Vérification automatique de protocoles un "Model-Checker" pour CCS

Eric Goubault

Eric.Goubault@cea.fr

1 Préambule

Le but de ce projet est d'implémenter un outil de vérification de petits protocoles de communication. Par exemple des protocoles comme le bit alterné, assurant un bon transport des messages sur des lignes de communication qui peuvent éventuellement perdre des messages, pourra être vérifié par ce petit outil. Ces protocoles seront écrits dans le langage ("algèbre de processus") CCS ("Calculus of Communicating Systems"[Mil89]). Le principe est de vérifier la validité de formules logiques écrites dans la logique temporelle CTL[CES86] sur les programmes CCS. Ces formules spécifient des relations de causalité entre actions (comme l'envoi ou la réception de messages), que l'on désire voir vérifier. Comme expliqué dans la section 7 on pourra se contenter d'implémenter seulement une sous-partie du langage et de l'algorithme. L'explication plus complète ci-dessous n'est destinée qu'à la compréhension du problème en général (et aux plus courageux).

2 Syntaxe de CCS

Le langage CCS décrit les actions d'envoi et de réception de messages entre des processus séquentiels[Mil89]. Il a la syntaxe suivante, étant donné un ensemble Σ de noms de canaux de communication:

$$
\begin{array}{rcll}
t & = & nil & \text{(1) fin de processus} \\
 & | & \overline{a} \in \Sigma & \text{(2) envoi sur } a \\
 & | & a \in \Sigma & \text{(3) réception sur } a \\
 & | & t + t & \text{(4) choix non-déterministe} \\
 & | & t.t & \text{(5) composition séquentielle} \\
 & | & t \mid t & \text{(6) composition parallèle} \\
 & | & t \backslash c & \text{(7) cache le canal } c \\
 & | & t[f] & \text{(8) renommage par fonction f} \\
 & | & \text{rec } x.t[x] & \text{(9) agent récursif} \\
 & | & (t) & \text{(10) parenthésage}
\end{array}
$$

Des exemples de programmes CCS seront disponibles à la demande, en me mailant, ou dans le livre [Mil89], et également sur le site web dédié à ce projet:

`http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/Cours04/projet04.html`

Intuitivement, CCS décrit l'envoi et la réception de messages synchrones sur des canaux de communication. Par exemple $(a.\overline{b}) \mid (b.\overline{a})$ est un programme composé de deux processus, l'un qui envoie un message sur le canal de communication a puis qui attend de recevoir un message sur le canal de communication b, l'autre qui envoie un message sur le canal b avant d'attendre un message sur a. Tel que la sémantique (ou signification des programmes) de CCS sera définie dans la section 4, comme les envois et les réceptions sont synchrones (c.a.d. bloquants), ce programme est bloqué avant de pouvoir exécuter toutes ses instructions. Cela fait partie des propriétés que l'on pourra décrire par la logique CTL (voir section suivante) et que l'on pourra vérifier en utilisant l'algorithme de la section 6. Un autre exemple simple est celui d'un "tampon à une place": c'est un processus qui peut prendre une donnée sur un canal a et attendre qu'on la lui demande sur le canal de sortie b, ceci etant exécuté à l'infini. On se convaincra sans peine que le terme CCS correspondant est $recX.(\overline{a}.b.X)$.

3 Syntaxe de CTL

Elle est définie par (on verra ce que signifie le τ dans la section suivante):

F	$=$	p_a	(I) prédicat de base pour $a \in \Sigma$: "envoi sur a est une exécution possible"
	$\mid$	$p_{\overline{a}}$	(II) prédicat de base pour $a \in \Sigma$: "réception sur a est une exécution possible"
	$\mid$	p_τ	(III) prédicat de base: "une synchronisation τ est une exécution possible"
	$\mid$	$\neg F$	(IV) négation
	$\mid$	$F \wedge F$	(V) et
	$\mid$	$F + F$	(VI) ou
	$\mid$	$F => F$	(VII) implique
	$\mid$	(F)	(VIII) parenthésage
	$\mid$	AXF	(IX) Toujours F
	$\mid$	EXF	(X) Quelquefois F
	$\mid$	$A[FUF]$	(XI) Toujours...jusqu'à ce que...
	$\mid$	$E[FUF]$	(XII) Quelquefois...jusqu'à ce que...

Des exemples de formules seront disponibles en me mailant, ou sur la page web du projet.

La signification (ou sémantique) de cette logique sera donnée en section 5. Intuitivement, on cherche à exprimer des propriétés de protocoles écrits en CCS telles que "le programme CCS est tel que dès qu'il peut recevoir une donnée sur le canal a il peut renvoyer immédiatement cette valeur sur b". Cela peut se traduire dans CTL par $p_{\overline{a}} => EX(p_b)$. Cette propriété est vraie pour le tampon à une place (et même la propriété plus forte $p_{\overline{a}} => AX(p_b)$).

4 Sémantique de CCS

On représente un programme CCS par un graphe décrivant les actions (envoi ou réception sur un canal, synchronisation τ) que peut effectuer ce programme. Un chemin dans ce graphe sera une suite d'action(s) que ce programme peut effectuer. Par exemple le tampon à une place de la section 2 pourra être représenté par un graphe cyclique n'ayant comme chemins que des chemins du type $\overline{a}.b.\overline{a}.b.\cdots$. Le programme $a \mid b$ contiendra comme chemins tous les sous-chemins de $a.b$ et de $b.a$. On introduit l'action spéciale τ pour tenir compte des actions envoi et réception sur un même canal qui se synchronisent. Le programme $(a \mid \overline{a})\backslash a$ n'a pour seul chemin autorisé que l'action synchronisation τ.

Formellement, le graphe en question s'appelle un système de transitions. C'est une structure $(S, T, \rightarrow)$ où,

- S est un ensemble d'états,
- T est un ensemble de transitions,
- $\rightarrow \subseteq S \times T \times S$ est la relation de transition.

On peut calculer les systèmes de transitions inductivement sur la syntaxe des programmes.

<u>Format:</u> Pour un opérateur $\mathcal{O}$ de CCS ($+$, $.$, $|$ etc.), et P et Q deux programmes CCS alors,

$$\frac{P : s \xrightarrow{a} t \qquad Q : s' \xrightarrow{b} t'}{P\mathcal{O}Q : u \xrightarrow{c} v}$$

se lit:

"Si P a une transition (s, a, t) et Q a une transition (s', b, t') alors $P\mathcal{O}Q$ a une transition (u, c, v).

$$\frac{}{\alpha.t \xrightarrow{\alpha} t} \tag{1}$$

$$\frac{t_i \xrightarrow{\alpha} t'_i}{t_1 + t_2 \xrightarrow{\alpha} t'_i} \tag{2}$$

$$\frac{t_1 \xrightarrow{\alpha} t'_1}{t_1 \mid t_2 \xrightarrow{\alpha} t'_1 \mid t_2} \tag{3}$$

$$\frac{t_2 \xrightarrow{\alpha} t'_2}{t_1 \mid t_2 \xrightarrow{\alpha} t_1 \mid t'_2} \tag{4}$$

$$\frac{t_1 \xrightarrow{\alpha} t'_1 \qquad t_2 \xrightarrow{\overline{\alpha}} t'_2}{t_1 \mid t_2 \xrightarrow{\tau} t'_1 \mid t'_2} \tag{5}$$

$$\frac{t \xrightarrow{\alpha} t' \qquad \alpha \neq c \text{ et } \alpha \neq \overline{c}}{t \backslash c \xrightarrow{\alpha} t' \backslash c} \tag{6}$$

$$\frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f(\alpha)} t'[f]} \tag{7}$$

$$\frac{\text{rec } x.t[x] \xrightarrow{\alpha} t'}{t[\text{rec } x.t[x]] \xrightarrow{\alpha} t'} \tag{8}$$

Les transitions seront représentées par une liste de pointeurs vers les états suivants, et aussi par une liste de pointeurs vers les états précédents (cela n'est nécessaire que si l'on considère l'opérateur $E[.U.]$).

5 Sémantique de CTL

Formellement, la signification d'une formule CTL peut être écrite comme suit. Etant donné un état s d'un système de transitions engendré par un programme CCS (donc s est un "sous-programme" CCS), et une formule de CTL f, f est vraie en s (noté $s \models f$) dans les cas suivants:

$s \models p_a$ ssi il existe une transition de nom a ayant pour source s (c.a.d. il existe un état t tel que $(s, a, t) \in \rightarrow$). On imaginera sans peine les cas similaires pour $p_{\overline{a}}$ et p_τ.

$s \models \neq f$ ssi $s \models f$ est faux.

$s \models f_1 \wedge f_2$ ssi $s \models f_1$ et $s \models f_2$.

$s \models AXf_1$ ssi pour tous les états t tels qu'il existe une transition $(s, a, t) \in \rightarrow$, $t \models f_1$.

$s \models EXf_1$ ssi il existe un état t tel qu'il y ait une transition $(s, a, t) \in \rightarrow$ et $t \models f_1$.

$s \models A[f_1 U f_2]$ ssi pour tous les chemins faits d'états successifs $(s_0, s_1, \cdots)$, il existe un indice $i \geq 0$ tel que $s_i \models f_2$ et pour tout j, $0 \leq j \leq i-1$, $s_j \models f_1$.

$s \models E[f_1 U f_2]$ ssi il existe un chemin fait d'états successifs $(s_0, s_1, \cdots)$ tel qu'il existe un indice $i \geq 0$ tel que $s_i \models f_2$ et pour tout j, $0 \leq j \leq i-1$, $s_j \models f_1$.

Il y a des abbréviations classiques de termes CTL, dont le sens est sans doute plus immédiat que les opérateurs $A[.U.]$ et $E[.U.]$. L'un est $AF(f) = A[vraiUf]$: "f est inévitable" (toujours vrai sur toutes les exécutions possibles). L'autre est $EF(f) = E[vraiUf]$: "f est éventuellement vrai" (il existe une exécution le long de laquelle f est vraie).

6 Algorithme de vérification

Une formule CTL f est vraie pour un programme CCS P si $P \models f$ (P est l'état initial du système de transition). Maintenant le principe de vérification est d'associer à chaque état du système de transition engendré par P l'ensemble des sous-formules de f vraies en cet état. Il suffit ensuite de vérifier que f est dans l'ensemble en l'état P. L'algorithme lui-même fonctionne en commençant par considérer les sous-formules de longueur 1, puis de longueur 2 etc. jusqu'à terminer à l'étape longueur(f).

La méthode de vérification est basée sur un parcours en profondeur du graphe de transition. Appelons $v(s)$ l'ensemble des sous-formules couramment trouvées vraies en un état s. Au début de cet algorithme $v(s)$ est vide pour tous les états. Si l'on a examiné le cas des formules de longueur i ($i \geq 1$), alors il est facile (voir les règles de la section 5) d'en déduire les formules vraies de longueur $i+1$, au moins pour les cas où on n'a pas les opérateurs $A[.U.]$ ni $E[.U.]$. C'est ces derniers cas que l'on décrit un peu plus en détail (voir [CES86] pour encore plus de détails).

Etant donné un opérateur à deux arguments de CTL f, appelons $arg(f)$ son premier argument (resp. $arg'(f)$ sont deuxième argument). $in - v(s, f)$ est vrai ssi la formule f est dans $v(s)$. $+v(s, f)$ est l'opération d'ajouter la formule f à l'ensemble $v(s)$ de l'état courant s. Un booléen $b(s)$ est également associé à chaque état s du système de transition. L'algorithme d'étiquetage du graphe de transitions par les ensembles $v(s)$ est la procédure décrite ci-dessous,

```
procedure etiquette(f)
begin
```

```
   ...
{ l'op\'erateur de tete de f est A[.U.] }
   begin
      Pour tous les s etat, b(s)=faux
      Pour tous les s etat, si non(b(s)) alors au(f,s,b)
   end
   ...
end
```

`au(f,s,b)` examine le cas de la formule f en commencant par l'état s. b sera vrai si $s \models f$. La procédure `au` est,

```
procedure au(f,s,b)
begin
   Si b(s) alors,
      Si in-v(s,f) alors b=vrai, return
      Sinon, b=faux, return
   Sinon,
      b(s)=vrai
      Si in-v(s,arg'(f)) alors +v(s,f), b=vrai, return
      Sinon,
         Si non(in-v(s,arg(f))) alors b=faux, return
         Sinon,
            Pour tous les s' successeurs de s (par une transition),
               au(f,s',b')
               Si non(b') alors b=faux, return
            +v(s,f), b=vrai, return
end
```

Pour des formules de la forme $f = E[f_1 U f_2]$, on trouve d'abord tous les états qui ont dans leur étiquette f_2 puis on travaille en revenant par l'inverse de la relation de transition pour trouver tous les états qui peuvent être atteints par un chemin dans lequel chacun des états a dans son étiquette f_1. On doit ajouter alors aux étiquettes de ces états la formule f.

7 Réalisation du projet

On devra d'abord écrire un programme pour parser le langage CCS (donc fournir un arbre syntaxique), et calculer le système de transition définissant sa sémantique. Telle qu'est donnée la grammaire de CCS, il n'y a aucune difficulté majeure: on pourra par exemple résoudre les problèmes de priorité des opérateurs en supposant que les programmes sont écrits avec un nombre suffisant de parenthèses.

Dans le même ordre d'idée on écrira un parser pour les formules CTL. Ensuite on implémentera l'algorithme de Clarke, Emerson et Sistla (pour plus de détails, on pourra se référer à [CES86], disponible sur la page web du projet).

On pourra se contenter de n'implémenter que le sous langage de CCS composé des termes (1) à (6) plus (10) (voir section 2). On pourra aussi se contenter de n'implémenter que les termes de CTL définis par les équations (I) à (X) (voir section 3).

Il y a plusieurs façons de découper le travail pour un binôme. Mis à part la définition de structures de données communes pour les arbres syntaxiques et les systèmes de transition, les algorithmes de parsing de CCS, de CTL, la sémantique de CCS (c'est à dire la génération du système de transitions correspondant à un programme CCS) et l'algorithme de vérification lui-même sont indépendants.

Je tiens à la disposition des élèves les documents inscrits dans la bibliographie sommaire ci-jointe (plus d'autres pour les mordus).

References

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages*, 8(2):244–263, 1986.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.