Calcul de plus faible précondition, revisité en Why3.

C. Marché A. Tafat

Lab. Recherche en Informatique, Univ Paris-Sud, CNRS, INRIA Saclay.

JFLA 2013

4 février 2013

Contexte

- Formaliser un approche **originale** d'un calcul de WP
  - Sémantique **bloquante**
  - Preuve par préservation et progrès
- Illustrer l'environnement Why3
  - Expressivité du langage de spécification
  - Efficacité des outils de preuve

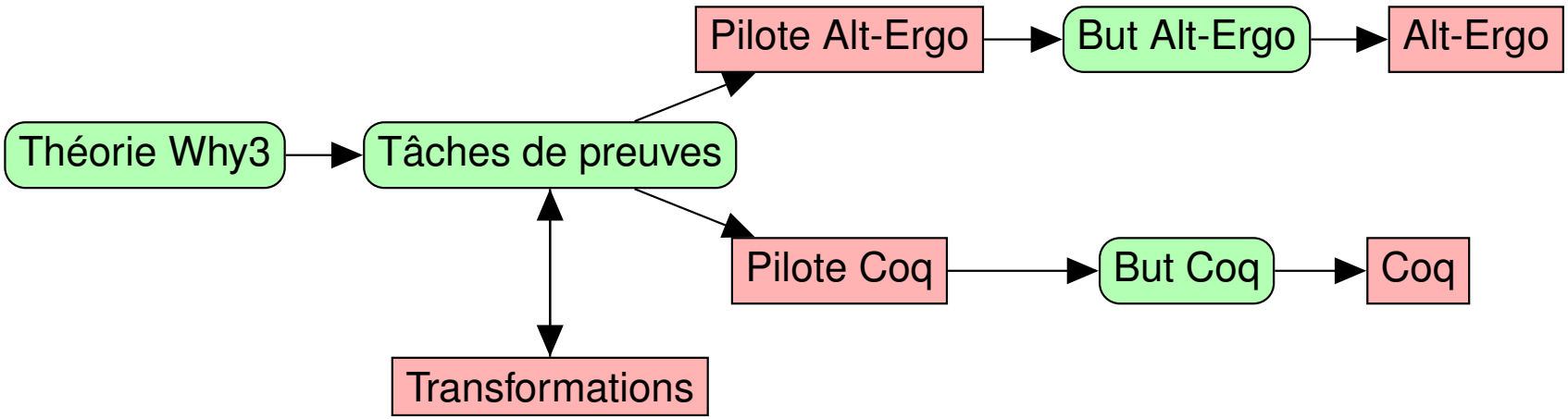
Calcul de plus faible précondition

Approche classique :

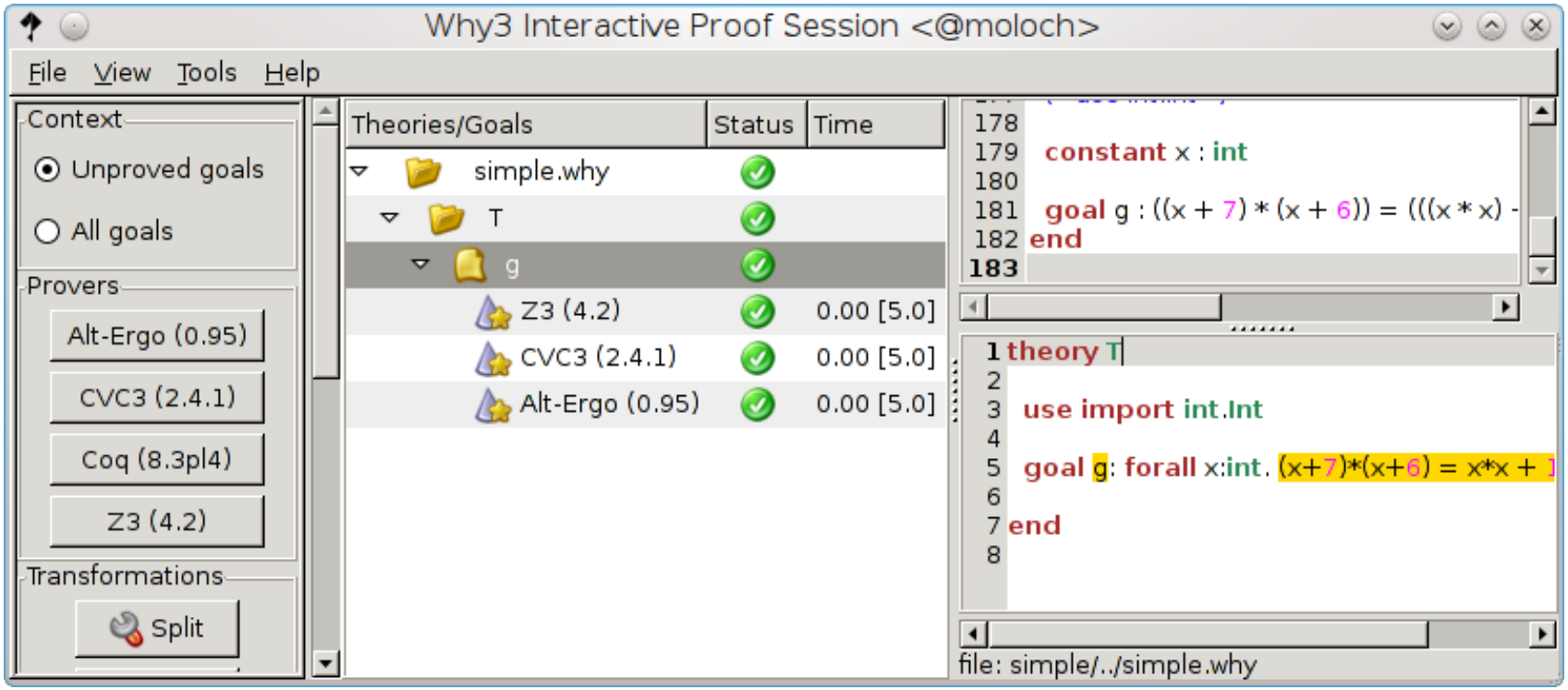
- Triplet de Hoare : $\{P\}\ S\ \{Q\}$
- Calcul de WP : $statement \rightarrow formula \rightarrow formula$
- Preuve de validité d'un triplet
  Si $P \Rightarrow WP(s, Q)$ alors $\{P\}\ S\ \{Q\}$ est valide

Why3 : Exemple

```
theory T

  use import int.Int

  goal g: forall x:int. (x+7)*(x+6) = x*x + 13*x + 42

end
```



Interface graphique Why3



Plan

Plan

Instructions

- Langage impératif minimal avec annotations
- Syntaxe abstraite formalisée grâce à un *type algébrique* de Why3

```
type stmt =
  | Sskip                    (* instruction no-op *)
  | Sassign mident term      (* affectation id := term *)
  | Sseq stmt stmt           (* sequence *)
  | Sif term stmt stmt       (* conditionnelle *)
  | Sassert fmla             (* assertion *)
  | Swhile term fmla stmt    (* while cond invariant body *)
```

Instructions

- Langage impératif minimal avec annotations
- Syntaxe abstraite formalisée grâce à un *type algébrique* de Why3

```
type stmt =
  | Sskip                    (* instruction no-op *)
  | Sassign mident term      (* affectation id := term *)
  | Sseq stmt stmt           (* sequence *)
  | Sif term stmt stmt       (* conditionnelle *)
  | Sassert fmla             (* assertion *)
  | Swhile term fmla stmt    (* while cond invariant body *)
```

Termes

- Why3 permet d'introduire des *types abstraits*

```
type mident (* identificateurs pour les variables mutables *)
type ident  (* identificateurs pour les variables logiques *)
```

```
type datatype = TYunit | TYint | TYbool
type value = Vvoid | Vint int | Vbool bool
type operator = Oplus | Ominus | Omult | Ole
type term =
 | Tvalue value                (* valeur *)
 | Tvar ident                  (* variable logique *)
 | Tderef mident               (* variable mutable *)
 | Tbin term operator term     (* operation binaire *)
```

Formules

```
type fmla =
  | Fterm term                    (* formule atomique *)
  | Fand fmla fmla                (* conjonction *)
  | Fnot fmla                     (* negation *)
  | Fimplies fmla fmla            (* implication *)
  | Flet ident term fmla          (* let id = term in fmla *)
  | Fforall ident datatype fmla   (* forall id: ty, fmla *)
```

Environnements

- Why3 possède une *librairie standard riche*
- Généricité grâce au *polymorphisme de type*

```
use map.Map as IdMap
type env = IdMap.map mident value
      (* associe des valeurs aux variables globales *)

use export list.List
type stack = list (ident, value)
      (* associe des valeurs aux variables logiques *)
```

Accès aux environnements

- Why3 permet de définir des fonctions *par récurrence structurelle*

```
function get_stack (i:ident) (pi:stack) : value =
  match pi with
  | Nil → Vvoid
  | Cons (x,v) r → if x=i then v else get_stack i r
  end
```

Accès aux environnements

- Why3 permet de définir des fonctions *par récurrence structurelle*

```
function get_stack (i:ident) (pi:stack) : value =
  match pi with
  | Nil → Vvoid
  | Cons (x,v) r → if x=i then v else get_stack i r
  end
```

En Why3, les fonctions logiques doivent être totales

Évaluation des termes

```
function eval_term (sigma:env)(pi:stack)(t:term): value =
  match t with
  | Tvalue v        → v
  | Tvar id         → get_stack id pi
  | Tderef id       → IdMap.get sigma id
  | Tbin t1 op t2   → eval_bin (eval_term sigma pi t1)
                               op (eval_term sigma pi t2)
  end
```

Évaluation des formules

```
predicate eval_fmla (sigma:env) (pi:stack) (f:fmla) =
  match f with
  | Fterm t          → eval_term sigma pi t = Vbool True
  | Fand f1 f2       → eval_fmla sigma pi f1 ∧
                                  eval_fmla sigma pi f2
  | Fnot f           → not (eval_fmla sigma pi f)
  | Flet x t f       → eval_fmla sigma
                         (Cons (x,eval_term sigma pi t) pi) f
  | Fforall x TYint f→ forall n:int.
                         eval_fmla sigma (Cons (x,Vint n) pi) f
  ...
  end
```

Sémantique opérationnelle

A petits pas, version classique

$$\frac{[\![cond]\!]_{\Sigma,\Pi} = True}{\Sigma, \Pi, \texttt{while}\ cond\ \texttt{inv}\ I\ \texttt{do}\ body \ \leadsto\ \Sigma, \Pi, (body;\ \texttt{while}\ cond\ \texttt{inv}\ I\ \texttt{do}\ body)}$$

$$\frac{}{\Sigma, \Pi, \texttt{assert}\ P \ \leadsto\ \Sigma, \Pi, Sskip}$$

Sémantique opérationnelle

A petits pas, version bloquante

$$\frac{[\![I]\!]_{\Sigma,\Pi} \qquad [\![cond]\!]_{\Sigma,\Pi} = True}{\Sigma, \Pi, \texttt{while}\ cond\ \texttt{inv}\ I\ \texttt{do}\ body \rightsquigarrow \Sigma, \Pi, (body;\ \texttt{while}\ cond\ \texttt{inv}\ I\ \texttt{do}\ body)}$$

$$\frac{[\![P]\!]_{\Sigma,\Pi}}{\Sigma, \Pi, \texttt{assert}\ P \rightsquigarrow \Sigma, \Pi, Sskip}$$

Sémantique bloquante
[Herms et al, VSTTE 2012, cas à grands pas]

Par définition, un programme annoté *respecte ses spécifications* si son exécution ne bloque pas

Sémantique opérationnelle

- Why3 permet la *définition de prédicat par induction*

```
inductive one_step env stack stmt env stack stmt =
  | one_step_assert: forall sigma:env, pi:stack, f:fmla.
      eval_fmla sigma pi f →        (* semantique bloquante *)
      one_step sigma pi (Sassert f) sigma pi Sskip

... (* 7 autres regles *)
```

Typage

- Nous définissons les environnements de typage

```
type type_stack = list (ident, datatype)
type type_env = IdMap.map mident datatype
```

Typage

- Nous définissons les environnements de typage

```
type type_stack = list (ident, datatype)
type type_env = IdMap.map mident datatype
```

- Règles de typage $\rightarrow$ prédicats inductifs en Why3

```
inductive type_term type_env type_stack term datatype =
...

inductive type_fmla type_env type_stack fmla =
...

inductive type_stmt type_env type_stack stmt =
...
```

Typage

- Nous définissons les environnements de typage

```
type type_stack = list (ident, datatype)
type type_env = IdMap.map mident datatype
```

- Règles de typage → prédicats inductifs en Why3

```
inductive type_term type_env type_stack term datatype =
...

inductive type_fmla type_env type_stack fmla =
...

inductive type_stmt type_env type_stack stmt =
...
```

```
predicate compatible_env (sigma:env) (sigmat:type_env)
                  (pi:stack) (pit: type_stack) = ...
```

Préservation du type par réduction

```
lemma type_preservation :
  forall s1 s2:stmt, sigma1 sigma2:env, pi1 pi2:stack,
                     sigmat:type_env, pit:type_stack.
   type_stmt sigmat pit s1 ∧
   compatible_env sigma1 sigmat pi1 pit ∧
   one_step sigma1 pi1 s1 sigma2 pi2 s2 →
   type_stmt sigmat pit s2 ∧
   compatible_env sigma2 sigmat pi2 pit
```

Préservation du type par réduction

Lemme

$$\forall\ s\ s'\ \Sigma\ \Sigma'\ \Pi\ \Pi'\ \Sigma_t\ \Pi_t.$$
$$\mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s \wedge \mathit{compatible\_env}\ \Sigma\ \Sigma_t\ \Pi\ \Pi_t \wedge$$
$$\Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s' \Rightarrow \mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s' \wedge$$
$$\mathit{compatible\_env}\ \Sigma'\ \Sigma_t\ \Pi'\ \Pi_t$$

Préservation du type par réduction

Lemme

$$\forall\ s\ s'\ \Sigma\ \Sigma'\ \Pi\ \Pi'\ \Sigma_t\ \Pi_t.$$
$$\mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s \wedge \mathit{compatible\_env}\ \Sigma\ \Sigma_t\ \Pi\ \Pi_t \wedge$$
$$\Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s' \Rightarrow \mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s' \wedge$$
$$\mathit{compatible\_env}\ \Sigma'\ \Sigma_t\ \Pi'\ \Pi_t$$

Preuve (Standard)

Par induction sur l'hypothèse $\Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s'$

Préservation du type par réduction

Lemme

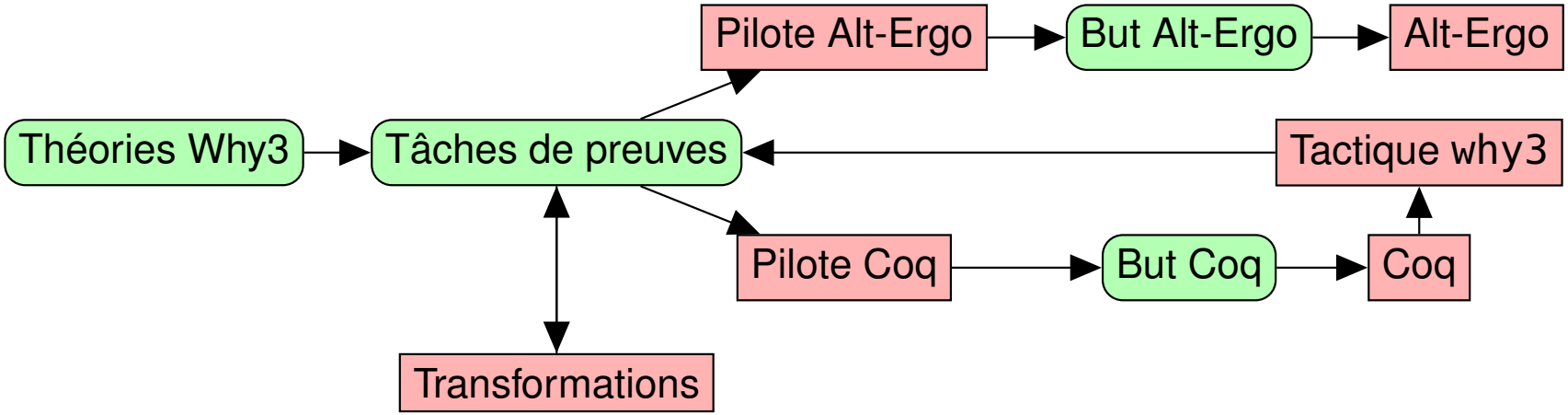
$$\forall\ s\ s'\ \Sigma\ \Sigma'\ \Pi\ \Pi'\ \Sigma_t\ \Pi_t.$$
$$\mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s \wedge \mathit{compatible\_env}\ \Sigma\ \Sigma_t\ \Pi\ \Pi_t \wedge$$
$$\Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s' \Rightarrow \mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s' \wedge$$
$$\mathit{compatible\_env}\ \Sigma'\ \Sigma_t\ \Pi'\ \Pi_t$$

Preuve (Standard)

Par induction sur l'hypothèse $\Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s'$

Impossible avec les prouveurs automatiques

Tactique `why3` de Coq



Preuve

```
Require Import Why3.
Ltac ae := why3 "alt-ergo" timelimit 5

intros s1 s2 sigma1 sigma2 pi1 pi2 sigmat pit (h1,(h2,h3)).
induction h3; try ae. (* 7 sous-buts prouves sur 8 *)
inversion h1; subst; ae.
```

Plan

Substitution

- Substituer une variable mutable par une variable logique

```
function msubst_term (t:term) (x:mident) (v:ident) : term =
  match t with
  | Tvalue _ | Tvar _ → t
  | Tderef y          → if x = y then Tvar v else t
  | Tbin t1 op t2     → Tbin (msubst_term t1 x v) op
                             (msubst_term t2 x v)
  end

function msubst (f:fmla) (x:mident) (v:ident) : fmla =
  match f with
  | Fterm e       → Fterm (msubst_term e x v)
  | Fand f1 f2    → Fand (msubst f1 x v) (msubst f2 x v)
  | Flet y t f    → Flet y (msubst_term t x v) (msubst f x v)
  ...
```

Fraîcheur

Affirmer qu'une variable est fraîche dans un terme ou une formule

```
predicate fresh_in_term (id:ident) (t:term) =
  match t with
  | Tvalue _ | Tderef _ → true
  | Tvar i        → id ≠ i
  | Tbin t1 _ t2 → fresh_in_term id t1 ∧ fresh_in_term id t2
  end

predicate fresh_in_fmla (id:ident) (f:fmla) =
  match f with
  | Fterm e     → fresh_in_term id e
  | Fand f1 f2  → fresh_in_fmla id f1 ∧ fresh_in_fmla id f2
  | Flet y t f  → id ≠ y ∧ fresh_in_term id t ∧
                                fresh_in_fmla id f
  ...
```

Propriétés sur `subst` et `fresh`

Lemme 1

$[\![t[x \leftarrow v]]\!]_{\Sigma,\,\Pi} = [\![t]\!]_{\Sigma[x \leftarrow \Pi(v)],\,\Pi}$

$[\![f[x \leftarrow v]]\!]_{\Sigma,\,\Pi} \Leftrightarrow [\![f]\!]_{\Sigma[x \leftarrow \Pi(v)],\,\Pi}$

Propriétés sur `subst` et `fresh`

Lemme 1

$$[\![t[x \leftarrow v]]\!]_{\Sigma,\ \Pi} = [\![t]\!]_{\Sigma[x \leftarrow \Pi(v)],\ \Pi}$$
$$[\![f[x \leftarrow v]]\!]_{\Sigma,\ \Pi} \Leftrightarrow [\![f]\!]_{\Sigma[x \leftarrow \Pi(v)],\ \Pi}$$

Preuve

Transformation "induction" de Why3

Propriétés sur `subst` et `fresh`

Lemme 1

$[\![t[x \leftarrow v]]\!]_{\Sigma,\ \Pi} = [\![t]\!]_{\Sigma[x \leftarrow \Pi(v)],\ \Pi}$

$[\![f[x \leftarrow v]]\!]_{\Sigma,\ \Pi} \Leftrightarrow [\![f]\!]_{\Sigma[x \leftarrow \Pi(v)],\ \Pi}$

Preuves

- Termes : Les 4 sous-buts prouvés automatiquement
- Formules :
  - 10 sous-buts prouvés automatiquement
  - 2 sous-buts (Fforall et Flet) prouvés en Coq

```
destruct f; auto.
simpl; ae.
```

Propriétés sur `subst` et `fresh`

Lemme 2

$$[\![t]\!]_{\Sigma,\ \Pi_1\cdot(id_1,v_1)\cdot(id_2,v_2)\cdot\Pi_2} = [\![t]\!]_{\Sigma,\ \Pi_1\cdot(id_2,v_2)\cdot(id_1,v_1)\cdot\Pi_2} \qquad id_1 \neq id_2$$
$$[\![f]\!]_{\Sigma,\ \Pi_1\cdot(id_1,v_1)\cdot(id_2,v_2)\cdot\Pi_2} \Leftrightarrow [\![f]\!]_{\Sigma,\ \Pi_1\cdot(id_2,v_2)\cdot(id_1,v_1)\cdot\Pi_2}$$

Lemme 3

$[\![t]\!]_{\Sigma,\ (id,v)\cdot\Pi} = [\![t]\!]_{\Sigma,\ \Pi}$ si id est frais.
$[\![f]\!]_{\Sigma,\ (id,v)\cdot\Pi} \Leftrightarrow [\![f]\!]_{\Sigma,\ \Pi}$ si id est frais.

Preuves avec la transformation "induction" de Why3

- 32 sous-buts
  - 30 prouvés automatiquement
  - 2 restants : 4 + 3 lignes de Coq

Plan

Définition du calcul de WP

```
function wp (s:stmt) (q:fmla) : fmla =
  match s with
  | Sskip                  → q
  | Sassert f              → Fand f (Fimplies f q)
  | Sseq s1 s2             → wp s1 (wp s2 q)
  | Sassign x t            → let id = fresh_from q in
                                Flet id t (msubst q x id)
  | Sif t s1 s2            →
       Fand (Fimplies (Fterm t) (wp s1 q))
            (Fimplies (Fnot (Fterm t)) (wp s2 q))
  | Swhile cond inv body →
    Fand inv (abstract_effects body
     (Fand (Fimplies (Fand (Fterm cond) inv)(wp body inv))
       (Fimplies (Fand (Fnot (Fterm cond)) inv) q)))
  end
```

Définition du calcul de WP

```
function wp (s:stmt) (q:fmla) : fmla =
  match s with
  | Sskip                → q
  | Sassert f            → Fand f (Fimplies f q)
  | Sseq s1 s2           → wp s1 (wp s2 q)
  | Sassign x t          → let id = fresh_from q in
                               Flet id t (msubst q x id)
  | Sif t s1 s2          →
       Fand (Fimplies (Fterm t) (wp s1 q))
            (Fimplies (Fnot (Fterm t)) (wp s2 q))
  | Swhile cond inv body →
    Fand inv (abstract_effects body
     (Fand (Fimplies (Fand (Fterm cond) inv)(wp body inv))
       (Fimplies (Fand (Fnot (Fterm cond)) inv) q)))
  end
```

Définition du calcul de WP

```
function wp (s:stmt) (q:fmla) : fmla =
  match s with
  | Sskip                  → q
  | Sassert f              → Fand f (Fimplies f q)
  | Sseq s1 s2             → wp s1 (wp s2 q)
  | Sassign x t            → let id = fresh_from q in
                                Flet id t (msubst q x id)
  | Sif t s1 s2            →
       Fand (Fimplies (Fterm t) (wp s1 q))
            (Fimplies (Fnot (Fterm t)) (wp s2 q))
  | Swhile cond inv body →
    Fand inv (abstract_effects body
     (Fand (Fimplies (Fand (Fterm cond) inv)(wp body inv))
       (Fimplies (Fand (Fnot (Fterm cond)) inv) q)))
  end
```

- On déclare des fonctions auxiliaires
  - Abstraites
  - Axiomatisées

```
function fresh_from (f: fmla) : ident
  (* renvoie une variable fraiche pour f *)
axiom fresh_from_fmla:
       forall f: fmla. fresh_in_fmla (fresh_from f) f
```

```
function abstract_effects (s: stmt) (f: fmla) : fmla
(* renvoie (forall w1,..,wk, f) ou w1,..,wk sont les variables
   affectees dans s *)
... (4 axiomes)
```

Propriété de correction

Rappel : Sémantique bloquante

Par définition, un programme annoté *respecte ses spécifications* si son exécution ne bloque pas

Théorème de correction

$\forall\ \Sigma\ \Pi\ s\ Q.$
Si $[\![\mathrm{WP}(s,\ Q)]\!]_{\Sigma,\Pi}$ alors

- Soit $\Sigma,\ \Pi,\ s \rightsquigarrow^*\ \Sigma',\ \Pi',\ \mathit{Sskip}$ et $[\![Q]\!]_{\Sigma',\Pi'}$
- Soit $\Sigma,\ \Pi,\ s$ se réduit indéfiniment

Préservation par réduction

Lemme de préservation

$$\forall\ \Sigma\ \Sigma'\ \Pi\ \Pi'\ s\ s'.\ \ \Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s' \Rightarrow$$
$$\forall\ Q.\ [\![\mathsf{WP}(s,\ Q)]\!]_{\Sigma,\Pi} \Rightarrow\ [\![\mathsf{WP}(s',\ Q)]\!]_{\Sigma',\Pi'}$$

Préservation par réduction

Lemme de préservation

$$\forall\ \Sigma\ \Sigma'\ \Pi\ \Pi'\ s\ s'.\ \ \Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s' \Rightarrow$$
$$\forall\ Q.\ [\![\mathsf{WP}(s,\ Q)]\!]_{\Sigma,\Pi} \Rightarrow\ [\![\mathsf{WP}(s',\ Q)]\!]_{\Sigma',\Pi'}$$

Preuve par induction sur l'hypothèse $\Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s'$

```
intros sigma sigma' pi pi' s s' h1.
induction h1; try (simpl; intro; ae).
simpl; intros q (_ & h).
generalize h; intro h'.
apply abstract_effects_specialize in h'; simpl in h'; ae.
simpl; intros q (_ & h).
apply abstract_effects_specialize in h; simpl in h; ae.
```

Progrès

$$\mathit{reducible}\ \Sigma\ \Pi\ s\ :=\ \exists \Sigma'\ \Pi'\ s'.\ \ \Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s'$$

Lemme de progrès

$\forall\ s\ \Sigma\ \Pi\ \Sigma_t\ \Pi_t\ Q.$
$\mathit{compatible\_env}\ \Sigma\ \Sigma_t\ \Pi\ \Pi_t\ \wedge\ \mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s\quad \wedge$
$[\![\mathsf{WP}(s,\ Q)]\!]_{\Sigma,\Pi} \wedge\ \ s \neq \mathit{Sskip}\ \Rightarrow\ \mathit{reducible}\ \Sigma\ \Pi\ s$

Progrès

$$\mathit{reducible}\ \Sigma\ \Pi\ s\ :=\ \exists \Sigma'\ \Pi'\ s'.\ \ \Sigma,\ \Pi,\ s \rightsquigarrow \Sigma',\ \Pi',\ s'$$

Lemme de progrès

$\forall\ s\ \Sigma\ \Pi\ \Sigma_t\ \Pi_t\ Q.$
$\mathit{compatible\_env}\ \Sigma\ \Sigma_t\ \Pi\ \Pi_t\ \ \wedge\ \mathit{type\_stmt}\ \Sigma_t\ \Pi_t\ s\quad \wedge$
$[\![\mathsf{WP}(s,\ Q)]\!]_{\Sigma,\Pi} \wedge\ \ s \neq \mathit{Sskip}\ \Rightarrow\ \mathit{reducible}\ \Sigma\ \Pi\ s$

Preuve avec la transformation "induction" de Why3

- 1 sous-but prouvé automatiquement : Sskip,
- 5 sous-buts prouvés en Coq (46 lignes)

Correction

Théorème de correction (formulation équivalente)

$$\begin{aligned}
&\forall\ n\ \Sigma\ \Sigma'\ \Pi\ \Pi'\ s\ s'\ \Sigma_t\ \Pi_t\ Q.\\
&\quad \mathit{compatible_env}\ \Sigma\ \Sigma_t\ \Pi\ \Pi_t \quad \wedge \quad \mathit{type_stmt}\ \Sigma_t\ \Pi_t t\ s \quad \wedge\\
&\quad \Sigma,\ \Pi,\ s \rightsquigarrow^n\ \Sigma',\ \Pi',\ s' \quad \wedge \quad \mathit{not}(\mathit{reducible}\ \Sigma'\ \Pi'\ s') \quad \wedge\\
&\quad [\![\mathsf{WP}(s,\ Q)]\!]_{\Sigma,\Pi} \quad \Rightarrow \quad s' = \mathit{Sskip} \wedge [\![Q]\!]_{\Sigma',\Pi'}
\end{aligned}$$

Preuve

En Coq, par induction sur n (22 lignes)

Conclusions

- Approche naturelle et simple à comprendre
  - Utilisée dans le cours *"Preuves de programmes"* au MPRI
- Études de fonctionnalités de l'environnement Why3
  - Types algébriques,
  - Définition de fonctions par récursion structurelle,
  - Définition de prédicats par induction,
  - Transformation d'induction structurelle de Why3,
  - Tactique `why3` dans Coq
- Degrés satisfaisant d'automatisation :
  - 390 lignes de code
  - 142 lignes de preuve
- Liens utiles :
  - `http://why3.lri.fr/`
  - `http://toccata.lri.fr/gallery/`

Perspectives

- Extension de l'étude
  1. Réalisation de `fresh_from` et `abstract_effects`
  2. Un langage avec fonctions et appels de fonction
- Développer du code correct par construction
  - Alternative à une approche tout en Coq
    - Plus d'automatisation
    - Programmes avec effets de bord
  - Mécanisme d'extraction de Why3 vers OCaml en cours de développement