Modularité pour les programmes de robots

Soutenance de Stage de M1 – Informatique

Lilian Besson

ENS Cachan
Département d'Informatique

Jeudi 5 Septembre 2013

Lilian.BESSON@ens-cachan.fr

http://www.dptinfo.ens-cachan.fr/~lbesson/stageM1/

Contexte humain

Quand?

- 10 semaines,
- 03 Juin → 09 Août.

Où?

À Londres, Royaume-Uni :

UCL University College London,

PPLV groupe "*Logique, Vérification, et Principes des langages de Programmation*".

Avec qui?

- Jules Villard,
- Peter O'Hearn.

Deux domaines

Intelligence Artificielle : domaine très large (I.A.)

Aujourd'hui, seulement de la **planification** :

- modéliser un monde dynamique,
- trouver des trajectoires dans ce monde.

Vérification (domaine de l'équipe *PPLV*)

Prouver une **spécification** d'un programme.
Exemple : $f(x) \stackrel{\text{def}}{=} x := x + 1$:

Effets de f **précondition** $x = 0$ et **postcondition** $x = 1$:

$$\{x = 0\} \quad x := x + 1 \quad \{x = 1\}$$

Non-effets de f si $y \neq x$, $y = 2$ est **conservé** :

$$\{x = 0 \wedge y = 2\} \quad x := x + 1 \quad \{x = 1 \wedge y = 2\}$$

Problématiques et buts

Différentes composantes :

décrire une axiomatisation $\rightsquigarrow$ **calcul des situations**,

programmer une fonction $\rightsquigarrow$ **langage GOLOG**,

prouver cette fonction $\rightsquigarrow$ **système de preuve HG**.

Pourquoi vouloir être modulaire ? **efficacité** (passage à l'échelle), **élégance**, et **simplicité** !

Différentes modularité :

décrire une axiomatisation la changer facilement **après**,

programmer une fonction intermédiaire, la spécifier, **puis** l'utiliser sans connaître son implémentation,

prouver cette fonction une seule fois, **puis** utiliser sa spécification partout ailleurs comme un axiome (sans le reprouver).

Problématiques et buts

Différentes composantes :

décrire une axiomatisation $\rightsquigarrow$ **calcul des situations**,

programmer une fonction $\rightsquigarrow$ **langage GOLOG**,

prouver cette fonction $\rightsquigarrow$ **système de preuve HG**.

Pourquoi vouloir être modulaire ? **efficacité** (passage à l'échelle), **élégance**, et **simplicité** !

Différentes modularité :

décrire une axiomatisation la changer facilement **après**,

programmer une fonction intermédiaire, la spécifier, **puis** l'utiliser sans connaître son implémentation,

prouver cette fonction une seule fois, **puis** utiliser sa spécification partout ailleurs comme un axiome (sans le reprouver).

Graphes = modèle de système dynamique

Exemple: un graph dirigé, $\mathcal{G} \stackrel{\text{def}}{=} \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}, \{\mathtt{A} \rightsquigarrow \mathtt{B}, \mathtt{B} \rightsquigarrow \mathtt{C}\}$

Recherche de chemin

Choisir u, v dans $\mathcal{V}$.
Y a-t-il un chemin $u \rightsquigarrow^* v$?

Deux réponses possibles :

Oui

$\mathtt{A} \rightsquigarrow^* \mathtt{C}$?
Oui, $\mathtt{A} \rightsquigarrow \mathtt{B} \rightsquigarrow \mathtt{C}$.

Non

$\mathtt{B} \rightsquigarrow^* \mathtt{A}$?
Non.

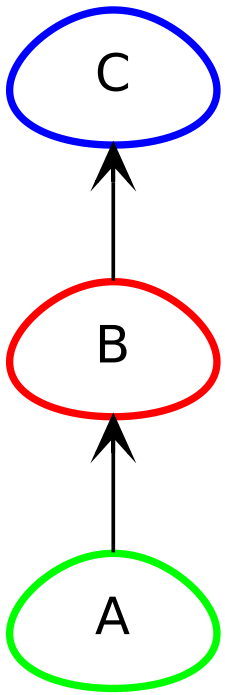

Figure 1: Le graphe $\mathcal{G}$.

Graphes = modèle de système dynamique

Exemple: un graph dirigé, $\mathcal{G} \stackrel{\text{def}}{=} \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}, \{\mathtt{A} \rightsquigarrow \mathtt{B}, \mathtt{B} \rightsquigarrow \mathtt{C}\}$

Recherche de chemin

Choisir u, v dans $\mathcal{V}$.
Y a-t-il un chemin $u \rightsquigarrow^* v$?

Deux réponses possibles :

Oui

$\mathtt{A} \rightsquigarrow^* \mathtt{C}$?
Oui, $\mathtt{A} \rightsquigarrow \mathtt{B} \rightsquigarrow \mathtt{C}$.

Non

$\mathtt{B} \rightsquigarrow^* \mathtt{A}$?
Non.

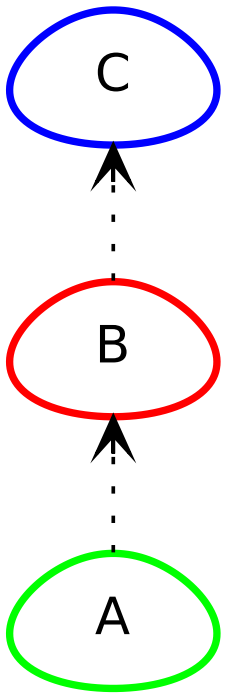

Figure 1: Le graphe $\mathcal{G}$.

Monde dynamique $\simeq$ un graphe

Jeu d'échec (Exemple)

Un *énorme* graphe fini.
$\longrightarrow$ Trop grand pour utiliser la théorie des graphes.

sommet $\Longleftrightarrow$ une "photo" du système $=$ un **état**,
arrête $\Longleftrightarrow$ transformation sur le système $=$ une **action**.

Monde des blocs (Exemple)

Blocs posés sur le sol – un robot pour les déplacer.

En pratique : trop d'états différents !

Graphe : pas utilisable, même pour des mondes "simples" en I.A.

Monde dynamique $\simeq$ un graphe

Jeu d'échec (Exemple)

Un *énorme* graphe fini.
$\longrightarrow$ Trop grand pour utiliser la théorie des graphes.

sommet $\Longleftrightarrow$ une "photo" du système $=$ un **état**,
arrête $\Longleftrightarrow$ transformation sur le système $=$ une **action**.

Monde des blocs (Exemple)

Blocs posés sur le sol – un robot pour les déplacer.

En pratique : trop d'états différents !

Graphe : pas utilisable, même pour des mondes "simples" en `I.A.`

Aperçu

États (Exemples)

- **objets** x, y, A
- **fluents** F, ParTerre
  - ⟶ **formules logiques** $\mathtt{F}(x, y)$, $\mathtt{ParTerre}(\mathtt{A})$

Transformations (Exemples)

- **actions** : a, BougeSur, BougeSol
  - ⟶ **triplets de Liu** $\{Q\}\ a\ \{R\}$

État initial

Figure 2: 3 blocs A, B et C sur le sol

Différents genres d'éléments

objets A, B, C

prédicats $\texttt{Libre}(x)$, $\texttt{ParTerre}(x)$ et ...

État initial

$$\begin{aligned}\Gamma_0 \stackrel{\text{def}}{=} & \texttt{Libre(A)} \wedge \texttt{ParTerre(A)} \\ & \wedge \texttt{Libre(B)} \wedge \texttt{ParTerre(B)} \\ & \wedge \texttt{Libre(C)} \wedge \texttt{ParTerre(C)}\end{aligned}$$

Destination = état but

Figure 3: Une tour: A sur B sur C

Différents genres d'éléments

objets A, B, C

prédicats $\texttt{Libre}(x)$, $\texttt{ParTerre}(x)$ et $\texttt{Sur}(x, y)$

État but

$$\Gamma_{but} \stackrel{\text{def}}{=} \texttt{Sur(A, B)} \wedge \texttt{Sur(B, C)}$$

Actions primitives de GOLOG

Déplacer x vers y : $\texttt{BougeSur}(x, y)$

$$\{\texttt{Libre}(x) \wedge (x \neq y) \wedge \texttt{Libre}(y)\}$$
$$\texttt{BougeSur}(x, y)$$
$$\{\texttt{Sur}(x, y)\}$$

Déplacer z depuis *un certain* w vers le sol : $\texttt{BougeSol}(z)$

$$\{\texttt{Libre}(z) \wedge (z \neq w) \wedge \texttt{Sur}(z, w)\}$$
$$\texttt{BougeSol}(z)$$
$$\{\texttt{ParTerre}(z) \wedge \neg\texttt{Sur}(z, w) \wedge \texttt{Libre}(w)\}$$

Actions primitives de GOLOG

Déplacer x vers y : $\texttt{BougeSur}(x, y)$

$$\{\texttt{Libre}(x) \wedge (x \neq y) \wedge \texttt{Libre}(y)\}$$
$$\texttt{BougeSur}(x, y)$$
$$\{\texttt{Sur}(x, y)\}$$

Déplacer z depuis *un certain* w vers le sol : $\texttt{BougeSol}(z)$

$$\{\texttt{Libre}(z) \wedge (z \neq w) \wedge \texttt{Sur}(z, w)\}$$
$$\texttt{BougeSol}(z)$$
$$\{\texttt{ParTerre}(z) \wedge \neg\texttt{Sur}(z, w) \wedge \texttt{Libre}(w)\}$$

Programme séquentiel = trajectoire

Proc $f(\mathtt{A}, \mathtt{B}, \mathtt{C})$: BougeSur(B, C) ; BougeSur(A, B) FinProc;

Figure 4: État initial Γ_0

Situation initiale : historique vide

$$\Gamma_0 \stackrel{\text{def}}{=} \mathtt{Libre(A)} \wedge \mathtt{ParTerre(A)} \wedge \mathtt{Libre(B)} \wedge \mathtt{ParTerre(B)} \wedge \mathtt{Libre(C)} \wedge \mathtt{ParTerre(C)}$$

Programme séquentiel = trajectoire

$$\texttt{Proc } f(\texttt{A},\texttt{B},\texttt{C}) : \underbrace{\texttt{BougeSur(B, C)}}_{\Gamma_0 \rightsquigarrow \Gamma_1} ; \texttt{BougeSur(A, B) FinProc;}$$

Figure 4: État intermédiaire Γ_1

Première action $\texttt{BougeSur(B, C)}$

$$\Gamma_1 \stackrel{\text{def}}{=} \texttt{Libre(A)} \wedge \texttt{ParTerre(A)} \wedge \texttt{Libre(B)} \wedge \underline{\texttt{Sur(B,C)}} \wedge \texttt{ParTerre(C)}$$

$$\{\texttt{Libre(B)} \wedge (\texttt{B} \neq \texttt{C}) \wedge \texttt{Libre(C)}\} \quad \texttt{BougeSur(B,C)} \quad \{\texttt{Sur(B,C)}\}$$

Programme séquentiel = trajectoire

Proc $f(\texttt{A}, \texttt{B}, \texttt{C})$: BougeSur(B, C) ; $\underbrace{\texttt{BougeSur(A, B)}}_{\Gamma_1 \rightsquigarrow \Gamma_{but}}$ FinProc;

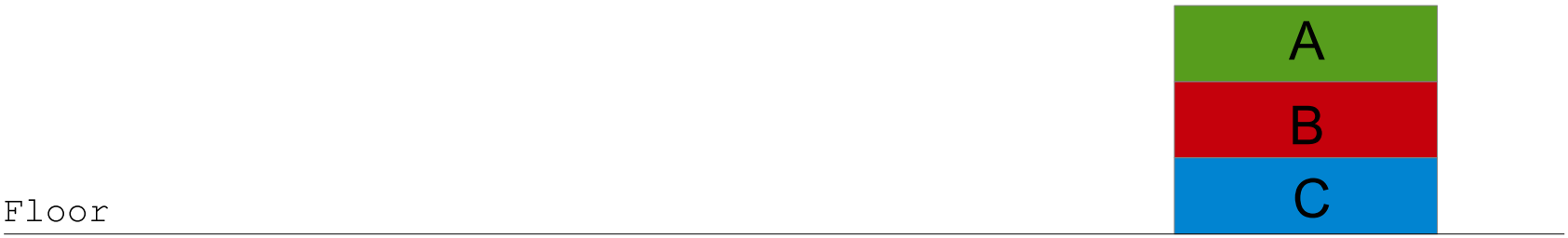

Figure 4: Destination $\Gamma_2 = \Gamma_{but}$

Seconde action BougeSur(A, B)

$\Gamma_2 = \Gamma_{but} \stackrel{\text{def}}{=} \texttt{Libre(A)} \wedge \underline{\texttt{Sur(A, B)}} \wedge \texttt{Sur(B,C)} \wedge \texttt{ParTerre(C)}$

$\{\texttt{Libre(A)} \wedge (\texttt{A} \neq \texttt{B}) \wedge \texttt{Libre(B)}\} \quad \texttt{BougeSur(A,B)} \quad \{\texttt{Sur(A,B)}\}$

BougeND : implémentation

Un programme GOLOG : BougeND(x, y, z)

pour déplacer x sur y **ou** sur z.

Proc BougeND(x, y, z) : BougeSur(x, y) | BougeSur(x, z) FinProc;

Spécification = **triplet de Liu** : $\{Q\}$ BougeND(x, y, z) $\{R\}$

$$\overbrace{\{\texttt{Libre}(x) \wedge (x \neq y) \wedge \texttt{Libre}(y) \wedge (x \neq z) \wedge \texttt{Libre}(z)\}}^{=Q}$$
$$\texttt{BougeND}(x, y, z)$$
$$\underbrace{\{\texttt{Sur}(x, y) \vee \texttt{Sur}(x, z)\}}_{=R}$$

Idée : Si $Q[s]$ est vrai, BougeND(x, y, z) *est applicable* et *termine* en partant de s, alors exécuter BougeND depuis $s \rightsquigarrow$ arriver en s', avec $R[s']$.

BougeND : implémentation et spécification

Un programme GOLOG : BougeND(x, y, z)

pour déplacer x sur y **ou** sur z.

Proc BougeND(x, y, z) : BougeSur(x, y) | BougeSur(x, z) FinProc;

Spécification = **triplet de Liu** : $\{Q\}$ BougeND(x, y, z) $\{R\}$

$$\overbrace{\{\texttt{Libre}(x) \wedge (x \neq y) \wedge \texttt{Libre}(y) \wedge (x \neq z) \wedge \texttt{Libre}(z)\}}^{=Q}$$

$$\texttt{BougeND}(x, y, z)$$

$$\underbrace{\{\texttt{Sur}(x, y) \vee \texttt{Sur}(x, z)\}}_{=R}$$

Idée : Si $Q[s]$ est vrai, BougeND(x, y, z) *est applicable* et *termine* en partant de s, alors exécuter BougeND depuis $s \rightsquigarrow$ arriver en s', avec $R[s']$.

$$\begin{aligned}
\textsf{Proc} \quad & \texttt{BougeND}(x,y,z) \ \texttt{:}\ \texttt{BougeSur}(x,y) \mid \texttt{BougeSur}(x,z) \ \textsf{FinProc} \\
Q &\stackrel{\text{def}}{=} \texttt{Libre}(x) \wedge x \neq y \wedge \texttt{Libre}(y) \wedge x \neq z \wedge \texttt{Libre}(z) \\
R &\stackrel{\text{def}}{=} \texttt{Sur}(x,y) \vee \texttt{Sur}(x,z) \\
\varphi(a,b) &\stackrel{\text{def}}{=} \texttt{Libre}(a) \wedge a \neq b \wedge \texttt{Libre}(b)
\end{aligned}$$

$$\textsc{Cons}\ \frac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,y)\}\ \texttt{Bge}(x,y)\ \{\texttt{Sur}(x,y)\}}}{\{Q\}\ \texttt{Bge}(x,y)\ \{R\}} \quad \textsc{ND}\ \frac{\qquad\qquad}{\{Q\}\ \texttt{Bge}(x,y) \mid \texttt{Bge}(x,z)\ \{R\}} \quad \frac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,z)\}\ \texttt{Bge}(x,z)\ \{\texttt{Sur}(x,z)\}}}{\{Q\}\ \texttt{Bge}(x,z)\ \{R\}}\ \textsc{Cons}$$

Figure 5: Schéma de preuve pour BougeND $\qquad (\texttt{Bge} \stackrel{\text{def}}{=} \texttt{BougeSur})$

Règle de choix : (ND)

Permet de séparer $\delta_1 \mid \delta_2$,

$$\textsc{ND}\ \frac{\{P\}\ \delta_1\ \{Q\} \qquad \{P\}\ \delta_2\ \{Q\}}{\{P\}\ \delta_1 \mid \delta_2\ \{Q\}}$$

$$\begin{aligned}
\textsc{Proc} \quad & \texttt{BougeND}(x,y,z) \ : \ \texttt{BougeSur}(x,y) \ | \ \texttt{BougeSur}(x,z) \ \textsc{FinProc} \\
Q &\stackrel{\text{def}}{=} \texttt{Libre}(x) \wedge x \neq y \wedge \texttt{Libre}(y) \wedge x \neq z \wedge \texttt{Libre}(z) \\
R &\stackrel{\text{def}}{=} \texttt{Sur}(x,y) \vee \texttt{Sur}(x,z) \\
\varphi(a,b) &\stackrel{\text{def}}{=} \texttt{Libre}(a) \wedge a \neq b \wedge \texttt{Libre}(b)
\end{aligned}$$

$$\textsc{ND}\ \frac{\textsc{Cons}\ \dfrac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,y)\}\ \texttt{Bge}(x,y)\ \{\texttt{Sur}(x,y)\}}}{\{Q\}\ \texttt{Bge}(x,y)\ \{R\}} \qquad \textsc{Cons}\ \dfrac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,z)\}\ \texttt{Bge}(x,z)\ \{\texttt{Sur}(x,z)\}}}{\{Q\}\ \texttt{Bge}(x,z)\ \{R\}}}{\{Q\}\ \texttt{Bge}(x,y)\,|\,\texttt{Bge}(x,z)\ \{R\}}$$

Figure 5: Schéma de preuve pour BougeND $\qquad (\texttt{Bge} \stackrel{\text{def}}{=} \texttt{BougeSur})$

Règle de conséquence : (Cons)

Deux choses: renforce la précondition, affaiblit la postcondition :

$$\textsc{Cons}\ \frac{\square\,(Q \Rightarrow Q') \qquad \{Q'\}\ \delta\ \{R'\} \qquad \square\,(R' \Rightarrow R)}{\{Q\}\ \delta\ \{R\}}$$

$$\begin{aligned}
\textsf{Proc} \quad & \texttt{BougeND}(x, y, z) \ : \ \texttt{BougeSur}(x, y) \ | \ \texttt{BougeSur}(x, z) \ \textsf{FinProc} \\
Q &\stackrel{\text{def}}{=} \texttt{Libre}(x) \wedge x \neq y \wedge \texttt{Libre}(y) \wedge x \neq z \wedge \texttt{Libre}(z) \\
R &\stackrel{\text{def}}{=} \texttt{Sur}(x, y) \vee \texttt{Sur}(x, z) \\
\varphi(a, b) &\stackrel{\text{def}}{=} \texttt{Libre}(a) \wedge a \neq b \wedge \texttt{Libre}(b)
\end{aligned}$$

$$\textsc{Nd}\ \dfrac{\textsc{Cons}\ \dfrac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,y)\}\ \texttt{Bge}(x,y)\ \{\texttt{Sur}(x,y)\}}}{\{Q\}\ \texttt{Bge}(x,y)\ \{R\}} \qquad \textsc{Cons}\ \dfrac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,z)\}\ \texttt{Bge}(x,z)\ \{\texttt{Sur}(x,z)\}}}{\{Q\}\ \texttt{Bge}(x,z)\ \{R\}}}{\{Q\}\ \texttt{Bge}(x,y)\,|\,\texttt{Bge}(x,z)\ \{R\}}$$

Figure 5: Schéma de preuve pour BougeND $\qquad (\texttt{Bge} \stackrel{\text{def}}{=} \texttt{BougeSur})$

Règle de conséquence : (Cons)

Deux choses: renforce la précondition, affaiblit la postcondition :

$$\textsc{Cons}\ \frac{\square\,(Q \Rightarrow Q') \qquad \{Q'\}\ \delta\ \{R'\} \qquad \square\,(R' \Rightarrow R)}{\{Q\}\ \delta\ \{R\}}$$

$$\begin{aligned}
\textsf{Proc} \quad & \texttt{BougeND}(x, y, z) \ \texttt{:}\ \texttt{BougeSur}(x, y) \ \texttt{|}\ \texttt{BougeSur}(x, z) \ \textsf{FinProc} \\
Q &\stackrel{\text{def}}{=} \texttt{Libre}(x) \wedge x \neq y \wedge \texttt{Libre}(y) \wedge x \neq z \wedge \texttt{Libre}(z) \\
R &\stackrel{\text{def}}{=} \texttt{Sur}(x, y) \vee \texttt{Sur}(x, z) \\
\varphi(a, b) &\stackrel{\text{def}}{=} \texttt{Libre}(a) \wedge a \neq b \wedge \texttt{Libre}(b)
\end{aligned}$$

$$\textsc{ND}\ \dfrac{\textsc{Cons}\ \dfrac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,y)\}\ \texttt{Bge}(x,y)\ \{\texttt{Sur}(x,y)\}}}{\{Q\}\ \texttt{Bge}(x,y)\ \{R\}} \qquad \textsc{Cons}\ \dfrac{\textsc{axiome}\ \dfrac{}{\{\varphi(x,z)\}\ \texttt{Bge}(x,z)\ \{\texttt{Sur}(x,z)\}}}{\{Q\}\ \texttt{Bge}(x,z)\ \{R\}}}{\{Q\}\ \texttt{Bge}(x,y)\ \texttt{|}\ \texttt{Bge}(x,z)\ \{R\}}$$

Figure 5: Schéma de preuve pour BougeND $\qquad (\texttt{Bge} \stackrel{\text{def}}{=} \texttt{BougeSur})$

Règle de conséquence : (Cons)

Deux choses: renforce la précondition, affaiblit la postcondition :

$$\textsc{Cons}\ \frac{\square\,(Q \Rightarrow Q') \qquad \{Q'\}\ \delta\ \{R'\} \qquad \square\,(R' \Rightarrow R)}{\{Q\}\ \delta\ \{R\}}$$

Programme BgeOu(x, y, z, w)

Rappel de BougeND

Proc BougeND(x, y, z) : BougeSur(x, y) | BougeSur(x, z) FinProc;

BgeOu(x, y, z, w)

pour déplacer un bloc x sur y **ou** z, **ou** w:

Proc BgeOu(x, y, z, w) : BougeND(x, y, z) | BougeSur(x, w) FinProc;

Spécification pour BgeOu

$Q' \stackrel{\text{def}}{=} \texttt{Libre}(x) \wedge (x \neq y) \wedge \texttt{Libre}(y) \wedge (x \neq z) \wedge \texttt{Libre}(z) \wedge (x, y, z \neq w) \wedge \texttt{Libre}(w)$

$$\{Q'\} \quad \texttt{BgeOu}(x, y, z, w) \quad \{\texttt{Sur}(x, y) \vee \texttt{Sur}(x, z) \vee \texttt{Sur}(x, w)\}$$

Preuve de BgeOu : ok :)

On veut **prouver** $\{Q'\}$ BgeOu $\{R'\}$

$$
\begin{aligned}
Q' &\stackrel{\text{def}}{=} \varphi(x,y) \wedge \varphi(x,z) \wedge y,z \neq w \wedge \varphi(x,w) \\
R' &\stackrel{\text{def}}{=} \texttt{Sur}(x,y) \vee \texttt{Sur}(x,z) \vee \texttt{Sur}(x,w) \\
\varphi(a,b) &\stackrel{\text{def}}{=} \texttt{Libre}(a) \wedge a \neq b \wedge \texttt{Libre}(b)
\end{aligned}
$$

$$
\text{ND}\ \dfrac{\text{Cons}\ \dfrac{\text{axiome}\ \dfrac{}{\{Q\}\ \texttt{BougeND}(x,y,z)\ \{R\}}}{\{Q'\}\ \texttt{BougeND}(x,y,z)\ \{R'\}} \qquad \text{Cons}\ \dfrac{\text{axiome}\ \dfrac{}{\{\varphi(x,w)\}\ \texttt{Bge}(x,w)\ \{\texttt{Sur}(x,w)\}}}{\{Q'\}\ \texttt{Bge}(x,w)\ \{R'\}}}{\{Q'\}\ \texttt{BougeND}(x,y,z) \,|\, \texttt{Bge}(x,w)\ \{R'\}}
$$

Figure 6: Schéma de preuvre pour BgeOu.

Programme BgeApres(x, y, z, w)

Rappel de BougeND

Proc BougeND(x, y, z) : BougeSur(x, y) | BougeSur(x, z) FinProc;

BgeApres(x, y, z, w)

utiliser BougeND(x, y, z), et **puis après** déplacer x sur w:

Proc BgeApres(x, y, z, w) : BougeND(x, y, z) ; BougeSur(x, w) FinProc;

Spécifications pour BgeApres

$$Q' \stackrel{\text{def}}{=} \text{Libre}(x) \wedge (x \neq y) \wedge \text{Libre}(y) \wedge (x \neq z) \wedge \text{Libre}(z) \wedge (x, y, z \neq w) \wedge \text{Libre}(w)$$

$$\{Q'\} \quad \text{BgeApres}(x, y, z, w) \quad \{\text{Sur}(x, w)\}$$

Preuve de BgeApres : soucis ! (1/2)

On veut prouver $\{Q'\}$ BgeApres $\{\texttt{Sur}(x,w)\}$

$$Q' \stackrel{\text{def}}{=} \varphi(x,y) \wedge \varphi(x,z) \wedge x,y,z \neq w \wedge \varphi(x,w)$$

$$\text{Invariant: } Inv \stackrel{\text{def}}{=} \texttt{Libre}(x) \wedge (x,y,z \neq w) \wedge \texttt{Libre}(w)$$

$$\varphi(a,b) \stackrel{\text{def}}{=} \texttt{Libre}(a) \wedge a \neq b \wedge \texttt{Libre}(b)$$

$$\dfrac{\dfrac{\dfrac{?\ \dfrac{}{\{Q_{\text{ND}}\}\ \texttt{BougeND}(x,y,z)\ \{R_{\text{ND}}\}}}{?!?}\ ?\ \dfrac{}{\{Q_{\text{ND}} \wedge Inv\}\ \texttt{BougeND}(x,y,z)\ \{R_{\text{ND}} \wedge Inv\}}}{\{Q'\}\ \texttt{BougeND}(x,y,z)\ \{R_{\text{ND}} \wedge Inv\}}\text{ons} \qquad \dfrac{\dfrac{}{\{\varphi(x,w)\}\ \texttt{Bge}(x,w)\ \{\texttt{Sur}(x,w)\}}\text{Axm.}}{\{R_{\text{ND}} \wedge Inv\}\ \texttt{Bge}(x,w)\ \{\texttt{Sur}(x,w)\}}\text{Con}}{\{Q'\}\ \texttt{BougeND}(x,y,z)\texttt{;Bge}(x,w)\ \{\texttt{Sur}(x,w)\}}\text{Seq}$$

Figure 7: Tentative d'un schéma de preuve pour BgeApres.

Preuve de BgeApres : soucis ! (2/2)

BgeApres(x, y, z, w)

pour déplacer un bloc x sur y **ou** z, et **puis après** déplacer x sur w:

Proc BgeApres(x, y, z, w) : BougeND(x, y, z) ; BougeSur(x, w) FinProc;

BougeND(x, y, z) doit conserver Libre$(x) \wedge x \neq w \wedge$ Libre(w)

$$\text{?}\ \frac{\text{?}\ \frac{\{Q_{\text{ND}}\}\ \texttt{BougeND}(x, y, z)\ \{R_{\text{ND}}\}}{?!?}}{\{Q_{\text{ND}} \wedge Inv\}\ \texttt{BougeND}(x, y, z)\ \{R_{\text{ND}} \wedge Inv\}}$$

Comment ?

En utilisant le corps de MoveND ? $\Longrightarrow$ **non-modulaire !**

Axiomes de HG pour les actions primitives

Axiomes de Frame et d'Effets (EF) — Réf. [Rei01]

F : fluent avec axiome *d'état successeur*
$\mathtt{F}(\overrightarrow{x},\ Faire(A, s)) \equiv \Phi_F(\overrightarrow{x}, A)[s]$:

$$\{\Phi_F(\overrightarrow{x_1}, A(\overrightarrow{x_2}))\} \quad A(\overrightarrow{x_2}) \quad \{\mathtt{F}(\overrightarrow{x_1})\}$$
$$\{\neg\Phi_F(\overrightarrow{x_1}, A(\overrightarrow{x_2}))\} \quad A(\overrightarrow{x_2}) \quad \{\neg\mathtt{F}(\overrightarrow{x_1})\}$$

Trop d'axiomes EF

Il y a en a 2 par fluents F et par actions A !

Source du soucis ?

Diagnostic

- "*Axiomes de Frame et d'Effet*"
  tant qu'on n'ajoute pas de nouvelles actions ou procédures,
- mais programmer en GOLOG =
  écrire des procédures intermédiaires et les utiliser ailleurs,
- possible d'actualiser les axiomes (EF), mais on doit prouver $2 \times Nb.Fluents$ lemmes par nouvelles procédures !

$\longrightarrow$ le système de preuve HG manque de modularité !

Générer les axiomes EF ?

En fait, on écrit des **"axiomes d'états successeur"** :

- un par fluent,
- mais de taille linéaire dans le nombre d'actions.

Conclusion

Nous avons présenté

situation c. décrire un monde dynamique,

GOLOG écrire des procédures pour ce monde,

HG prouver ces procédures.

Nous avons dévoilé des limitations pour HG

pas de bonne **abstraction procédurale** $\longrightarrow$ pas modulaire !

Approche concrète : formalisme, norme et outils

Aussi : STRIPS, PDDL et pyperplan

STRIPS autre formalisme,

PDDL norme pour STRIPS, développée pour l'IPC,

pyperplan solveur pour des problèmes de planification écrits en PDDL,

validate outil pour prouver la validité de trajectoires écrites en PDDL.

Contributions

- expériences et tests concrets:
  - ↪ faiblesses du formalisme,
  - ↪ faiblesses de ces outils,
- développer des techniques pour les résoudre.

Exemple d'expérience avec `pyperplan` Réf. [Mal11]

Limitations concrètes *(Exemple)*

↪ n'utilise pas le fait que certaines parties du monde sont disjointes : **perte de performance !**

Number of object	Number of world	Computation time (in s)	Lenght of the plan	Optimal plan
4,0,0,0	4	0.137	6	6
4,4,0,0	4	0.277	24	12
4,4,4,0	4	1.213	42	18
4,4,4,4	4	3.486	60	24

Figure 8: Union **disjointe** de mondes similaires, et un but croissant.

Quelles perspectives ?

Améliorer la compositionalité pour HG

trouver un moyen d'avoir un axiome de "frame" générique:

$$\text{Frame}\ \frac{\{Q_{\mathtt{P}}\}\ \mathtt{P}(x)\ \{R_{\mathtt{P}}\} \qquad \text{“}Q_{\mathtt{P}} * Inv\text{”}}{\{Q_{\mathtt{P}} \wedge Inv\}\ \mathtt{P}(x)\ \{R_{\mathtt{P}} \wedge Inv\}}$$

Implémentation ? Mise en pratique de ces idées ?

GOLOG ajouter procédures GOLOG à un *planning solver* (pyperplan),

HG ajouter HG à un *planning verifier* (validate).

Merci d'avoir écouté !

Des questions ?

Pour en savoir plus ?

- ↪ lire mon rapport, disponible en ligne ici :
  http://www.dptinfo.ens-cachan.fr/~lbesson/rapportM1Info13.pdf
- ↪ utiliser la bibliographie.

Bibliographie


Richard Fikes and Nils Nilsson.
[FN72] *Strips: A new approach to the application of theorem proving to problem solving*.
Artificial intelligence, 2(3):189–208, 1972.


Raymond Reiter.
[Rei01] *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*.
The MIT Press, 2001.


Yangmei Liu.
[Liu02] *A Hoare-style proof system for robot programs*.
AAAI, 2002.


Herbert Malte.
[Mal11] *pyperplan, a lightweight STRIPS planner, written in python 3*.
https://bitbucket.org/malte/pyperplan, 2011.

Un domaine pour la planification (Annexe)

```
(define (domain BLOCKS)
(:requirements :strips :typing)
(:types block)
(:predicates
       (On           ?x ?y  - block)
       (OnFloor      ?x     - block)
       (Clear        ?x     - block))
(:action Move
     :parameters   (?block1 ?block2 - block)
     :precondition (and (Clear ?block1) (Clear ?block2) (OnFloor ?block1))
     :effect (and (not (Clear ?block2)) (not (OnFloor ?block1)) (On ?block1 ?block2) ) )
(:action PutFloor
     :parameters   (?block1 ?block2 - block)
     :precondition (and (Clear ?block1) (On ?block1 ?block2))
     :effect (and (OnFloor ?block1) (Clear ?block2) (not (On ?block1 ?block2)) ) ) )
```

Figure 9: Une axiomatisation du monde des blocs en PDDL.

Remarque : on doit changer un peu : Move(x, y) demande à x d'être sur le sol.

Un problème de planification (Annexe)

```
;;; This goal is reachable, and the solver proves his reachability, by giving a solution
(define  (problem REACHABLE)
(:domain  BLOCKS)
(:objects A B C - block)
(:init    (Clear A) (OnFloor A) (Clear B) (OnFloor B) (Clear C) (OnFloor C) )
(:goal    (and (On A B) (On B C) ) ) )
;;; The generated solution  is: (Move B C) (Move A B)
```

Figure 10: Axiomatisation un but atteignable Γ_{goal} en PDDL

```
(move b c)
(move a b)
```

Figure 11: Une trajectoire trouvée par pyperplan