Calcul situationnel (situation calculus): Quelques mots sur “histoire” et références

(McCarthy & Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, Machine Intelligence IV, p.463–502, 1969).

Papier fondateur toujours très cité. Domaine toujours très actif:

Deux exemples de papiers plus récents:

(Pirri & Reiter, Some Contributions to the Metatheory of the Situation Calculus, J. ACM, Vol 46, No 3, p.335–361, May 1999)

(Levesque, Pirri & Reiter, Foundations for the Situation Calculus, ETAI Vol 3, nr 18, 1999)

Pour ces deux-là, cf le site du laboratoire de robotique cognitive à Toronto (http://www.cs.toronto.edu/cogrobo/).

Aussi le livre de R. Reiter: Knowledge in action, MIT Press, September 2001.

Calcul situationnel: Introduction

Il s’agit d’un langage logique, (du second ordre, mais cela intervient peu) qui sert à représenter des changements (ou évolutions).

Les changements proviennent d’une *action*.
Une *situation* est une histoire possible du monde, c’est-à-dire une suite d’actions.

Formellement, le langage comporte donc trois types d’objets (représentés comme d’habitude en logique par des termes):

- les ***situations***, dont l’ensemble (et le type) est noté Sit, les variables sont notées $s^{\cdots}_{\cdots}$, des termes $\sigma^{\cdots}_{\cdots}$;
- les ***actions***, dont l’ensemble est noté Act, les variables sont notées $a^{\cdots}_{\cdots}$ et des termes $\alpha^{\cdots}_{\cdots}$; et, pour tout le reste:
- les ***objets***, dont l’ensemble est noté Obj, et les variables sont notées par une autre minuscule (des termes par $\tau^{\cdots}_{\cdots}$).

Calcul situationnel: Introduction (2)

- Une constante particulière de type Sit, notée S_0, dénote la *situation initiale*.
- Un symbole de fonction particulier, de type $Act \times Sit \rightarrow Sit$, noté ici do (noté aussi $result$ ailleurs). $do(a,\ s)$ dénote la situation résultant de l'exécution de l'action a à partir de la situation s.

Le calcul situationnel présenté à l'origine était une logique *réifiée*:
des propriétés étaient représentées par des objets
(moins net pour la variante présentée ici que pour les versions originales, même si on peut considérer que do effectue une réification. Voir aussi 40 –42).

Exemples: $do(posersur(x, y),\ S_0)$
représente la situation résultant de l'action consistant à poser l'objet x sur l'objet y en partant de la situation initiale;

$do(poser(C),\ do(avancerde(L),\ do(prendre(C),\ s)))$
représente la situation résultant de la suite d'actions $[prendre(C),\ avancerde(L), poser(C)]$ en partant de la situation s.

Calcul situationnel: Introduction (3)

Plusieurs situations peuvent être associées au même instant:
situations hypothétiques: "si je fais ceci, alors,..." (futur)
"si j'avais fait ceci, alors,..." (passé, présent et futur), ...

"À part ça", au moins dans ce cours, le temps est linéaire.

À chaque *situation* s est associé un ***état***, qui est un ensemble maximal consistant de formules (ou un "modèle") représentant l'état du monde dans cette situation.
Cet état, bien que théoriquement *complet* (cf feuilles "Bases logiques" § 2.2, p.3), ne sera jamais calculé en entier:

on décrira et considérera quelques *faits* le concernant.

Ces faits sont décrits par des ***fluents***.

Calcul situationnel: fluents

- ***Fluents relationnels:*** ($n \in \mathbb{N}$)

symboles de prédicat de type $(Act \cup Obj)^n \times Sit$.
Décrivent les propriétés (relations) qui dépendent des situations.

Trois exemples: $surtable(x,\ s)$, $posésur(x,\ y,\ s)$,
$lit(Jean, Livre_1, s)$.

- ***Fluents fonctionnels:*** ($n \in \mathbb{N}$)

symboles de fonction de type $(Act \cup Obj)^n \times Sit \rightarrow Act \cup Obj$.
Permettent de construire des objets ou actions (dénotés par des termes), qui dépendent des situations.

Exemples: $age(Marie, s)$, $temps(s)$, $président(Italie, s)$ $(\in Obj)$
$posersur(x,\ y,\ s)$ $(\in Act)$.

Calcul situationnel: langage (suite)

Il y a aussi des symboles de prédicats et fonctions (y compris donc des constantes) dénotant des relations et fonctions indépendantes des situations ($n \in \mathbb{N}$):

- Prédicats: type $(Act \cup Obj)^n$,
- Fonctions: type $(Act \cup Obj)^n \rightarrow Obj$
- Fonctions d'action: type $(Act \cup Obj)^n \rightarrow Act$. (on a vu des exemples)

Exemples: prédicat: $homme(Jean)$, fonction: $altitude(Mt\text{-}Blanc)$,
fonctions d'action: $prendre(x)$, $posersur(x, y)$.

(Les fonctions d'action seront axiomatisées grâce à des
"*axiomes de pré-conditions d'action*".)

Remarque: Les seules fonctions à valeur dans Sit sont S_0 et do.

Calcul situationnel: deux prédicats particuliers

La variante du calcul situationnel considérée ici contient aussi:

- Un symbole de prédicat binaire $\sqsubset$, de type $Sit \times Sit$,
  qui définit une relation d'ordre strict (trans. + irréfl.) sur les situations
  (la fermeture transitive de $\sqsubset_1$ définie par $s \sqsubset_1 do(a, s)$).

  $s \sqsubset s'$ signifie: s est une sous-histoire commençante stricte de s'.

Exemple:
$do(a_2,\ do(a_1,\ S_0)) \quad \sqsubset \quad do(a_4,\ do(a_3,\ do(a_2,\ do(a_1,\ S_0))))$.

- Un symbole de prédicat binaire $Poss$, de type $Act \times Sit$,
  qui définit quand l'exécution d'une action est possible.

  $Poss(a,\ s)$ signifie: l'action a peut être exécutée en situation s.
  Cela permettra d'énoncer des *axiomes de pré-condition d'action*.

Calcul situationnel: Axiomes fondamentaux (1) et (2)

Le but est de donner un cadre logique naturel, simple et précis.
On se place en logique du second ordre (le second ordre intervient peu).
La calculabilité sera examinée plus loin . On se concentre ici sur les situations.

$$do(a_1, s_1) = do(a_2, s_2) \Rightarrow a_1 = a_2 \wedge s_1 = s_2 \qquad (1)$$

$$\forall P\ \ \{[P(S_0) \wedge \forall a, s\ [P(s) \Rightarrow P(do(a, s))]] \Rightarrow \forall s\ P(s)\} \qquad (2)$$

(1) est l'axiome de *séparation des constantes (SA)* (ou d'*unicité des noms (UN)*) pour les situations. Sit est un arbre de racine S_0 (situation $=$ chemin partant de S_0).

Rappel: *situation* $\neq$ *instant* (on peut introduire un fluent fonctionnel $temps(s)$).
Deux situations sont égales ssi elles racontent (sont) la même histoire (suite d'actions).

(2) est l'axiome d'*induction* sur les situations
(indispensable d'un point de vue formel) (cf l'axiome d'induction sur les entiers).

Calcul situationnel: Axiomes fondamentaux (3) et (4)

On définit aussi l'abréviation $\sqsubseteq$: $s_1 \sqsubseteq s_2$ si $s_1 \sqsubset s_2 \vee s_1 = s_2$.

$$\neg\ s \sqsubset S_0 \tag{3}$$

$$s \sqsubset do(a, s') \Leftrightarrow s \sqsubseteq s' \tag{4}$$

La relation $\sqsubset$ est un ordre strict sur Sit (donc $\sqsubseteq$ ordre). $s \sqsubset s'$ signifie que la suite d'actions s' est obtenue en ajoutant une suite non vide d'actions à s.

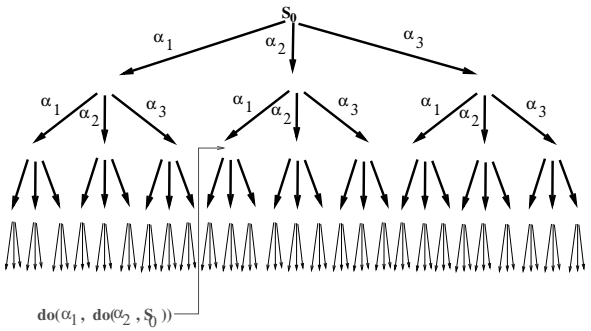
On appelle Σ l'ensemble des axiomes (1)–(4).

Ces axiomes ne dépendent pas du domaine considéré.

Ils décrivent un "*cadre situationnel*".

L'arbre des situations

Voici l'arbre des situations (une façon de représenter l'ensemble Sit), si l'ensemble Act des actions possède 3 éléments $\alpha_1, \alpha_2, \alpha_3$:



À chaque situation correspond un état du monde en cette situation. Un même état peut être satisfait en plusieurs situations distinctes.

Quelques conséquences des axiomes fondamentaux

$S_0 \neq do(a,s)$, $do(a,s) \neq s$.

$s = S_0 \ \vee \ \exists a, s' \ s = do(a,s')$ (existence d'un prédécesseur).

$S_0 \sqsubseteq s$, $\neg do(a,s) \sqsubseteq s$.

$\neg s \sqsubset s$ (irréflexivité). $s_1 \sqsubset s_2 \ \Rightarrow \ s_1 \neq s_2$ (unicité des noms).

$s_1 \sqsubset s_2 \ \Rightarrow \ \neg s_2 \sqsubset s_1$

$(s_1 \sqsubseteq s_2 \ \wedge \ s_2 \sqsubseteq s_1) \ \Rightarrow s_1 = s_2$ (anti-symétries).

$(s_1 \sqsubset s_2 \ \wedge \ s_2 \sqsubset s_3) \ \Rightarrow \ s_1 \sqsubset s_3$ (aussi pour $\sqsubseteq$) (transitivité).

Principe de double induction:

$$\forall R \left\{ \begin{bmatrix} R(S_0,S_0) \wedge \\ \forall a,s (R(s,s) \Rightarrow R(do(a,s),do(a,s))) \wedge \\ \forall a,s,s' ((s \sqsubseteq s' \wedge R(s,s')) \Rightarrow R(s,do(a,s'))) \end{bmatrix} \Rightarrow \forall s,s' (s \sqsubseteq s' \Rightarrow R(s,s')) \right\}.$$

Le problème de qualification pour les actions (qualification problem)

Exemple: Voici quelques ***pré-conditions*** pour l'action $prendre(r,x)$,

$Poss(prendre(r,x),s) \ \Rightarrow$

$(\forall z \ \neg tient(r,z,s)) \wedge \neg lourd(x) \wedge prochede(r,x,s)$ (Q).

On ne peut jamais déduire $Poss(prendre(r,x),s)$ à l'aide de cette formule.

Or, la réciproque de l'implication (Q) est fausse: on peut avoir aussi $colleausol(x,s) \Rightarrow \neg Poss(prendre(r,x),s)$.

Il n'est jamais possible/souhaitable d'énumérer toutes les conditions qui rendent une action possible.

On aimerait conclure $Poss(prendre(r,x),s)$ dès que les conditions (ou ***qualifications***) "importantes" sont établies (ici celles de (Q)), sans avoir besoin d'établir aussi les qualifications mineures, comme $\neg colleausol(x,s)$.

Le problème de qualification pour les actions (qualification problem) (suite)

Ce problème n'est pas spécifique aux actions, mais est général

(*règles avec exception*, comme "les oiseaux volent").

Cela rend nécessaire un raisonnement ***non monotone***:

un résultat vrai dans certaines circonstances peut devenir faux si on augmente notre connaissance sur les circonstances réelles.

Le problème du cadre (frame problem)

Contrairement au précédent, ce problème est spécifique aux actions, mais il est aussi important. Quand une action est exécutée, elle modifie quelques propriétés, et laisse les autres inchangées: il y a un "cadre fixe" devant lequel quelques personnages se déplacent (dessin animé).

Ainsi, déplacer un objet ne modifie pas sa couleur, ce qui sera traduit par un

axiome du cadre: $$couleur(x, s) = c \Rightarrow couleur(x, (do(déplacer(x), s))) = c.$$

Pour les fluents relationnels, on distingue les ***axiomes du cadre positifs***

$$cassé(x, s) \Rightarrow cassé(x, do(peindre(x), s)),$$

et les ***axiomes du cadre négatifs***: $$\{\neg cassé(x, s) \wedge [x \neq y \vee solide(x, s)]\} \Rightarrow \neg cassé(x, do(lâcher(r, y), s)).$$

Le problème est de trouver une méthode *modulaire* (de nouveaux fluents ou actions n'obligent pas à tout recalculer ou réécrire) pour générer tous ces axiomes, en partant uniquement des axiomes qui décrivent les effets des actions.

Si possible, on cherche aussi une représentation économique.

Situations exécutables

Toutes les situations de l'arbre p. 10 ne peuvent pas être atteintes. Elles doivent correspondre à une suite d'actions exécutables. Si $\neg tient(T, S_0)$, alors $do(poser(T), S_0)$ ne correspond pas à une situation possible.

On introduit l'abréviation suivante:

$$executable(s) \quad =_{def} \quad \forall a, s' \ (do(a, s') \sqsubseteq s \quad \Rightarrow \quad Poss(a, s')) \quad \text{(EXEC)}$$

Voici quelques conséquences des axiomes fondamentaux:

$executable(do(a, s)) \quad \Leftrightarrow \quad executable(s) \wedge Poss(a, s)$;

$executable(s) \quad \Leftrightarrow$
$[s = S_0 \vee \exists a, s' \ (s = do(a, s') \wedge Poss(a, s') \wedge executable(s'))]$;

$executable(s') \wedge s \sqsubseteq s' \quad \Rightarrow \quad executable(s)$.

Situations exécutables (suite)

Le principe d' induction est aussi vrai pour les situations exécutables:

$$\forall P \ \{[\, P(S_0) \wedge \forall a, s \ [(P(s) \wedge executable(s) \wedge Poss(a, s)) \Rightarrow P(do(a, s))]\,] \Rightarrow \forall s \ (executable(s) \Rightarrow P(s))\}$$

(*induction sur les situations exécutables*)

Ainsi, pour démontrer $\forall s \ executable(s) \Rightarrow \phi(s)$,
il suffit de démontrer $\phi(S_0)$ et
$\forall s \ ((\phi(s) \wedge executable(s) \wedge Poss(a, s)) \Rightarrow \phi(do(a, s)))$.

Le principe de la double induction est aussi vrai pour les situations exécutables,
Il sert à établir des formules du genre:

$$\forall s, s' (executable(s') \wedge s \sqsubseteq s') \Rightarrow R(s, s')).$$

Exemple de démonstration inductive

Circuit électrique avec deux interrupteurs Sw_1, Sw_2, une lampe, et une action $basculer(sw)$ qui manipule l'interrupteur sw.

$$ouvert(sw, do(a, s)) \Leftrightarrow [\neg ouvert(sw, s) \wedge a = basculer(sw) \vee ouvert(sw, s) \wedge a \neq basculer(sw)].$$

$$\begin{aligned}&éclaire(do(a, s)) \Leftrightarrow \\ &\{[\neg éclaire(s) \wedge (a = basculer(Sw_1) \vee a = basculer(Sw_2))] \vee \\ &[éclaire(s) \wedge a \neq basculer(Sw_1) \wedge a \neq basculer(Sw_2)]\}.\end{aligned}$$

On suppose qu'on a, en situation initiale:
$éclaire(S_0) \Leftrightarrow [ouvert(Sw_1, S_0) \Leftrightarrow ouvert(Sw_2, S_0)]$. ($D_0$)

Pour démontrer la formule
$\forall s\ éclaire(s) \Leftrightarrow [ouvert(Sw_1, s) \Leftrightarrow ouvert(Sw_2, s)]$, (D)
prendre pour formule $P(s)$:
$éclaire(s) \Leftrightarrow [ouvert(Sw_1, s) \Leftrightarrow ouvert(Sw_2, s)]$.

On a (D) en situation initiale (cf D_0), et la démonstration (fastidieuse...) marche, à condition de ***supposer l'unicité des noms pour les objets*** $(Sw_1 \neq Sw_2)$ ***et "les deux actions"*** $(basculer(sw))$.

Un exemple d'utilisation du calcul situationnel: une théorie élémentaire de l'action

On va décrire une théorie de l'action dans le langage $\mathcal{L}_{sit}$ du calcul situationnel, en ajoutant à la base Σ:

- une spécification de la situation initiale,
- des axiomes de succession d'états (un par fluent),
- des axiomes de pré-conditions d'actions (un par fonction d'action).

Définition utile: Une formule est ***uniforme en une situation*** σ si elle ne contient

- ni les prédicats $Poss$ et $\sqsubset$,
- ni aucun terme de type situation différent de σ comme argument d'un fluent,
- ni quantification ni égalité portant sur les situations.

Une théorie élémentaire de l'action (2)

- Un ***axiome de pré-condition d'action*** pour l'action représentée par le symbole de fonction n-aire A, est une formule close du type

$$Poss(A(x_1, \cdots, x_n), s) \Leftrightarrow \Pi_A(x_1, \cdots, x_n, s)$$

- Un ***axiome de succession d'états*** pour un fluent relationnel (n+1)-aire F est une formule du type

$$F(x_1, \cdots, x_n, do(a, s)) \Leftrightarrow \Phi_F(x_1, \cdots, x_n, a, s) \ (SEFR)$$

- Un ***axiome de succession d'états*** pour un fluent fonctionnel (n+1)-aire f est une formule du type

$$(f(x_1, \cdots, x_n, do(a, s)) = y) \ \Leftrightarrow \ \phi_f(x_1, \cdots, x_n, y, a, s) \ (SEFF)$$

Formules à droite de $\Leftrightarrow$: uniformes en s, que les variables libres indiquées.

On ne peut modéliser ainsi que des systèmes ***markoviens*** (seule la situation présente est prise en compte) et que des actions élémentaires.

Une théorie élémentaire de l'action (3)

$\mathcal{D}_{S_0}$: un ensemble de formules uniformes en S_0 (décrit la situation initiale, peut contenir des formules sans situation: des contraintes d'intégrité, des axiomes d'unicité des noms,...).

Soit $\mathcal{D}_{SE}$ et $\mathcal{D}_{PA}$ des ensembles d'axiomes, respectivement de succession d'états et de pré-conditions d'actions. Soit $\mathcal{D}_{UNA}$ l'ensemble des axiomes d'unicité des noms pour les actions. On considère des théories $\mathcal{D}$ de la forme

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{SE} \cup \mathcal{D}_{PA} \cup \mathcal{D}_{UNA} \cup \mathcal{D}_{S_0}.$$

Une ***théorie élémentaire de l'action*** est une telle théorie $\mathcal{D}$ qui vérifie la ***propriété de consistance des fluents fonctionnels***:
Pour tout fluent fonctionnel f dont l'axiome de succession d'états dans $\mathcal{D}_{SE}$ est (SEFF) , on a (où $\vec{x} = (x_1, \cdots, x_n)$):

$$\mathcal{D}_{UNA} \cup \mathcal{D}_{S_0} \models \forall \vec{x} \ \{\exists y \ \phi_f(\vec{x}, y, a, s) \wedge [\forall y, y' \ ((\phi_f(\vec{x}, y, a, s) \wedge \phi_f(\vec{x}, y', a, s)) \Rightarrow y = y')]\}.$$

On a alors le théorème de *satisfiabilité* (ou *consistance*) relative:
Une théorie élémentaire de l'action $\mathcal{D}$ est satisfiable ssi $\mathcal{D}_{UNA} \cup \mathcal{D}_{S_0}$ l'est.

Une “solution” au problème du cadre (dans une théorie élémentaire de l’action $\mathcal{D}$)

Axiomes d’effets et leur forme générale

axiomes d’effets positifs:

Exemple: Supposons qu’on ait deux ***axiomes d’effets positifs*** pour le fluent relationnel $cassé$
(si on lâche un objet fragile, ou si une bombe explose à proximité, l’objet devient cassé):

$fragile(x,s) \Rightarrow cassé(x, do(lâcher(r,x),s))$ et
$prochede(b,x,s) \Rightarrow cassé(x, do(exploser(b),s))$.

On réunit tous les axiomes d’effets positifs d’un fluent pour obtenir la ***forme générale*** *de l’axiome d’effets positifs* de ce fluent

(cf pp. 11–12 des feuilles “Bases logiques”, au sujet de la complétion de prédicats).

Ici, cela donnerait:

$\{\exists r\ [a = lâcher(r,x) \wedge fragile(x,s)]\ \vee$
$\exists b\ [a = exploser(b) \wedge prochede(b,x,s)]\}$
$\Rightarrow \quad cassé(x, do(a,s))$.

Une “solution” au problème du cadre (2)

Axiomes d’effets et leur forme générale (suite)

axiomes d’effets négatifs:

De même, si on a l’unique ***axiome d’effets négatifs*** de $cassé$
(si on répare un objet, il cesse d’être cassé):

$\neg cassé(x, do(répare(r,x),s))$, voici la

forme générale de l’axiome d’effets négatifs de $cassé$:

$[\exists r \quad a = répare(r,x)] \quad \Rightarrow \quad \neg cassé(x,\ do(a,s))$.

Une solution au problème du cadre (3)

Axiomes d'effets et hypothèse de fermeture causale

On fait l'***hypothèse de fermeture causale*** (cf la complétion de prédicat): les axiomes d'effets décrivent toutes les possibilités de ***modifications des fluents***, c'à-d
toutes les conditions dans lesquelles une action a peut rendre un fluent vrai (faux).

Attention, il ne s'agit pas ici d'ajouter les $\Leftarrow$ aux $\Rightarrow$ des deux formes générales des axiomes d'effets de $casse$, puisqu'on ne "***complète***" pas le ***fluent*** directement, mais ses ***modifications***. La formalisation exacte de cette notion est donnée page suivante, par les deux axiomes (FERN) et (FERP) et leur conséquence (PCFR) .

Une solution au problème du cadre (4)

Pour chaque fluent relationnel F, on a donc les deux *formes générales*:

axiomes d'effets positifs $\gamma^+(\vec{x}, a, s) \Rightarrow F(\vec{x}, do(a, s))$ $(EFRP)$

axiomes d'effets négatifs $\gamma^-(\vec{x}, a, s) \Rightarrow \neg F(\vec{x}, do(a, s))$ $(EFRN)$

La ***fermeture causale*** correspond aux ***axiomes de fermeture de l'explication***:

$$(F(\vec{x}, s) \wedge \neg F(\vec{x}, do(a, s))) \Rightarrow \gamma^-(\vec{x}, a, s) \qquad (FERN)$$
$$(\neg F(\vec{x}, s) \wedge F(\vec{x}, do(a, s))) \Rightarrow \gamma^+(\vec{x}, a, s) \qquad (FERP)$$

Si la ***condition de cohérence*** $\neg(\exists \vec{x}, a, s)\ (\gamma^+(\vec{x}, a, s) \wedge \gamma^-(\vec{x}, a, s))$
est satisfaite, alors la conjonction de (EFRP), (EFRN), (FERN) et (FERP) équivaut à l'axiome de succession d'états suivant pour F:

$$F(\vec{x}, do(a, s)) \Leftrightarrow [\gamma^+(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg\gamma^-(\vec{x}, a, s))] \quad (PCFR)$$

Une solution au problème du cadre (4')

Une curiosité logique au sujet de (PCFR) :

Comme on a (d'après la condition de cohérence)

$$\forall \vec{x}, a, s \ (\neg\gamma^{+}(\vec{x}, a, s) \vee \neg\gamma^{-}(\vec{x}, a, s))$$

on a les deux formes suivantes pour $F(\vec{x}, do(a, s))$

$$\gamma^{+}(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg\gamma^{-}(\vec{x}, a, s))$$

équivalent à

$$(\gamma^{+}(\vec{x}, a, s) \vee F(\vec{x}, s)) \wedge \neg\gamma^{-}(\vec{x}, a, s)$$

Une solution au problème du cadre (5)

Une solution similaire part des *axiomes d'effets pour les fluents fonctionnels*:
$\gamma_f(\vec{x}, y, a, s) \quad \Rightarrow \quad f(\vec{x}, do(a, s)) = y$ (EFF)

et ajoute l'axiome de fermeture de l'explication (pour les fluents fonctionnels):
$f(\vec{x}, do(a, s)) \neq f(\vec{x}, s) \quad \Rightarrow \quad \exists y \ \gamma_f(\vec{x}, y, a, s)$ (FEF).

C'est bien un ***axiome du cadre***, sa contraposée étant:
$\neg\exists y \ \gamma_f(\vec{x}, y, a, s) \quad \Rightarrow \quad f(\vec{x}, do(a, s)) = f(\vec{x}, s)$.

Comme pour les fluents propositionnels, moyennant une hypothèse de cohérence naturelle,
ici $\neg\exists \vec{x}, y, y', a, s \ \ [\gamma_f(\vec{x}, y, a, s) \wedge \gamma_f(\vec{x}, y', a, s) \wedge y \neq y']$
(conséquence de la propriété de consistance des fluents fonctionnels)

(EFF) et (FEF) sont équivalents à l'axiome de succession d'états suivant pour f:

$$f(\vec{x}, do(a, s)) = y \quad \Leftrightarrow$$
$$(\gamma_f(\vec{x}, y, a, s) \ \vee \ (f(\vec{x}, s) = y \ \wedge \ \nexists y' \ \gamma_f(\vec{x}, y', a, s))) \qquad \text{(PCFF)}$$

Une solution au problème du cadre (6)

- L'utilisateur fournit une description du problème considéré:
  - Axiomes d'ef fets (manière naturelle et modulaire de décrire le problème).
  - Un axiome de pré-condition d'actions pour chaque fonction d'action
    Là aussi, il existe des méthodes, non décrites dans ce cours (voir des pistes dans le cours sur calcul situationnel) qui permettent à l'utilisateur de fournir ces données de façon assez naturelle et modulaire.
- La formalisation "en interne" de cette solution consiste donc en ceci:
  - Les axiomes de succession d'états: un (PCFR) pour chaque fluent relationnel F et un (PCFF) pour chaque fluent fonctionnel f
    (issus des axiomes d'effets par la méthode donnée pages 21 – 26).
  - Un axiome de pré-condition d'actions pour chaque fonction d'action A: $Poss(A(\vec{x}), s) \Leftrightarrow \Pi_A(\vec{x}, s)$.
  - les axiomes d'unicité des noms d'action de $\mathcal{D}_{UNA}$, indispensables ici.

Une solution au problème du cadre (6')

La pertinence et la concision de cette méthode proviennent de:

- la quantification sur les actions,
- l'hypothèse que peu d'actions modifient un fluent, et bien sûr,
- le caractère assez élémentaire de cette théorie de l'action.

Un exemple de méthode facilitant les calculs
Régression en calcul situationnel

Il s'agit d'une méthode souvent utile, qui facilite les calculs effectués en interne, par le système.

On veut prouver qu'une formule W, qui mentionne le fluent relationnel $F(\vec{\tau}, do(\alpha, \sigma))$, est vraie dans une théorie élémentaire de l'action $\mathcal{D}$. L'axiome de succession d'états de F est $F(\vec{x}, do(a, s)) \Leftrightarrow \Phi_F(\vec{x}, a, s)$.

On peut donc remplacer $F(\vec{\tau}, do(\alpha, \sigma))$ par $\Phi_F(\vec{\tau}, \alpha, \sigma)$ dans W, ce qui simplifie la situation concernée. On peut ainsi revenir jusqu'à la situation initiale (d'où le terme *régression*).

Régression en calcul situationnel (2)

Voici une méthode possible.

On note $do([a_1,\cdots,a_n], s)$ pour $do(a_n,do(a_{n-1},\cdots,do(a_1, s),\cdots))$.

Une formule W de $\mathcal{L}_{sit}$ est ***régressable*** si:

- Les seuls termes de situation sont des $do([\alpha_1, \cdots, \alpha_n], S_0)$ (α_i termes d'action) et il n'y a pas de quantification sur les actions (c'à-d.: pas de variable de type situation).
- Dans les $Poss(\alpha, \sigma)$, α est $A(\vec{\tau})$ avec A symbole de fonction d'action (c'à-d.: pas de variable de type action ici).
- Il n'y a ni $\sigma \sqsubset \sigma'$ ni $\sigma = \sigma'$ pour des termes de situations.

Régression en calcul situationnel (3)

W est une formule régressable sans fluent fonctionnel. On définit $\mathcal{R}[W]$ ainsi:

- W atome.
  - Aucune situation ($=$ dans $Obj \cup Act$, ou non fluent): $\mathcal{R}[W] = W$.
  - W Fluent $F(\vec{\tau}, S_0)$: $\mathcal{R}[W] = W$.
  - W est $Poss(A(\vec{\tau}), \sigma)$: Alors, il existe un axiome de pré-condition d'action pour A, $Poss(A(\vec{x}), s) \Leftrightarrow \Pi_A(\vec{x}, s)$. Renommer les variables liées dans $\Pi_A(\vec{x}, s)$ afin qu'elles soient distinctes des variables libres, s'il y en a, de $Poss(A(\vec{\tau}), \sigma)$ et $\mathcal{R}[W] = \mathcal{R}[\Pi_A(\vec{\tau}, \sigma)]$.
  - W est $F(A(\vec{\tau}), do(\alpha, \sigma))$: Soit $F(\vec{x}, do(a, s)) \Leftrightarrow \Phi_F(\vec{x}, a, s)$ l'axiome de succession d'état de F. Renommer les variables liées de $\Phi_F(\vec{x}, a, s)$ et $\mathcal{R}[W] = \mathcal{R}[\Phi_F(\vec{\tau}, \alpha, \sigma)]$.
- $\mathcal{R}[\neg W] = \neg\mathcal{R}[W]$, $\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2]$, $\mathcal{R}[\exists v\ W] = \exists v\ \mathcal{R}[W]$.

Régression en calcul situationnel (4)

Si W a des fluents fonctionnels, c'est plus compliqué, mais faisable.

On obtient: Si W est une formule régressable de $\mathcal{L}_{sit}$, et si $\mathcal{D}$ est une théorie élémentaire de l'action, on a le ***théorème de la régression***:

- $\mathcal{D} \models \mathcal{W} \Leftrightarrow \mathcal{R}[\mathcal{W}]$, et aussi
- $\mathcal{D} \models \mathcal{W}$ ssi $\mathcal{D}_{S_0} \cup \mathcal{D}_{UNA} \models \mathcal{R}[W]$.

Ce résultat est important: une fois calculé $\mathcal{R}[W]$ (ce qui peut être complexe), on n'a plus besoin de considérer que la théorie de la situation initiale $\mathcal{D}_{S_0}$ (par exemple, on n'a plus besoin des axiomes fondamentaux, ni des axiomes de succession d'états ou de pré-conditions d'action) et $\mathcal{D}_{UNA}$.

C'est un exemple de *compilation des connaissances* (i.e. *knowledge compilation*).

Régression en calcul situationnel (5)
Exemples de formules régressables

- Exemple 1: $\exists p \; \{Poss(marcher(A, bureau(p)), S_0) \wedge$
  $Poss(entrer(bureau(p)), do(marcher(A, bureau(p)), S_0)) \wedge$
  $Poss(donnerCafé(p)), do([marcher(A, bureau(p)),$
  $entrer(bureau(p))], S_0)\}.$

  Sens: Est-il possible, à partir de S_0, d'exécuter la suite d'actions $[marcher(A, bureau(p)), entrer(bureau(p)), donnerCafé(p)]$?

- Exemple 2: $\exists x, y \; \{x \neq y \wedge \forall z \; (z \neq A \Rightarrow$
  $libre(z, do([poserSurTable(x), poserSur(y, A)], S_0)))\}.$

  Sens: Existe-t'il deux blocs tels que, après que l'un soit posé sur la table puis l'autre posé sur A, tous les blocs autres que A soient libres?

- Exemple 3: $vitesse(x, do(lâcher(x), S_0)) \; \leq$
  $vitesse(y, do([prendre(y), jeter(y)], S_0))$.

- Contre-exemples: Ni $Poss(a, S_0)$
  ni $tenir(x, do(prendre(A), s))$ ne sont régressables.

Actions complexes

Les seules actions vues jusqu'ici sont très élémentaires.

On présente ici une possibilité d'exprimer des actions moins élémentaires: ***conditionnelles***, ***boucles***, et "***procédures***".

On utilise pour cela des abréviations permettant d'utiliser de façon lisible des formules plus complexes dans $\mathcal{L}_{sit}$.

Un prédicat Do est introduit, pour formaliser les actions complexes:

$Do(\delta, s, s')$ signifie qu'il est possible de passer de la situation s à la situation s' en exécutant une suite δ d'actions complexes.

Les actions complexes peuvent être indéterministes (plusieurs s' possibles).

Actions complexes: une notation naturelle

Il est commode d'introduire $\alpha[\sigma]$ et $\phi[\sigma]$ (σ: terme de type situation).

- Si α est un terme d'action, $\alpha[\sigma]$ dénote le résultat de la restauration de l'argument σ à tous les fluents fonctionnels mentionnés par le terme α.

Ainsi, si α est $lire(livreFavori(Jean, s))$, où $livreFavori$ est un fluent fonctionnel, alors $\alpha[\sigma]$ est $lire(livreFavori(Jean, \sigma))$.

- Si $\phi[s]$ est une formule de $\mathcal{L}_{sit}$ qui mentionne la situation s comme seule variable de situation, on notera ϕ le "pseudo-fluent" associé et $\phi[\sigma]$ la formule obtenue en substituant σ à s dans $\phi[s]$ (en λ-notation, $\phi = \lambda s.\phi[s]$ et $\phi[\sigma] = (\lambda s.\phi[s])(\sigma)$).

Ainsi, si $\phi[s]$ est la formule $\forall x \;\; surTable(x, s) \wedge \neg sur(x, A, s)$, alors ϕ désigne le "pseudo-fluent" $\forall x \;\; surTable(x) \wedge \neg sur(x, A)$ (qui n'est pas une vraie formule du langage considéré dans cet exemple) et $\phi[\sigma]$ désigne la formule $\forall x \;\; surTable(x, \sigma) \wedge \neg sur(x, A, \sigma)$.

Actions complexes: définition de Do

- Actions primitives:
  $Do(a, s, s') =_{def} Poss(a[s], s) \;\wedge\; s' = do(a[s], s)$.
- Actions de test: $Do(\Phi?, s, s') =_{def} \Phi[s] \;\wedge\; s = s'$.
- Suites d'actions:
  $Do([\delta_1; \delta_2], s, s') =_{def} \exists s'' \; Do(\delta_1, s, s'') \;\wedge\; Do(\delta_2, s'', s')$.
- Choix non déterministe entre deux actions:
  $Do((\delta_1 \mid \delta_2), s, s') =_{def} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$
- Choix non déterministe d'arguments d'actions:
  $Do((\pi\; x)\; \delta(x), s, s') =_{def} \exists x \;\; Do(\delta(x), s, s')$.
- Itération non déterministe d'actions (exécuter δ, 0 ou plusieurs fois):
  $Do(\delta^*, s, s') =_{def}$
  $\forall P \; \{\{\forall s_1 \, P(s_1, s_1) \;\wedge\; \forall s_1, s_2, s_3 \, ((P(s_1, s_2) \wedge$
  $Do(\delta, s_2, s_3)) \;\Rightarrow\; P(s_1, s_3))\} \;\Rightarrow\; P(s, s')\}$.
  (Second ordre: (s, s') est "dans tout P" – donc "dans les plus petits" – qui contient tous les (s_1, s_1) et tous les (s_1, s_3) tels que P contient (s_1, s_2) et que δ fait passer de s_2 à s_3.)

Actions complexes: deux classiques

- **si ϕ alors δ_1 sinon δ_2 finsi** $=_{def}$ $[\Phi?;\delta_1] \mid [\neg\Phi?;\delta_2]$.
- **tantque ϕ faire δ fintantque** $=_{def}$ $[\Phi?;\delta]^*;\neg\Phi?$.
- On pourrait aussi définir des “actions procédures” de cette façon (les appels récursifs posent quelque problème de définition, là encore, une formule du second ordre intervient).

Soit δ une action complexe, δ correspond à un ***programme***. On a:

$$\Sigma \models \forall s\ Do(\delta, S_0, s) \Rightarrow executable(s).$$

GOLOG: un langage fondé sur cette variante du calcul situationnel

Le langage GOLOG (implémenté en PROLOG) correspond assez précisément à la variante du calcul de situations exposée ici (cf les deux papiers et le livre de Reiter et généralement le site cogrobo).

GOLOG permet par exemple de faire de la *planification* et du *diagnostique*.

Il existe (ou a existé?) même deux “robots distributeurs de courrier” (dans des bureaux de l’université de Toronto) programmés principalement en GOLOG.

Il existe aussi une extension (Legolog) à GOLOG, pour les petits robots MINDSTORMS Robotics Invention System de LEGO (extension due au laboratoire de robotique cognitive de Toronto, pas associée du tout – “in no way” – à LEGO, à but “pédagogique”), cf site cogrobo et précisément
Levesque et Pagnucco. Legolog: Inexpensive Expriments in Cognitive Robotics. Second International Cognitive Robotics Workshop, Berlin, 2000.

Actions complexes: remarques

Tel que défini formellement, $executable(s)$ ne signifie pas qu'un robot ait réellement assez d'information pour effectuer une tâche décrite. Il faut s'assurer que tel est le cas, en précisant les préconditions par exemple.

Ce petit exemple précise aussi la signification des actions composées décrites:
Un robot arrive devant deux portes, derrière l'une il y a un trésor, et derrière l'autre un monstre. Il faut ouvrir la bonne porte. L'action composée non déterministe

$[Ouvrir(Porte1) \mid Ouvrir(Porte2)] \; ; \; TrouvéTrésor?$

décrit l'action voulue mais, sans plus d'information, ne permet pas de l'effectuer.

Supposons maintenant le robot muni d'un capteur sachant voir à travers les portes.

$Voir\text{-}derrière; (\pi p)[Porte(p)?; Ouvrir(p)]; TrouvéTrésor?$

décrit l'action demandée à ce robot très perfectionné, capable aussi d'anticiper.

Cet exemple (un peu magique par son capteur, mais significatif de la puissance d'expression du langage) illustre aussi la différence entre les actions complexes

- $\phi?; \delta$: l'action δ se produit quand le fluent ϕ est vrai, et
- $\delta; \phi?$: l'action δ se produit, après quoi le fluent ϕ est vrai.

Version réifiée des actions complexes

Un inconvénient de cette méthode est que les "termes" δ représentant des actions complexes ne sont pas des termes du langage $\mathcal{L}_{sit}$
(ils ne sont pas *réifiés*, ils correspondent à des *macros*).

On peut adopter une autre approche, et ***réifier les actions complexes***:

On introduit alors de ***nouveaux symboles de fonctions*** "$?$", "$;$", "$|$", "π" , et si on veut "$si-alors-sinon$" etc...

Le ϕ de "$si \;\; \phi \;\; alors \cdots$" doit donc être lui aussi un ***terme***. Il faut donc réifier les "formules" comme $surTable(bloc)$, qui vient de la vraie formule $surTable(bloc, s)$.

Il faut aussi **axiomatiser la correspondance** entre un pseudo-fluent ϕ et le terme qui lui correspond. C'est parfois préférable, mais trop souvent cela compliquerait beaucoup les choses, d'où l'approche décrite en pages précédentes.

Bien sûr, l'expressivité de la méthode par réification est plus grande.

Version réifiée des actions complexes: l'expressivité est très grande

Avec la version réifiée, on peut quantifier sur les actions complexes
(penser "sur les programmes").

Ainsi, pour formaliser un problème de planification, on écrirait, où

- $Axioms$ représente Σ plus les axiomes de successions d'états et de pré-conditions d'actions particuliers au problème, et
- $But(s)$ est la formule qui décrit les propriétés que l'on souhaite obtenir:

$$Axioms \models \exists \delta, s \;\; Do(\delta, S_0, s) \wedge But(s).$$

Version réifiée des actions complexes: c'est souvent évitable

Il demeure de nombreuses propriétés exprimables avec la version non réifiée:

- ***Correction de programme.*** Pour montrer que, si un programme δ termine, il conduit à une situation satisfaisant la propriété P:
  - $Axioms \models \forall s \; Do(\delta, S_0, s) \Rightarrow P(s)$, ou, plus fort:
  - $Axioms \models \forall s', s \; Do(\delta, s', s) \Rightarrow P(s)$.
- ***Terminaison de programme.*** Montrer que δ se termine:
  - $Axioms \models \exists s \; Do(\delta, S_0, s)$, ou, plus fort:
  - $Axioms \models \forall s' \; \exists s \; Do(\delta, s', s)$.

Cette méthode suffit donc quand un programme δ est donné.
Ainsi, pour trouver une situation terminale de δ, s'il en existe, on prouve (avec une méthode constructive) la terminaison de δ et on extrait une assignation possible de la situation terminale de cette preuve.

Limites de la version présentée jusqu'ici

On peut programmer un robot "Off line" par la méthode présentée: Il réagira correctement si on a pu prévoir tout ce qui se produit dans tous les cas de figures.

Un vrai robot doit aussi pouvoir réagir avec son environnement. Voici quelques aspects non encore abordés sérieusement (certains seront évoqués dans la suite):

- ***Perception*** (vue, etc..). Cela modifie l'état des connaissances que le robot a sur le monde.
- ***Actions extérieures*** (hors du contrôle du robot considéré): appuyer sur le bouton d'un étage pour un ascenseur (le "robot" ici).

Les actions de test vérifient qu'une action s'est déroulée comme prévu, et sont également utilisables dans ces deux cas: $TEST_P$ lit la valeur d'une formule P.

- ***Actions concurrentes***.
- ***Processus continus*** et autres problèmes liés au ***temps***.

Temps et actions concurrentes

Jusqu'ici, on n'a considéré en fait que des actions "instantanées" ou presque.

Une action qui dure peut être représentée ainsi:

Exemple: $Marcher$ sera formalisée par

- deux actions $débutMarcher$ et $finMarcher$,
- un fluent relationnel $marche$ (et un fluent fonctionnel $lieu$).

dont voici les axiomes de pré-condition et de succession d'états:

$Poss(débutMarcher(x,y),s) \Leftrightarrow$
$\neg\exists u,v\ marche(u,v,s) \wedge lieu(s) = x,$

$Poss(finMarcher(x,y),s) \Leftrightarrow marche(x,y,s),$

$marche(x,y,do(a,s)) \Leftrightarrow [(a = débutMarcher(x,y)) \vee$
$(marche(x,y,s) \wedge a \neq finMarcher(x,y))],$

$lieu(do(a,s)) = y \Leftrightarrow [\exists x\ (a = finMarcher(x,y)) \vee$
$(lieu(s) = y \wedge \neg\exists x,y'\ (a = finMarcher(x,y')))].$

Temps et actions concurrentes (2)

On peut représenter ainsi des suites assez complexes d'actions qui durent.

Exemple (Eau chaude, eau froide):

$$\begin{array}{l}
Poss(ouvrirChaud, s) \Leftrightarrow \neg chaud(s), \\
Poss(ouvrirFroid, s) \Leftrightarrow \neg froid(s), \\
Poss(fermerChaud, s) \Leftrightarrow chaud(s), \\
Poss(fermerFroid, s) \Leftrightarrow froid(s), \\
chaud(do(a, s)) \Leftrightarrow [a = ouvrirChaud \vee \\
\qquad (chaud(s) \wedge a \neq fermerChaud)]. \\
froid(do(a, s)) \Leftrightarrow [a = ouvrirFroid \vee \\
\qquad (froid(s) \wedge a \neq fermerFroid)].
\end{array}$$

Axiome de succession d'états pour conditions de brûlure:

$$\begin{array}{l}
brule(do(a, s)) \Leftrightarrow [(chaud(s) \wedge a = fermerFroid) \vee \\
\qquad (\neg froid(s) \wedge a = ouvrirChaud) \vee brule(s)].
\end{array}$$

Une limite de cette méthode, qui reste séquentielle, est que, si deux actions démarrent exactement en même temps, il faut qu'il soit peu important que l'une se produise "un peu avant" l'autre: duel avec coups mortels simultanés inexprimable.

Calcul situationnel séquentiel temporel

Là encore, on présente une extension possible pour traiter le problème (ici utilisation du "temps daté"). On explicite le temps, toujours avec la notion d'actions instantanées, en ajoutant un argument temporel à toutes les actions. On dénote ainsi par *taper(mur, balle, 7.3)* l'action d'une *balle* qui tape dans un *mur* au temps 7.3.

On ajoute un axiome fondamental et deux symboles de fonction à $\mathcal{L}_{sit}$:

- Le symbole de fonction $temps$, de type $Act \rightarrow \mathbb{R}$.
  $temps(a)$ dénote quand se produit l'action a. Pour chaque application concernant une action particulière $A(\vec{x}, t)$, on a donc besoin de l'axiome donnant le temps de l'action A: $temps(A(\vec{x}, t)) = t$.
- Le symbole de fonction $début$, de type $Sit \rightarrow \mathbb{R}$,
  dénote l'instant où commence une situation.
- On a aussi besoin d'un cinquième axiome fondamental

$$début(do(a, s)) = temps(a). \qquad (5)$$

Calcul situationnel séquentiel temporel (2)

On ne fournit pas d'axiomes sur les réels (ou entiers, ou ...): on utilise l'interprétation standard (avec $+$, $\times$, ..., $\leq$, ...).

Il faut revoir la notion d'action exécutable, afin d'éliminer
$\boldsymbol{do(taper(B, M, 4), do(débutPleuvoir(M, 6), S_0))}$.

On modifie ainsi l'abréviation (EXEC) , remplacée par (EXECT):

$$\boldsymbol{executable(s)} =_{\boldsymbol{def}} \forall \boldsymbol{a, s'}\ [\boldsymbol{do(a, s')} \sqsubseteq \boldsymbol{s} \Rightarrow \boldsymbol{(Poss(a, s')\ \wedge\ début(s') \leq temps(a))}].$$

La contrainte $\boldsymbol{début(s') \leq temps(a)}$ évite les paradoxes temporels sans interdire (grâce à l'utilisation de l'inégalité large), une "succession" de deux actions qui se produisent en fait en même temps.

Calcul situationnel séquentiel temporel (3)

On peut ainsi représenter des données de ce genre:

On a une action $\boldsymbol{marcher(A, B)}$ qui débute en $\mathbf{1}$ et se termine en $\mathbf{4}$, une action $\boldsymbol{chanter}$ qui débute en $\mathbf{1}$, en même temps que $\boldsymbol{marcher(A, B)}$ (mais "avant") et se termine en $\mathbf{3}$ et une action $\boldsymbol{fumer}$ qui se termine en $\mathbf{4}$, en même temps que $\boldsymbol{marcher(A, B)}$ (et était en route en $\boldsymbol{S_0}$, et donc n'a pas de début).

Cela fait donc intervenir les actions (datées):
$\boldsymbol{débutMarcher(A, B, 1)}$, $\boldsymbol{finMarcher(A, B, 4)}$,
$\boldsymbol{débutChanter(1)}$, $\boldsymbol{finChanter(3)}$, $\boldsymbol{finFumer(4)}$;

et on sait (entre autres choses):

$\boldsymbol{marche(A, B, do([débutChanter(1), débutMarcher(A, B, 1)]), S_0)}$,
$\boldsymbol{chante(do(débutChanter(1), S_0))}$ et $\boldsymbol{fume(S_0)}$.

(Ici, on peut penser qu'on a $\boldsymbol{début(S_0) = 0}$, mais rien ne l'impose.)

Calcul situationnel: autres extensions

On peut aussi introduire des ajouts qui permettent de traiter le problème des actions exactement concurrentes (comme pour le duel).

Cela peut se faire en “temps qualitatif” (cf pages 44 , 45) et aussi

avec un temps explicite comme pour le calcul séquentiel temporel précédant (cf pages 46 – 48).

La dernière version permet de représenter les lois physiques naturelles (mouvements d’astres, ...).

Une autre extension permet de traiter du problème des actions extérieures et des actions de test (réactivité).

Ce cours ne donne qu’un aperçu. En particulier, une seule variante du calcul situationnel a été présentée ici, et de nombreuses autres variantes existent.