Étude théorique et pratique d'un langage intermédiaire dédié à la preuve de programmes

soutenance M2 MPRI

Paul Patault

Jean-Christophe Filliâtre, Andrei Paskevich

LMF
Laboratoire
Méthodes
Formelles

Langage impératif minimal : While

```
let l = l₀
let r = r₀
while l ≠ nil do
    let h, t = l
    r := cons h r
    l := t
done
```

Langage impératif minimal : While

```
let l = l₀
let r = r₀
while l ≠ nil do
    let h, t = l
    r := cons h r
    l := t
done
assert { r = rev l₀ ++ r₀ }
```

Langage impératif minimal : While

```
let l = l₀
let r = r₀
while l ≠ nil do invariant { rev l ++ r = rev l₀ ++ r₀ }
    let h, t = l
    r := cons h r
    l := t
done
assert { r = rev l₀ ++ r₀ }
```

Conditions de vérification (VCs)

la formule suivante exprime la correction du programme

elle est obtenue par le calcul de plus faible pré-condition (WP)

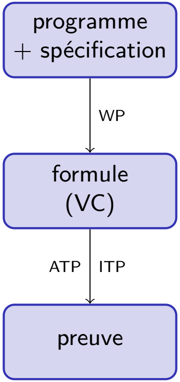
$$
\begin{array}{l}
\mathrm{I}[\ell_0, r_0] \wedge \\
\forall \ell, r.\ \mathrm{I}[\ell, r] \rightarrow \\
\quad\quad \textbf{if } \ell \neq \texttt{nil} \textbf{ then} \\
\quad\quad\quad\quad \ell \neq \texttt{nil} \wedge \\
\quad\quad\quad\quad \forall h, t.\ \ell = \texttt{cons}\ h\ t \rightarrow \mathrm{I}[t, \texttt{cons}\ h\ r] \\
\quad\quad \textbf{else } r = \texttt{rev}\ \ell_0 \texttt{ ++ } r_0
\end{array}
$$

où $\mathrm{I}[x, y] \triangleq$ `rev x ++ y = rev` $\texttt{l}_0$ `++` $\texttt{r}_0$

Why3

logiciel pour faire de la vérification déductive comprenant

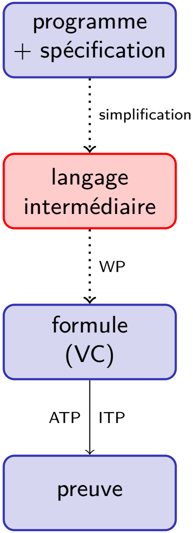
- un langage de haut niveau
  - de programmation
    $\rightarrow$ while + fonctions + ADTs + exceptions + ...
  - de spécification
    $\rightarrow$ assert + invariants + pre/post-conditions + code fantôme + ...
- un générateur de VCs (1300 loc)
- une interface avec 25+ démonstrateurs (automatiques et interactifs)



Why3

logiciel pour faire de la vérification déductive comprenant

- un langage de haut niveau
  - de programmation
    $\rightarrow$ while + fonctions + ADTs + exceptions + ...
  - de spécification
    $\rightarrow$ assert + invariants + pre/post-conditions + code fantôme + ...
- un générateur de VCs (1300 loc)
- une interface avec 25+ démonstrateurs (automatiques et interactifs)



Langage Coma

```
loop
/ loop =
      unList l (λht. assign &r (cons h r) (λ. assign &l t loop))
               out
/ out =
    halt
/ &r = r₀
/ &l = l₀
```

Langage Coma

```
loop
/ loop = assert { rev l ++ r = rev l₀ ++ r₀ }
    ↑ unList l (λht. assign &r (cons h r) (λ. assign &l t loop))
               out
/ out = assert { r = rev l₀ ++ r₀ }
    halt
/ &r = r₀
/ &l = l₀
```

Langage Coma

```
loop
/ loop = assert { rev l ++ r = rev l₀ ++ r₀ }
    ↑ unList l (λht. assign &r (cons h r) (λ. assign &l t loop))
               out
/ out = assert { r = rev l₀ ++ r₀ }
    halt
/ &r = r₀
/ &l = l₀
```

- langage intermédiaire pour la preuve de programmes
- barrière d'abstraction explicite : ↑
- calcul de conditions de vérification : $\mathrm{VC}(e)$

Traduction While vers Coma

Contribution 1

opérateur $[\![\ \textit{programme} \mid \textit{continuation},\ \ldots\ ]\!]$

$$[\![\ \texttt{let}\ x = e \mid k,\ \ \ldots\ \ ]\!] \triangleq k \,/\, \&x = e$$

$$[\![\ \texttt{let}\ x, y = e \mid k,\ \ \ldots\ \ ]\!] \triangleq \texttt{unList}\ e\ (\lambda ht.\, k \,/\, \&x = h \,/\, \&y = t)\ \texttt{absurd}$$

$$[\![\ i_1\ \texttt{;}\ i_2 \mid k,\ \ \ldots\ \ ]\!] \triangleq [\![\ i_1 \mid [\![\ i_2 \mid k,\ \ \ldots\ \ ]\!],\ \ \ldots\ \ ]\!]$$

$$\vdots$$

Traduction While vers Coma

Contribution 1

opérateur $\llbracket\, \textit{programme} \mid \textit{default}, \textit{break}, \textit{continue} \,\rrbracket$

$$
\begin{aligned}
\llbracket\, \texttt{let}\ x = e \mid k,\ b,\ c \,\rrbracket &\triangleq k \,/\, \&x = e \\
\llbracket\, \texttt{let}\ x, y = e \mid k,\ b,\ c \,\rrbracket &\triangleq \texttt{unList}\ e\ (\lambda ht.\, k \,/\, \&x = h \,/\, \&y = t)\ \texttt{absurd} \\
\llbracket\, i_1\ ;\ i_2 \mid k,\ b,\ c \,\rrbracket &\triangleq \llbracket\, i_1 \mid \llbracket\, i_2 \mid k,\ b,\ c \,\rrbracket,\ b,\ c \,\rrbracket \\
&\ \vdots \\
\llbracket\, \texttt{break} \mid k,\ b,\ c \,\rrbracket &\triangleq b \\
\llbracket\, \texttt{continue} \mid k,\ b,\ c \,\rrbracket &\triangleq c
\end{aligned}
$$

Traduction While vers Coma

Contribution 1

opérateur $[\![\, \textit{programme} \mid \textit{default}, \textit{break}, \textit{continue} \,]\!]$

$$
\begin{aligned}
[\![\, \texttt{let } x = e \mid k,\ b,\ c \,]\!] &\triangleq k \,/\, \&x = e \\
[\![\, \texttt{let } x, y = e \mid k,\ b,\ c \,]\!] &\triangleq \texttt{unList } e \; (\lambda ht.\, k \,/\, \&x = h \,/\, \&y = t) \; \texttt{absurd} \\
[\![\, i_1 \, \texttt{;} \, i_2 \mid k,\ b,\ c \,]\!] &\triangleq [\![\, i_1 \mid [\![\, i_2 \mid k,\ b,\ c \,]\!],\ b,\ c \,]\!] \\
&\vdots \\
[\![\, \texttt{break} \mid k,\ b,\ c \,]\!] &\triangleq b \\
[\![\, \texttt{continue} \mid k,\ b,\ c \,]\!] &\triangleq c
\end{aligned}
$$

initialisation $[\![\, P \mid \texttt{halt},\ \texttt{absurd},\ \texttt{absurd} \,]\!]$

Traduction selon la procédure

```
loop
/ loop
  = assert { rev l ++ r = rev l₀ ++ r₀ }
    ↑ if (l ≠ nil)
        (λ. unList l (λht.
                        assign &r (cons x r) (λ. assign &l s loop)
                        / &x = h / &s = t)
                      (λ. absurd))
        (λ. out)
/ out = ↓ assert { r = rev l₀ ++ r₀ } halt
/ &r = r₀
/ &l = l₀
```

Correction de la traduction

Contribution 2

La compilation de While vers Coma est correcte.

$\rightarrow$ préservation de la sémantique axiomatique de While

Théorème

Pour tout programme While P,

$$\mathrm{WP}(P, \top, \bot, \bot) \equiv \mathrm{VC}(\llbracket\, P \mid \texttt{halt},\ \texttt{absurd},\ \texttt{absurd} \,\rrbracket)$$

Méta-théorie du langage Coma

- Coma est un langage fortement typé
- système de types proche de Hindley-Milner
- garantit l'absence d'alias entre les variables mutables

$$\frac{\Gamma, \Delta' \vdash e : (\&r : \tau)\,\pi \qquad \Delta' \text{ est } \Delta \text{ sans les signatures de sous-routines}}{\Gamma, \&r : \tau, \Delta \vdash e\ \&r : \pi} \text{(T-AppR)}$$

$\rightarrow$ la référence r est retirée

$\rightarrow$ les sous-routines dans la portée lexicale de r sont retirés

Sûreté du typage

Contribution 3

«Well-typed expressions do not go wrong.» — Milner, 1978

Proposition : le typage de Coma est sûr

Pour toute expression close e, bien typée et entièrement appliquée,

si sa condition de vérification $\mathrm{VC}(e)$ est valide, alors soit $e \longrightarrow^{*} \texttt{halt}$,

soit e s'évalue indéfiniment.

Preuve :

- progrès : « un programme bien typé et correct est soit réductible, soit final »
- préservation de type : « le réduit d'un programme bien typé reste bien typé »
- préservation de VC : « le réduit d'un programme correct reste correct » (en cours)

Progrès du calcul

Contribution 3.1

Théorème : progrès

Une expression close, correcte et entièrement appliquée est réductible ou égale à `halt`.

$$\text{Si } \Gamma_{\text{prim}} \vdash e : \Box \text{ et la formule } \mathrm{VC}(e) \text{ est valide,}$$
$$\text{alors } e = \mathtt{halt} \text{ ou il existe } e' \text{ tel que } e \longrightarrow e'.$$

Preuve. Induction structurelle sur e.

Préservation du typage

Contribution 3.2

Théorème : préservation

La réduction préserve le typage : si $e \longrightarrow e'$ et $\Gamma_{\mathsf{prim}} \vdash e : \square$, alors $\Gamma_{\mathsf{prim}} \vdash e' : \square$.

Preuve. Pas facile ! (à cause de T-AppR)
induction sur la taille de la sous-expression réduite

Conclusion

trois contributions

1. compilation While vers Coma
2. preuve de correction de cette traduction

3.1 preuve du progrès de calcul

3.2 preuve de la préservation du typage

$\rightarrow$ ce travail a permis de déceler des problèmes dans les définitions initiales

Perspectives

- travail en cours sur une publication
- encore *beaucoup* de preuves à faire
  - $\rightarrow$ correction du calcul des effets
  - $\rightarrow$ correction de l'élimination de l'état mutable
  - $\rightarrow$ préservation de la VC
  - $\rightarrow$ correction de l'élimination du code fantôme
- implémentation de Coma
  - $\rightarrow$ intégration dans Why3
  - $\rightarrow$ comparer en pratique le calcul de VC de Coma avec l'état de l'art

réserve

Code fantôme

- uniquement à des fins de vérification
- pas d'effet observable à l'exécution du programme
- permet de se passer des existentiels
- exemple : reste de la division euclidienne

```
assert { a >= 0 ∧ b > 0 }
let ghost q = 0;
let r = a;
while r >= y do invariant { a = b * q + r ∧ 0 <= r }
  r := r - b;
  q := q + 1;
done;
assert { a = b * q + r }
assert { 0 <= r < b }
```

Non aliasing : T-AppR

la règle

$$\frac{\Gamma, \Delta' \vdash e : (\&r : \tau)\,\pi \qquad \Delta' \text{ est } \Delta \text{ sans les signatures de sous-routines}}{\Gamma, \&r : \tau, \Delta \vdash e\ \&r : \pi}\ \text{(T-AppR)}$$

permet de ne pas typer

```
h &r &r
/ &r = 55
/ h (&p &q : int) = ...
```

```
g &r
/ g (&p : int) = ... r ...
/ &r : int = 89
```

Définition du langage While

$$
\begin{array}{rcl}
\textit{instruction} & ::= & \texttt{halt} \\
& \mid & \texttt{skip} \\
& \mid & \texttt{break} \mid \texttt{continue} \\
& \mid & \texttt{assert}\ \textit{formula} \\
& \mid & \texttt{let}\ \textit{variable}\ \texttt{=}\ \textit{term} \\
& \mid & \texttt{let}\ \textit{variable}\ ,\ \textit{variable}\ \texttt{=}\ \textit{term} \\
& \mid & \textit{variable}\ \texttt{:=}\ \textit{term} \\
& \mid & \texttt{if}\ \textit{term}\ \texttt{then}\ \textit{instruction}\ \texttt{else}\ \textit{instruction} \\
& \mid & \texttt{while}\ \textit{term}\ \texttt{invariant}\ \textit{formula}\ \texttt{do}\ \textit{instruction}\ \texttt{done} \\
& \mid & \textit{instruction}\ \texttt{;}\ \textit{instruction}
\end{array}
$$

Traduction While vers Coma (1/2)

opérateur $[\![\, \textit{programme} \mid \textit{default},\ \textit{break},\ \textit{continue} \,]\!]$
initialisation $[\![\, P \mid \texttt{halt},\ \texttt{absurd},\ \texttt{absurd} \,]\!]$

$$
\begin{aligned}
[\![\, \texttt{assert}\ \varphi \mid k,\ b,\ c \,]\!] &\triangleq \{\varphi\}\ k \\
[\![\, x \texttt{ := } e \mid k,\ b,\ c \,]\!] &\triangleq \texttt{assign}\ \&x\ e\ (\rightarrow k) \\
[\![\, \texttt{let}\ x \texttt{ = } e \mid k,\ b,\ c \,]\!] &\triangleq k \,/\, \&x = e \\
[\![\, \texttt{let}\ x, y \texttt{ = } e \mid k,\ b,\ c \,]\!] &\triangleq \texttt{unList}\ e\ (h\ t \rightarrow k \,/\, \&x = h \,/\, \&y = t) \\
&\qquad\qquad\qquad (\rightarrow \texttt{absurd}) \\
&\vdots
\end{aligned}
$$

Traduction While vers Coma (2/2)

$$
\begin{aligned}
[\![\, \texttt{skip} \mid k,\ b,\ c \,]\!] &\triangleq k \\
[\![\, \texttt{break} \mid k,\ b,\ c \,]\!] &\triangleq b \\
[\![\, \texttt{continue} \mid k,\ b,\ c \,]\!] &\triangleq c \\
[\![\, \texttt{halt} \mid k,\ b,\ c \,]\!] &\triangleq \texttt{halt} \\
[\![\, i_1 \,;\, i_2 \mid k,\ b,\ c \,]\!] &\triangleq [\![\, i_1 \mid [\![\, i_2 \mid k,\ b,\ c \,]\!],\ b,\ c \,]\!] \\
[\![\, \texttt{if}\ c\ \texttt{then}\ i_1\ \texttt{else}\ i_2 \mid k,\ b,\ c \,]\!] &\triangleq \texttt{if}\ c\ (\to [\![\, i_1 \mid \texttt{out},\ b,\ c \,]\!]) \\
&\qquad\quad (\to [\![\, i_2 \mid \texttt{out},\ b,\ c \,]\!]) \\
&\quad /\ \texttt{out}\ [\bar{q}] = {\downarrow}\, k \\
[\![\, \texttt{while}\ c\ \texttt{invariant}\ \varphi\ \texttt{do}\ i\ \texttt{done} \mid k,\ b,\ c \,]\!] &\triangleq \texttt{loop} \\
&\quad /\ \texttt{loop}\ [\bar{q}] = \{\varphi\}\, {\uparrow} \\
&\qquad\quad \texttt{if}\ c \\
&\qquad\qquad (\to [\![\, i \mid \texttt{loop},\ \texttt{out},\ \texttt{loop} \,]\!]) \\
&\qquad\qquad (\to \texttt{out}) \\
&\quad /\ \texttt{out}\ [\bar{q}] = {\downarrow}\, k
\end{aligned}
$$

Sémantique axiomatique de While

$$
\begin{aligned}
\mathrm{WP}(\mathtt{skip}, K, B, C) &\triangleq K \\
\mathrm{WP}(\mathtt{break}, K, B, C) &\triangleq B \\
\mathrm{WP}(\mathtt{continue}, K, B, C) &\triangleq C \\
\mathrm{WP}(\mathtt{halt}, K, B, C) &\triangleq \top \\
\mathrm{WP}(\mathtt{assert}\ \varphi, K, B, C) &\triangleq \varphi \wedge (\varphi \rightarrow K) \\
\mathrm{WP}(x \mathrel{\mathtt{:=}} e, K, B, C) &\triangleq K[x \mapsto e] \\
\mathrm{WP}(\mathtt{let}\ x \mathrel{\mathtt{=}} e, K, B, C) &\triangleq K[x \mapsto e] \\
\mathrm{WP}(\mathtt{let}\ x, y \mathrel{\mathtt{=}} e, K, B, C) &\triangleq e \neq \mathtt{nil} \wedge \forall xy(e = \mathtt{cons}\ x\ y \rightarrow K) \\
\mathrm{WP}(i_1\ \mathtt{;}\ i_2, K, B, C) &\triangleq \mathrm{WP}(i_1, \mathrm{WP}(i_2, K, B, C), B, C) \\
\mathrm{WP}(\mathtt{if}\ c\ \mathtt{then}\ i_1\ \mathtt{else}\ i_2, K, B, C) &\triangleq \mathtt{if}\ c\ \mathtt{then}\ \mathrm{WP}(i_1, K, B, C) \\
&\qquad\qquad \mathtt{else}\ \mathrm{WP}(i_2, K, B, C) \\
\mathrm{WP}(\mathtt{while}\ c\ \mathtt{invariant}\ \varphi\ \mathtt{do}\ i\ \mathtt{done}, K, B, C) &\triangleq \varphi \wedge \\
&\quad \forall \overline{q}(\varphi \rightarrow \mathtt{if}\ c\ \mathtt{then}\ \mathrm{WP}(i, \varphi, K, \varphi)\ \mathtt{else}\ K)
\end{aligned}
$$

Correction de la traduction While vers Coma

Théorème : préservation de sémantique

Pour tout programme While P,

$$\mathsf{WP}(P, \top, \bot, \bot) \equiv \mathsf{VC}(\llbracket\, P \mid \mathtt{halt},\ \mathtt{absurd},\ \mathtt{absurd} \,\rrbracket)$$

Lemme : généralisation du théorème

Pour tout programme While P, et toutes expressions Coma k, b et c,

$$\mathsf{WP}(P, \mathsf{VC}(k), \mathsf{VC}(b), \mathsf{VC}(c)) \equiv \mathsf{VC}(\llbracket\, P \mid k,\ b,\ c \,\rrbracket)$$

Preuve. Par induction sur P :

$$\begin{aligned}
\mathsf{VC}(\llbracket\, \mathtt{assert}\ \varphi \mid k,\ b,\ c \,\rrbracket) &= \mathsf{VC}(\{\varphi\}\, k) \\
&= \varphi \wedge (\varphi \rightarrow \mathsf{VC}(k)) \\
&= \mathsf{WP}(\mathtt{assert}\ \varphi, \mathsf{VC}(k), \mathsf{VC}(b), \mathsf{VC}(c))
\end{aligned}$$

Barrières d'abstraction (1/3)

While + fonctions

la fonction sert de barrière d'abstraction, il y a deux vérifications (appelé/appelant) :

→ correction de la fonction

→ respect du contrat à chaque appel l'appelant

```
let f (args)
    requires { P[args] }
    ensures  { Q[args] }        ⇝ ∀args. P[args] → WP(e, Q[args])
= e
⋮
  f (args)                      ⇝ P[args]
⋮
  f (args')                     ⇝ P[args']
⋮
```

Barrières d'abstraction (2/3)

Modes

- $\mathsf{VC}^{\bot}_{\top}$: *mode appelé* (définition de sous-routine)
- $\mathsf{VC}^{\top}_{\bot}$: *mode appelant* (appel à une sous-routine)
- $\mathsf{VC}^{\top}_{\top}$: *mode total*
- $\mathsf{VC}^{\bot}_{\bot}$: *mode nul* (toujours vrai)

$$\mathsf{VC}^{\mathsf{p}}_{\mathsf{d}}(\uparrow e) \triangleq \mathsf{VC}^{\mathsf{d}}_{\mathsf{d}}(e) \qquad\qquad \mathsf{VC}^{\mathsf{p}}_{\mathsf{d}}(\downarrow e) \triangleq \mathsf{VC}^{\mathsf{p}}_{\mathsf{p}}(e)$$

Barrières d'abstraction (3/3)

Conditions de vérification pour Coma

- $\mathsf{VC}^{\perp}_{\top}$: *mode appelé* (définition de sous-routine)
- $\mathsf{VC}^{\top}_{\perp}$: *mode appelant* (appel à un sous-routine)
- $\mathsf{VC}^{\top}_{\top}$: *mode total*
- $\mathsf{VC}^{\perp}_{\perp}$: *mode nul* (toujours vrai)

$$\mathsf{VC}^{\mathsf{p}}_{\mathsf{d}}(\{\varphi\}\ e) \triangleq (p \to \varphi) \wedge (\varphi \to \mathsf{VC}^{\mathsf{p}}_{\mathsf{d}}(e))$$

$$\begin{aligned} \mathsf{VC}^{\mathsf{p}}_{\mathsf{d}}(e \,/\, h = d) \ &:\ \mathsf{VC}^{\mathsf{p}}_{\mathsf{d}}(e) \\ &\quad \text{et } \mathsf{VC}^{\perp}_{\mathsf{p}}(d) \text{ pour toutes valeurs de paramètres} \\ &\quad \text{et } \mathsf{VC}^{\top}_{\perp}(d) \text{ à chaque appel au sous-routine } h \end{aligned}$$

Progrès du calcul

Contribution 3.1

Théorème : progrès

Une expression close, correcte et de type $\square$ est réductible ou égale à `halt`.

Si $\Gamma_{\text{prim}} \vdash e : \square$ et la formule $\mathsf{VC}(e)$ est valide,

alors $e = \mathtt{halt}$ ou il existe e' tel que $e \longrightarrow e'$.

Preuve. Par induction sur e et par cas sur e_0 (où e_0 est minimal selon $e = e_0\ \bar{a} \mathbin{/\!/} \Lambda$) :

- assertion, $e_0 = \{\varphi\}\ e_1$: on peut appliquer la règle E-Assert car
  - $\rightarrow$ $\bar{a} = \square$ (par typage)
  - $\rightarrow$ $\mathsf{VC}(e)$ (par hypothèse)
  - $\rightarrow$ lemme : φ n'est pas affaiblie en traversant le contexte

$$\frac{\Vdash \varphi}{\{\varphi\}\ e \mathbin{/\!/} \Lambda \longrightarrow e \mathbin{/\!/} \Lambda}\ \text{E-Assert}$$

Préservation du typage

Contribution 3.2

$$
\begin{array}{rll}
\forall e, e'. & e = [\![e]\!]_\square \implies & \text{forme normale} \\
& \Gamma_{\text{prim}} \vdash e : \square \implies & \text{clos et appliqué entièrement} \\
& e \longrightarrow e' \implies & \text{se réduit} \\
\forall e_0, \Lambda_0, \Gamma_0, \bar{a}, \pi. & e = e_0 \ \bar{a} \,//\, \Lambda_0 \implies & \text{décomposition de } e \\
& (e_0 = h_{\text{prim}} \ \bar{a}_0 \implies \bar{a} = \square) \implies & \\
& (e_0 = \pi' \to d \implies \pi' = \square) \implies & \\
& \Gamma_0 \vdash e_0 : \pi \implies & \\
\exists e_0', \Lambda_0'. & e' = e_0' \ \bar{a} \,//\, \Lambda_0' \wedge & \text{décomposition de } e' \\
& \Lambda_0' \sim \Lambda_0 \wedge & \\
& \Gamma_0 \vdash e_0' : \pi & \text{permet de conclure}
\end{array}
$$

(Par induction sur la taille de la sous-expression e_0.)

Théorème complet

$$\begin{array}{ccccc} e & \longrightarrow & e' & \longrightarrow & \dots \\ \Gamma_{\mathsf{prim}} \vdash e : \Box & & \Gamma_{\mathsf{prim}} \vdash e' : \Box & & \\ \mathsf{VC}_{\top}^{\top}(e) & & \mathsf{VC}_{\top}^{\top}(e') & & \end{array}$$