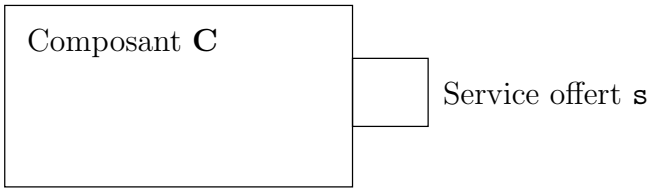
Sémantique formelle du plus simple composant Kmelia

Pascal Sotin

12 mars 2009



1 Description du système

Le composant **C** contient :
- Un ensemble V de variables identifiées par leur nom. Chacune a un domaine, donné par la fonction *dom* qui associe à un nom de variable (dans V) le domaine de la variable.
- Un état. L'état du composant appartient à Σ_V (espace d'état généré par un ensemble de variables typées : voir annexe A.1). On note abusivement cet état, propre au composant, $\Sigma_{\mathbf{C}}$.
- Le service `s`.
- Un prédicat invariant $inv_{\mathbf{C}}$ et une séquence d'initialisation, $\texttt{ini}_{\mathbf{C}}$. L'application de la substitution associée à cette séquence d'initialisation établit l'invariant : $[\texttt{ini}_{\mathbf{C}}]\,\Sigma_{\mathbf{C}} \subseteq inv_{\mathbf{C}} \subseteq \Sigma_{\mathbf{C}}$.

Le service `s` comporte :
- Un nom, un ensemble P de paramètres. On note $dom(p)$ le type du paramètre p (de P). On note $ret(\texttt{s})$ le type de la valeur de retour.
- Un ensemble L de variables locales. Typées elles aussi.
- Un LTS. Nous décrivons plus loin ce LTS, mais dont nous mentionnons déjà l'ensemble d'état Q et en particulier l'état initial q_0.
- On note $\Sigma_{\texttt{s}}$ l'espace d'état du service. On a $\Sigma_{\texttt{s}} = \Sigma_{\mathbf{C}} \times \Sigma_P \times \Sigma_L \times Q$.
- Un prédicat de précondition, $pre \subseteq \Sigma_{\mathbf{C}} \times \Sigma_P$.
- Une séquence d'initialisation, $\texttt{ini}_{\texttt{s}}$.
- Un prédicat de postconditions, $post \subseteq (\texttt{before} : \Sigma_{\mathbf{C}}) \times \Sigma_P \times \Sigma_{\mathbf{C}} \times ret(\texttt{s})$.

Le LTS du service `s` comporte :
- Un ensemble Q d'états d'automate.
- Un état $q_0 \in Q$ dit initial et un ensemble $Q_F \in Q$ d'état finaux.

– Un ensemble A d'actions.
– Une relation de transition, $\delta : Q \times A \to Q$.

Une action, mentionnée sur une transition du LTS peut être :
– Un appel de service. Mais il n'y a pas ici de services requis, donc personne à appeler.
– Un retour de service, noté `??s`(v) où v est une variable connue, globale, locale ou paramètre. On pourrait supposer que cette transition donne toujours sur un état final et bloquant, dans le cas d'un service non partagé.
– Une alteration de l'état des variables par une fonction f. On acceptera aussi tout langage de programmation dont on peut interpréter le code par une sémantique dénotationnelle. Exemple : `x:=5;`.
– Une lecture ou une écriture sur un canal. Ici, le seul canal existant est celui ouvert par l'appelant, par convention nommé Caller. La lecture est sous la forme Canal?message$(e_1, \ldots, e_n)$ ou les e_i sont des expressions. L'écriture est similaire, mais le `?` devient un `!` et les expressions sont remplacées par des variables.

2 Sémantique

On décrit la sémantique d'un appel de service. Comme il n'y a qu'un seul service, et qu'il n'est pas partagé, il ne peut être lancé simultanément qu'une seule fois par l'appellant.

Un état de la sémantique est de la forme :

$$\Sigma_V \times \mathit{option}\,(\Sigma_P \times \Sigma_L \times Q) \;\Big|\; \tau\,\langle \text{ErrMsg} \rangle$$

2.1 Démarrage du service

$$\text{call}\,\frac{\text{Démarrage du service } \mathtt{s}(p) \qquad (v,p) \in \mathit{pre} \qquad l \in \mathit{ini}_{\mathtt{s}}}{(v, \mathit{none}) \Rightarrow (v, \mathit{some}\,(p, l, q_0))} \qquad \text{bad call}\,\frac{\text{Démarrage du service } \mathtt{s}(p) \qquad (v,p) \notin \mathit{pre}}{(v, \mathit{none}) \Rightarrow \tau\,\langle \text{appel illégal} \rangle}$$

Le démarrage de service est légal, si l'état actuel du composant et les paramètres de l'appel satisfont la précondition. Dans le cas contraire, on obtient une erreur. Ce choix sémantique suppose qu'une vérification dynamique des préconditions est effectuée (quelles possibilités de garanties statiques ?). Le démarrage du service s'il est déjà en cours n'est pas autorisé. La sémantique est bloquante pour l'appelant sur ce point (vérification statique de non-blocage au démarrage d'un services ?).

2.2 Arrêt du service

$$\text{termination}\,\frac{q \in Q_F \qquad \text{post...}}{(v, \mathit{some}\,(p, l, q)) \Rightarrow (v, \mathit{none})}$$

$$\text{bad termination}\ \frac{q \in Q_F \qquad \neg\text{post...}}{(v, some\,(p,l,q)) \Rightarrow \tau\,\langle\text{Violation de post-conditions}\rangle}$$

Des questions émérgent :
- Rôle de l'instruction de retour `!!` *vs*.rôle des états finaux.
  - Risques de retour multiple.
  - Le retour débloque l'appellant. Il est succeptible de réappeler immédiatement, avant la fin du service.
- Portée des postconditions.
  - Une post-condition peut elle parler de la valeur de retour ? De la valeur des locales ?
- Synchronisation avec l'appelant. Le service termine-t-il si l'appelant n'est pas en attente de retour (`??`).

2.3 Modification d'état par programme

$$\text{code}\ \frac{q \overset{code}{\rightarrow} q' \qquad res\,\langle v', l'\rangle = [\![code]\!](v,l) \qquad v \in inv_{\mathbf{C}} \qquad (c,p,l,q) \in inv_{\mathbf{s}}}{(v, some\,(p,l,q)) \Rightarrow (v', some\,(p,l',q'))}$$

$$\text{echec code}\ \frac{q \overset{code}{\rightarrow} q' \qquad err\,\langle reason\rangle = [\![code]\!](v,l)}{(v, some\,(p,l,q)) \Rightarrow \tau\,\langle reason\rangle}$$

$$\text{bad code}\ \frac{q \overset{code}{\rightarrow} q' \qquad res\,\langle v', l'\rangle = [\![code]\!](v,l) \qquad (v \in inv_{\mathbf{C}} \vee (c,p,l,q) \in inv_{\mathbf{s}})}{(v, some\,(p,l,q)) \Rightarrow \tau\,\langle\text{Violation d'invariant}\rangle}$$

- Les appels de méthode sont par valeur.
  - Les paramètres sont-ils modifiables ?
- L'invariant porte-t-il sur l'état de l'automate ?

A Formalisme pour la sémantique

A.1 Prédicats et ensembles

Dans la présente sémantique, on manipule assez librement des prédicats pour des ensembles et vice-versa. Cette équivalence est bien connue et nous en rappelons ici les propriétés principales.

Soit S et P un ensemble et un prédicat équivalents. On a :

- $P(x) = x \in S$.
- $S = \{x \mid P(x)\}$.

On aura donc les équivalence suivantes :

Ensembliste	Logique
$A \subseteq B$	$A \Rightarrow B$
$\emptyset$	$faux$
$A \cup B$	$A \vee B$
$A \cap B$	$A \wedge B$

Dans notre sémantique, les ensembles considérés sont des sous-ensembles d'un espace d'état généré par un jeu X de variables typées. On note Π le produit cartésien généralisé et on définit l'espace d'état $\Sigma_X = \prod_{x \in X} dom(x)$. Dans notre sémantique, les prédicats considérés portent également sur un ensemble de variables. Pour un même ensemble de variables X on retrouve l'équivalence mentionnée plus haut entre des ensembles de tuples et des prédicats portant sur les même tuples.

Soit deux prédicats/ensembles A et B portant respectivement sur deux jeux de variables X et Y. On note que si $X \cap Y = \emptyset$, alors $A \wedge B = A \times B$, modulo flattening du tuple et ordonnancement des variables. On suppose sans perte de généralité que le produit cartésien utilisé gère cet aspect.