Introduction au model-checking

Cours de langages de spécification

Catalin Dima

Un art de la programmation ?

- Programmer = transformer une spécification en programme.
  - Spécification = demande du client/prof.
  - Implémentation = tranformation de la spécification en programme/module/procédure/paquetage.
- Mythes entourant la programmation :
  - La programmation est un art.
  - Les programmeurs sont des hackers/gourous/illuminés.
- Faux ! Méthodologie de la programmation :
  - Algorithmique, génie logiciel, etc.

Problème

Si la programmation est une science, comment prouve-t-on que ce qu'on produit est correct ?...

Méthodologies du développement de programmes

- Approches informelles :
  - Gourous/hackers/illuminés...
- Approches semi-formelles :
  - Méthodologie orientée objet, UML, Entité-relation.
  - Développement en V, en spirale, etc.
  - Développement agile.
- Mais comment fait-on pour assurer le client de la bonne implémentation de sa spéc ?...
- Mais comment le prof fait-il pour corriger la copie ?...
  - *Confiance, test, preuve*.
- Approches formelles :
  - Simulation/test.
  - Raffinement.
  - Vérification de type model-checking.
  - Preuve mathématique/logique.

Méthodologies formelles de développement de programmes

- Test :
  - Critères de couverture.
  - Méthodologie de spécification des cas de test.
  - Approche probabiliste : hypothèses statistiques, intervalles de confiance, etc.
- Simulation :
  - Modélisation (construction d'un modèle à échelle réduite) du logiciel.
    - Similaire aux approches de modélisation dans l'industrie aéronautique/nucléaire etc.
  - Ou modélisation de l'environnement dans lequel le logiciel évolue.
  - Suivies par des batteries de test.

Méthodologies formelles de développement de programmes

- Raffinement.
  - Construction incrémentale d'une implémentation à partir d'une spécification formelle.
    - On part d'une spécification formelle, mais de haut niveau (formules logiques, pas d'instructions).
    - À chaque pas, on raffine une partie de la spécification précédente...
    - ... jusqu'à arriver à du code implémentable.
  - Exemple : méthode B.
- Vérification par model-checking.
  - Modélisation (construction d'un modèle à échelle réduite) du logiciel.
    - Abstraction, peut se faire de manière automatique.
  - Prise en compte de la spécification (formule logique) que le logiciel est censé implémenter.
  - Vérification, à l'aide d'algorithmes (développés dans ce cours !) que le modèle vérifie la formule.
- Preuve.
  - Le comportement du logiciel est décrit à l'aide des formules, selon la sémantique du langage de spécification (v. cours de sémantique).
  - À l'aide d'outils assistants de preuve, montrer que ces formules impliquent les formules de la spécification.

Avantages et désavantages des méthodologies formelles (3)

- Test : rapidité, mais gros pb. de couverture.
- Simulation : bon compromis de rapidité par rapport au test, mais toujours pb. de couverture et nécessité d'une modélisation.
- Raffinement : sûreté garantie, mais limites d'application et méthodologie nécessitant une formation poussée.
- Preuve : sûreté garantie, mais formation très poussée et difficulté algorithmique.
- Model-checking : bon compromis quant à la garantie de la sûreté et utilisable en tant que "bug chasing", mais formation nécessaire et certaines pb. de modélisation et de difficulté algorithmiques (moins que l'approche preuve).

Pourquoi model-checking en M1 Info, UPEC ?

- Applications hautement sensibles : solutions de sécurité propriétaires.
  - M2 SSI !
- Validation des implémentations de protocoles de communication propriétaires (client-serveur à clients multiples, communication multipartite).
- Implémentations d'ordonnanceurs de type particulier.
- Toute nouvelle implémentation de solutions classiques, lorsqu'une validation très stricte est à produire.

Rappels de théorie des automates

- Automates = modèles de type finitaire du logiciel.
- $\mathcal{A} = (Q, \Sigma, \delta, Q_0, Q_f)$:
  - Alphabet = ensemble de valeurs de variables/registres du système informatique sur lequel le logiciel tourne.
  - État de l'automate = abstraction de l'état du système informatique.
  - Transition dans l'automate = abstraction d'un pas d'exécution dans le logiciel.
- Automate de Mealy : $\delta \subseteq Q \times \Sigma \times Q$.
- Automate de Moore : $\delta \subseteq Q \times Q$, mais étiquetage supplémentaire des états :
  $\mathcal{A}_{Moore} = (Q, \Sigma, \delta, \lambda, Q_0, Q_f)$ avec $\lambda : Q \longrightarrow \Sigma$.
- Trajectoire : suite de transitions partant d'un état initial $\in Q_0$ et s'arrêtant dans un état final $\in Q_f$.
- Exemple...

Construction d'un automate de Moore à partir d'un automate de Mealy

- $\mathcal{A} = (Q, \Sigma, \delta, Q_0, Q_f)$ automate de Mealy.
- Comment construire (**de manière générique !**) aut. Moore $\mathcal{A}' = (Q', \Sigma, \delta', \lambda, Q'_0, Q'_f)$ t.q. $L(\mathcal{A}) = L(\mathcal{A}')$?
- L'idée : tirer les étiquetes des transitions vers les états source.
- **Formalisation** :

$$
\begin{aligned}
Q' &= \delta \\
\lambda(q \xrightarrow{a} r) &= r \\
\delta' &= \left\{(q \xrightarrow{a} r, r \xrightarrow{b} s) \mid q \xrightarrow{a} r, r \xrightarrow{b} s \in \delta\right\} \\
Q'_0 &= \left\{q \xrightarrow{a} r \mid q \in Q_0\right\} \\
Q'_f &= \left\{q \xrightarrow{a} r \mid r \in Q_f\right\}
\end{aligned}
$$

- $L(\mathcal{A}') = L(\mathcal{A}) \setminus \{\varepsilon\}$.
- Le mot vide est perdu...

Construction d'un automate de Mealy à partir d'un automate de Moore

- $\mathcal{B} = (Q, \Sigma, \delta, \lambda, Q_0, Q_f)$ automate de Mealy.
- Comment construire $\mathcal{B} = (Q', \Sigma, \delta', Q'_0, Q'_f)$ automate de Moore avec $L(\mathcal{B}) = L(\mathcal{B}')$.
- Idée : déplacer les étiquettes des états vers toutes les transitions sortantes.
- Et bien choisir les états finaux.
- Formalisation :

$$Q'_f = \{r' \mid r \in Q_f\} \qquad Q'_f \text{ copie disjointe de } Q_f$$
$$Q' = Q \cup Q'_f$$
$$\delta' = \{q \xrightarrow{a} r \mid q \in Q, r \in Q', (q, r) \in \delta, \lambda(q) = a\}$$
$$\cup \{q \xrightarrow{a} q' \mid q \in Q_f, q' \in Q_f \text{ étant la copie de } q, \lambda(q) = a\}$$
$$Q'_0 = Q_0$$

- Donc, attention, on n'a pas de transition sortante de Q'_f !

Constructions : union

- On nous donne deux automates : $\mathcal{A}_1 = (Q_1, \Sigma_1, \delta_1, \lambda_1, Q_0^1, Q_f^1)$ et $\mathcal{A}_2 = (Q_2, \Sigma_2, \delta_2, \lambda_2, Q_0^2, Q_f^2)$.
- Et on voudrait savoir si $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ peut être aussi reconnu par un automate fini.
- Idée : mettre les deux automates côte-à-côte !
- Exemple ...
- Formalisation : $\mathcal{A} = (Q, \Sigma, \delta, \lambda, Q_0, Q_f)$ avec

$$Q = Q_1 \cup Q_2$$

Mais on suppose que $Q_1 \cap Q_2 = \emptyset$!

$$\Sigma = \Sigma_1 \cup \Sigma_2$$
$$Q_0 = Q_0^1 \cup Q_0^2$$
$$Q_f = Q_f^1 \cup Q_f^2$$
$$\delta = \delta_1 \cup \delta_2$$
$$\lambda(q) = \begin{cases} \lambda_1(q) & \text{si } q \in Q_1 \\ \lambda_2(q) & \text{si } q \in Q_2 \end{cases}$$

Concaténation

- Même situation, avec deux automates donnés : $\mathcal{A}_1 = (Q_i, \Sigma_i, \delta_i, \lambda_i, Q_0^i, Q_f^i)$, $i = 1, 2$.
- On veut cette fois-ci un automate pour $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$.
- Idée : relier par ε-transitions chaque état final de $\mathcal{A}_1$ à chaque état initial de $\mathcal{A}_2$.
- Exemple ...
- Formalisation : $\mathcal{A} = (Q, \Sigma, \delta, \lambda, Q_0, Q_f)$ avec

$$
\begin{aligned}
Q &= Q_1 \cup Q_2 \\
\Sigma &= \Sigma_1 \cup \Sigma_2 \\
Q_0 &= Q_0^1 \\
Q_f &= Q_f^2 \\
\delta &= \delta_1 \cup \delta_2 \cup \left\{ q_1 \longrightarrow q_2 \mid q_1 \in Q_f^1, q_2 \in Q_0^2 \right\} \\
\lambda(q) &= \begin{cases} \lambda_1(q) & \text{si } q \in Q_1 \\ \lambda_2(q) & \text{si } q \in Q_2 \end{cases}
\end{aligned}
$$

Étoile

- Cette fois-ci on nous donne un seul automate : $\mathcal{A} = (Q, \Sigma, \delta, Q_0, Q_f)$.
- Et on veut construire l'automate pour $L(\mathcal{A})^+$.
  - Il faut se rappeler que $\varepsilon \notin L(\mathcal{A})$ pour n'importe quel automate de Moore $\mathcal{A}$!
- De tout état final on doit revenir dans un état initial pour recommencer !
- Exemple ...
- Formalisation : $\mathcal{A}' = (Q', \Sigma, \delta', \lambda', Q_0', Q_f')$ avec

$$Q' = Q, \qquad Q_0' = Q_0, \qquad Q_f' = Q_f$$
$$\delta' = \delta \cup \{(q_1, q_2) \mid q_1 \in Q_f, q_2 \in Q_0$$
$$\lambda'(q) = \lambda(q)$$

Intersection

- Maintenant on veut un automate pour $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.
  - Attention ! Il faut que les deux automates **partagent le même alphabet** !
- Idée : simuler synchronémment $\mathcal{A}_1$ et $\mathcal{A}_2$ sur le même automate :
  - L'état de $\mathcal{A}$ serait une paire d'états, un état de $\mathcal{A}_1$ et un autre de $\mathcal{A}_2$.
  - Etats initiaux : paire d'états initiaux dans chaque automate.
  - Pareil pour les états finaux.
- Exemple :

$$L_5 = \{w \in \{a,b\}^* \mid \#_a(w) \text{ est un multiple de 3 et } \#_b(w) \text{ est un multiple de 4}\}$$

- Formalisation : $\mathcal{A} = (Q, \Sigma, \delta, \lambda, Q_0, Q_f)$ avec

$$Q = \{(q_1, q_2) \in Q_1 \times Q_2 \mid \lambda_1(q_1) = \lambda_2(q_2)\}$$
$$Q_0 = Q \cap Q_0^1 \times Q_0^2$$
$$Q_f = Q \cap Q_f^1 \times Q_f^2$$
$$\delta = \{(q_1, q_2) \longrightarrow (r_1, r_2) \mid q_1 \longrightarrow r_1 \in \delta_1, q_2 \longrightarrow r_2 \in \delta_2\}$$
$$\lambda(q_1, q_2) = \lambda_1(q_1) = \lambda_2(q_2)$$

Shuffle

- Et enfin, on veut construire un automate pour le shuffle $L(\mathcal{A}_1) \sqcup\!\sqcup L(\mathcal{A}_2)$.
- On doit pouvoir faire des transitions soit dans une, soit dans l'autre des automates.
- Il faut donc avoir en "mémoire" l'état de chaque automate.
  - Un peu comme pour l'intersection.
- Exemple ...
- Formalisation : $\mathcal{A} = (Q, \Sigma, \delta, \lambda, Q_0, Q_f)$ avec

$$Q = Q_1 \times Q_2 \times \{1,2\}$$
$$Q_0 = Q_0^1 \times Q_0^2 \times \{1,2\}$$
$$Q_f = Q_f^1 \times Q_f^2 \times \{1,2\}$$
$$\delta = \{(q_1, q_2, i) \longrightarrow (r_1, q_2, j) \mid q_1 \longrightarrow r_1 \in \delta_1, q_2 \in Q_2, i, j \in \{1,2\}\}$$
$$\cup \{q_1, q_2, 2) \longrightarrow (q_1, r_2, 2) \mid q_2 \longrightarrow r_2 \in \delta_2, q_1 \in Q_1, i, j \in \{1,2\}\}$$
$$\lambda(q,r,1) = \lambda_1(q), \qquad \lambda(q,r,2) = \lambda_2(r)$$