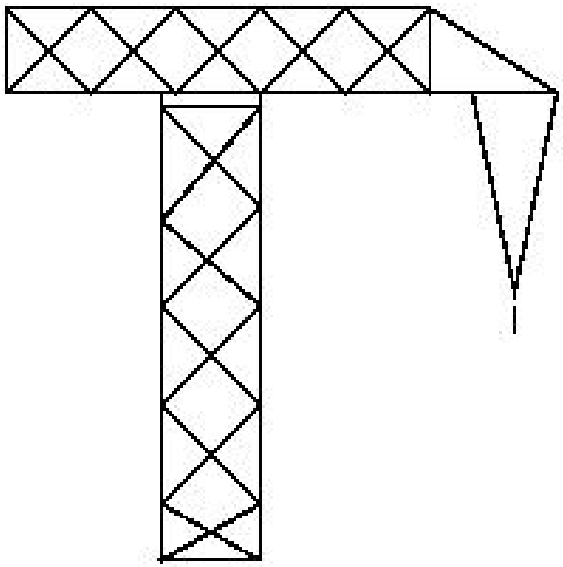


Vérificateur
(noyau)

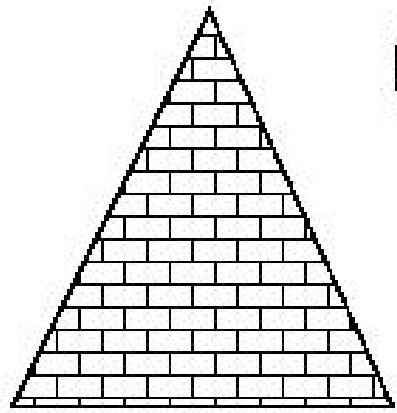
aussi simple que possible

Good cop, bad cop

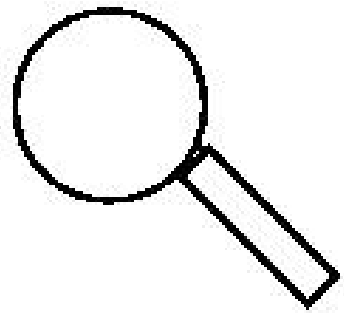
Architecture du système de preuves :



Aide à la preuve



Preuve



Vérificateur
(noyau)

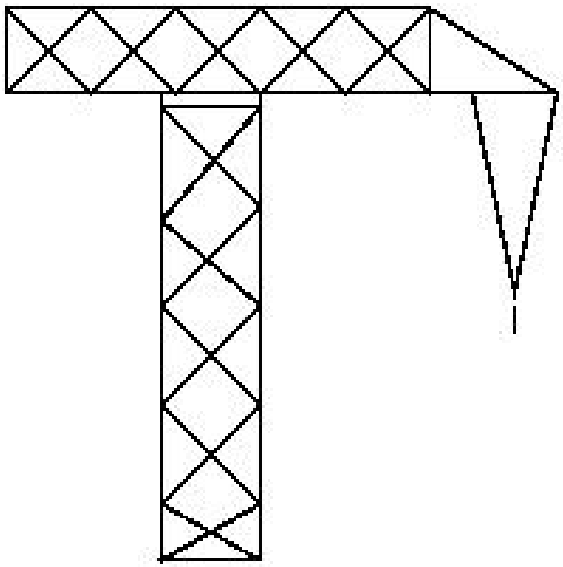
compliqué si nécessaire

aussi simple que possible

La vérité mathématique est validée expérimentalement !

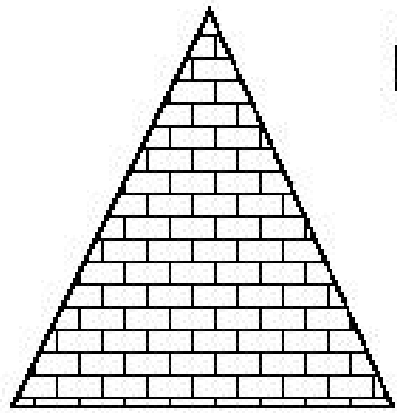
Good cop, bad cop

Architecture du système de preuves :

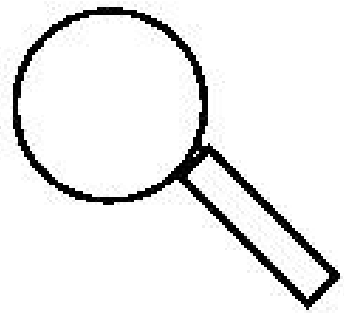


Aide à la preuve

compliqué si nécessaire



Preuve



Vérificateur
(noyau)

aussi simple que possible

La vérité mathématique est validée expérimentalement !

Mapple, Mathlab, Mathematica ne sont pas des systèmes de preuves

Peut-on faire confiance à la machine ?

Des règles logiques bien définies

- Cohérence : question de logique
- Théorème de Gödel : toujours une part d'incertitude de principe

Peut-on faire confiance à la machine ?

Des règles logiques bien définies

- Cohérence : question de logique
- Théorème de Gödel : toujours une part d'incertitude de principe

Le programme doit implémenter les règles

- On peut prouver formellement la correction du noyau
- D'abord une question d'architecture logicielle

On peut rejouer la vérification sur une autre machine, un autre OS, avec un autre noyau : varier les conditions expérimentales
Les Mathématiques deviennent une science comme les autres !

A quoi ça sert ?

Ca sert à être sûr.

A quoi ça sert ?

Ca sert à être sûr.

- En maths

A quoi ça sert ?

Ca sert à être sûr.

- En maths
- En informatique : preuves de correction de logiciels critiques

A quoi ça sert ?

Ca sert à être sûr.

- En maths
- En informatique : preuves de correction de logiciels critiques
  - Logiciels embarqués (ABS, ligne 14...)
  - Robots médicaux
  - Argent en jeu : protocoles de cartes bancaires...

A quoi ça sert ?

Ca sert à être sûr.

- En maths
- En informatique : preuves de correction de logiciels critiques
  - Logiciels embarqués (ABS, ligne 14...)
  - Robots médicaux
  - Argent en jeu : protocoles de cartes bancaires...

On sait prouver des propriétés de programmes !

More good cop

Nouvelles utilisations de l'ordinateur dans la recherche scientifique :

- Physique : simulations, calcul numériques, Monte-Carlo...
- Biologie : génomique, simulations du cycle cellulaire...
- Économie : modèles d'interactions entre agents vérifiés par simulation
- Chimie
- Mathématiques (!)
- Informatique

On peut établir des vérités jusque-là inaccessibles

Le télescope du mathématicien

plus grand nombre premier connu en 1951 :

$$(2^{148} + 1)/17 \quad \text{(44 chiffres)}$$

aujourd'hui : $2^{25964951} - 1$ (7.816.230 chiffres)
cause du progrès : évidente
Mais aussi de nouvelles mathématiques intéressantes

Le télescope du mathématicien

plus grand nombre premier connu en 1951 :

$$(2^{148}+1)/17 \quad \text{(44 chiffres)}$$

aujourd'hui : $2^{25964951}-1$ (7.816.230 chiffres)
cause du progrès : évidente
Mais aussi de nouvelles mathématiques intéressantes

Quel est le statut des ces "preuves" du point de vue formel ?

Le télescope du mathématicien

plus grand nombre premier connu en 1951 :

$$(2^{148}+1)/17 \quad (44 \text{ chiffres})$$

aujourd'hui : $2^{25964951}-1$ (7.816.230 chiffres)
cause du progrès : évidente
Mais aussi de nouvelles mathématiques intéressantes

Quel est le statut des ces "preuves" du point de vue formel ?

Dans quel *langage* sont-elles écrites ?

Formalismes calculatoires

Les objets du formalismes sont des programmes : l'addition est définie par un algorithme.
Qui dit programme dit calcul ; les objets sont identifiés modulo évaluation.

$$2 + 2 \rhd 4$$

par congruence :

$$2 + 2 = 4 \text{ est identifiée à } 4 = 4$$

$\Rightarrow$ preuve par réflexivité.
On n'a pas plus de théorèmes, mais les preuves sont plus courtes.

l'intégration du calcul donne plus de preuves *en pratique*

28554254222827961390156356610216400832616423864470288919924745660228440039060065387595457150553984323975451391589615029787839937056071435169747221107988791198200988477531339214282772016059009904586686254989084815735422480409022344297588352526004383890632616124076317387416881148592486188361873904175783145696016919574390765598280188599035784485910776836771755204340742877265780062667596159707595213278285556627816783856915818444364448125115624281367424904593632128101802760960881140100337757036354572512092407364692157679714619938761929656030268026179011813292501232304644443862230887792460937377301248168167242449367447448853777015578300688085264816151306714481479028836666406225727466527578712737464923109637500117090189078626332461957879573142569380507305611967758033808433338198750090296883193591309526982131114132239335649017848872898228815628260081383129614366384594543114404375382154287127774560644785856415921332844358020642271469491309176271644704168967807009677359042980890961675045292725800084350034483162829708990272864998199438764723457427626372969484830475091717418618113068851879274862261229334136892805663438446664632657247616727566083910565052897571389320211121495795311427946254553305387067821067601768750977866100460014602138408448021225053689054793742003095722096732954750721718115531871310231057902608580607est premier

2855425422282796139015635661021640083261642386447028891992474566022844
0039060065387595457150553984 0615029787839937705607143516
974722110798879119820098847 016059009904586686254989084
8157354224804090223442975883 326161240763173874168811485
9248618836187390417578314569 559828018859903557844859107
7683677175520434074287726578 75952132782855562781678385
6915818444364448125115624281 12810180276096088114010033
7757036354572512092407364692 761929656030268026179011813
2925012323046444438622308877 48168167242449367447448537

7701557830068808526481 0622572746652757871273
74649231096375001170901 2569380507305611967758
0338084333381987500902 1322393356490178488728

Proved in Coq

9822881562826008138312961436638459454311440437538215428712777456064478
5856415921332844358020642271469491309176271644704168967807009677359042
9808909616750452927258000843500344831628297089902728649981994387647234
5742762637296948483047509171741861811306885187927486226122933413689280
5663438446664632657247616727566083910565052897571389932021112149579531
1427946254553305387067821067601768750977866100460014602138408448021225
053689054793742003095722096732954750721718115531871310231057902608580607
est premier

Le calcul est partout

It is not just about the numbers

Certains théorèmes ne semblent pas être de nature calculatoire. Pourtant leurs seules preuves connues demandent des calculs importants.

- Le théorème des quatre couleurs (1976)
- La conjecture de Kepler (Thomas Hales, 1998)

Intéressant car les arguments utilisées mélangent de la déduction mathématique au sens usuel et des gros calculs mécaniques ; les deux parties sont sophistiquées.

Un problème de standards de vérification

Le calcul est partout

It is not just about the numbers

Certains théorèmes ne semblent pas être de nature calculatoire. Pourtant leurs seules preuves connues demandent des calculs importants.

- Le théorème des quatre couleurs (1976) prouvé in Coq
- La conjecture de Kepler (Thomas Hales, 1998)

Intéressant car les arguments utilisées mélangent de la déduction mathématique au sens usuel et des gros calculs mécaniques ; les deux parties sont sophistiquées.

Un problème de standards de vérification

The Future : Kepler's conjecture

The Future : Kepler's conjecture

Is there a better packing ?

The Future : Kepler's conjecture

Is there a better packing ?

Conjecture : Kepler, 1611 ..... Proof : Hales, 1998

The Future : Kepler's conjecture

Is there a better packing ?

Conjecture : Kepler, 1611 ..... Proof : Hales, 1998
"only to 99%"

The future : a small piece of Kepler's conjecture

Lemma 751442360

$$2.51^2 \leq x_1 \leq 2.696^2 \rightarrow$$
$$4 \leq x_2 \leq 2.168^2 \rightarrow$$
$$4 \leq x_3 \leq 2.168^2 \rightarrow$$
$$4 \leq x_4 \leq 2.51^2 \rightarrow$$
$$4 \leq x_5 \leq 2.51^2 \rightarrow$$
$$4 \leq x_6 \leq 2.51^2 \rightarrow$$

$$\frac{\begin{array}{c} -x_1x_3 - x_2x_4 + x_1x_5 + x_3x_6 - x_5x_6 + \\ x_2(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6) \end{array}}{\sqrt{4x_2 \left(\begin{array}{c} x_2x_4(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6) + \\ x_1x_5(x_2 - x_1 + x_3 + x_4 - x_5 + x_6) + \\ x_3x_6(x_2 + x_1 - x_3 + x_4 + x_5 - x_6) \\ - x_1x_3x_4 - x_2x_3x_5 - x_2x_1x_6 - x_4x_5x_6 \end{array} \right)}} < \tan\left(\frac{\pi}{2} - 0.74\right)$$

Les Objects de Coq

Quels objets ?

On parle des objets du discours mathématique.

On veut donc :

- Un "vrai" langage de programmation (ou presque)
- Qui puisse servir de base à la construction des Mathématiques
- Signification claire

Bon candidat : programmation fonctionnelle typée.

Essentiellement le noyau de ML

Rappels de programmation fonctionnelle

Les programmes sont des fonctions :

- argument = entrée, résultat = sortie
- les fonctions sont (aussi) des objets

```
fun x : nat => x + 1
     : nat -> nat
```

```
fun f : nat -> nat => f (f 3)
     : (nat -> nat) -> nat
```

Différence avec ML : on doit mettre plus d'information de typage.

Types de base

Définition inductive ; le plus petit type clos par des constructeurs

```
Coq < Inductive bool : Type :=
Coq <   | true : bool   |   false : bool.

Coq < Definition negation b :=
Coq < match b with
Coq < | false => true
Coq < | true => false
Coq < end.
negation is defined
```

Ca calcule

Pas besoin de parler de mémoire :

- un programme rend un résultat
- Un programme s'évalue en une valeur

Une instruction pour ça :

```
Coq < Eval compute in (negation true).
     = false
     : bool

Coq <
Coq < Eval compute in (negation (negation true)).
     = true
     : bool
```

On pourra aussi raisonner (teaser)

```
bool_ind
     : forall P : bool -> Prop,
       P true -> P false -> forall b : bool, P b
```

Il n'y a pas d'autres booléens que `true` et `false`.

Types récursifs

```
Coq < Inductive nat : Type := O : nat  |  S : nat->nat.
```

$0, S(0), S(S(0)) \ldots$

Types récursifs

```
Coq < Inductive nat : Type := O : nat  |  S : nat->nat.
```

$0, S(0), S(S(0)) \ldots$

```
nat_ind
     : forall P : nat -> Prop,
       P O -> (forall n : nat, P n -> P (S n)) -> forall n
```

Programmes récursifs :

```
Coq < Fixpoint plus (n m :nat) : nat :=
Coq < match n with
Coq < | O => m
Coq < | S p => S(plus p m)
Coq < end.
```

Restriction

On ne peut pas construire n'importe quelle fonction récursive :

```
Coq < Fixpoint foo (n :nat) : nat :=  foo n.
Error :
Recursive definition of foo is ill-formed.
In environment
foo : nat -> nat
n : nat
Recursive call to foo has principal argument equal to
"n"
instead of a subterm of n.
```

Condition de récurrence structurelle

Pour calculer `f (S p)` on peut utiliser :

- `(f p)`
- `(f p')` si `p'` est un sous-terme de `p`

Personne n'est parfait

C'est un peu arbitraire :

```
Coq < Fixpoint foon (n :nat) :=
Coq < match n with
Coq < | O => O
Coq < | S p => foon (foon p)
Coq < end.
Error :
Recursive definition of foon is ill-formed.
In environment
foon : nat -> nat
n : nat
p : nat
Recursive call to foon has principal argument equal to
"foon p"
instead of p.
```

Personne n'est parfait

C'est un peu arbitraire :

```
Coq < Fixpoint foon (n :nat) :=
Coq < match n with
Coq < | O => O
Coq < | S p => foon (foon p)
Coq < end.
Error :
Recursive definition of foon is ill-formed.
In environment
foon : nat -> nat
n : nat
p : nat
Recursive call to foon has principal argument equal to
"foon p"
instead of p.
```

Dommage, certes, mais en général la terminaison est indécidable.

Looping is dangerous

Pourquoi cette restriction ?

Looping is dangerous

Pourquoi cette restriction ?

On comprendra mieux la semaine prochaine, mais en gros :

```
Coq < Fixpoint foob (b :bool) : bool := negb (foob b).
```

On aurait `foob true = negb (foob true)`

- Soit `true = false`
- soit `(foob true)` n'est ni `true` ni `false`

Les deux sont logiquement inacceptables.