École d'Été Temps-Réel
Rennes
Août 2015

Sujet de travaux pratiques avec la boîte à outils CADP

1 Ressources sur CADP

Diverses ressources concernant CADP sont disponibles à l'URL `http://cadp.inria.fr`, telles que :

- la liste des outils ainsi que leurs manuels d'utilisation,
- les informations utiles pour obtenir une copie de CADP,
- une FAQ (foire aux questions),
- des publications scientifiques et des tutoriels,
- des exemples de démonstration,
- une liste d'études de cas et outils dérivés réalisés à travers le monde avec CADP,
- un lien vers le forum des utilisateurs de CADP.

La principale référence bibliographique concernant CADP est [3]. Les techniques de vérification compositionnelle sont décrites dans [2].

2 Mise en route

Pour ce TP, CADP a été installé sur le machines de cette salle (merci à Damien Hardy pour son aide). Il est néanmoins nécessaire de positionner quelques variables d'environnement en utilisant les commandes suivantes (sous bash) :

```
export CADP=/import/share/2015-2016/ETR2015/CADP
export PATH="$CADP/com:$CADP/bin.‘$CADP/com/arch‘:$PATH"
```

Ouvrir un navigateur Web et récupérer les sources du TP depuis le site d'ETR (`http://etr2015.irisa.fr`) dans la rubrique "Travaux pratiques". Décompresser les sources à l'aide des commandes suivantes :

```
gunzip tp-cadp.tar.gz
tar xvf tp-cadp.tar
```

Aller dans le répertoire `tp-cadp`.

3 L'exemple du TFTP

3.1 Description du système

On considère un protocole de communication sol-avion étudié par Airbus et basé sur TFTP/UDP (*Trivial File Transfer Protocol/User Datagram Protocol*). Ce protocole est modélisé formellement en LNT et des formules de logique temporelle sont spécifiées dans le langage MCL. [1]

Le système modélisé représente deux utilisateurs (par exemple le sol et l'avion) s'échangeant des fichiers via le protocole. Pour les besoins du protocole, chaque fichier à échanger est divisé en fragments. Dans l'instance que nous considérons, un seul utilisateur envoie à l'autre utilisateur un fichier constitué d'un unique fragment. Ces entités communiquent par le biais d'un medium de communication bidirectionnel de type FIFO avec pertes aléatoires. Dans le cadre de ce TP, il n'est pas nécessaire de comprendre plus précisément le fonctionnement du protocole.

Le système est décrit sous la forme de plusieurs processus dont des instances s'exécutent en parallèle :

- USER_PROCESS décrit le comportement générique (non-déterministe) d'un utilisateur intéragissant avec le protocole.
- TFTP_PROCESS décrit le comportement générique du protocole. Ce processus intéragit avec l'utilisateur d'une part et les mediums de communication d'autre part.
- MEDIUM_PROCESS décrit le comportement générique d'un sens de communication du medium.

On considère deux processus nommés TFTP_A et TFTP_B, définis chacun par composition parallèle d'une instance de USER_PROCESS et d'une instance de TFTP_PROCESS, et deux processus nommés MEDIUM_A et MEDIUM_B définis comme des instances de MEDIUM_PROCESS.

Les différents processus intéragissent par rendez-vous (avec échange de messages) sur des portes nommées comme suit :

- PUT_A (resp. PUT_B) pour l'envoi de fichiers de USER_PROCES à TFTP_PROCESS au sein de TFTP_A (resp. TFTP_B),
- GET_A (resp. GET_B) pour la réception par USER_PROCESS des fichiers émis par TFTP_PROCESS au sein de TFTP_A (resp. TFTP_B),
- SEND_A (resp. SEND_B) pour l'envoi des messages de TFTP_A (resp. TFTP_B) vers MEDIUM_A (resp. MEDIUM_B),
- et enfin RECEIVE_A (resp. RECEIVE_B) pour la réception par TFTP_B (resp. TFTP_A) des messages émis par MEDIUM_B (resp. MEDIUM_A).

De plus, deux portes EVENT_A et EVENT_B, non-synchronisées, sont utilisées par les différentes instances de processus pour signaler l'occurrence de certains événements.

Pour résumer, le système a donc l'architecture décrite par la figure 1, où les boîtes représentent les processus et les liens représentent les portes sur lesquels ont lieu les rendez-vous.

Le code LNT du système est réparti dans plusieurs fichiers :

[1] Cet exemple a été traité dans le cadre des projets collaboratifs OpenEmbedd et Topcased impliquant des industriels et des laboratoires de recherche publics. Il est décrit en détails dans la thèse de Damien Thivolle [7] et plus brièvement dans un article scientifique publié dans une conférence internationale avec comité de lecture [4]. Il est également mentionné et utilisé comme benchmark dans un article de journal international [2] sur la vérification compositionnelle.

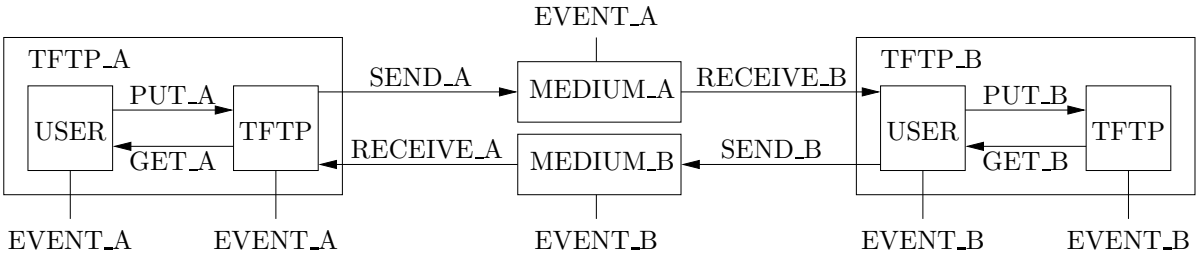

Figure 1: Architecture du système

- `LOCAL_CONSTANTS.lnt` (29 lignes) : définitions de constantes, telles que la taille du medium de communication ou le nombre maximal de tentatives de réémission de messages pour chacune des entités.
- `FILES.lnt` (194 lignes) : définitions de types et de fonctions pour la représentation et la manipulation des fichiers (listes de caractères) et des listes de fichiers.
- `MESSAGES.lnt` (249 lignes) : définitions de types et de fonctions pour la représentation et la manipulation des événements et des messages du protocole.
- `QUEUE.lnt` (138 lignes) : définitions de types et de fonctions pour la représentation et la manipulation des files de messages.
- `REPOSITORY.lnt` (100 lignes) : diverses fonctions auxiliaires.
- `AUTOMATON.lnt` (405 lignes) : définition de l'automate décrivant le protocole TFTP.
- `MEDIUM.lnt` (59 lignes) : squelette de MEDIUM_PROCESS.
- `USER.lnt` (93 lignes) : définition de USER_PROCESS.
- `TFTP.lnt` (452 lignes) : définition de TFTP_PROCESS.
- `SCENARIO_A.lnt` (67 lignes) : description du système présenté dans la figure 1.

3.2 Définition de MEDIUM_PROCESS

Le processus MEDIUM_PROCESS a le comportement non-déterministe générique suivant :

- Il peut recevoir un message M valide sur la porte RECEIVE, puis :
  - soit ajouter ce message à la file d'attente Q si la taille de la file est inférieure au nombre maximal d'éléments qu'elle peut contenir ;
  - soit perdre ce message, auquel cas un message LOSS est émis sur la porte EVENT ; si la file est pleine, c'est la seule option.
- Alternativement, il peut envoyer le premier message de la file d'attente Q sur la porte SEND si la file n'est pas vide, puis supprimer ce message de la file.

Compléter la définition de MEDIUM_PROCESS dans le fichier `MEDIUM.lnt`. Pour la syntaxe du langage, on prendra exemple sur les autres processus dont le code est donné. On utilisera l'opération IS_VALID sur les messages, définie dans le fichier `MESSAGES.lnt`, et les opérations SIZE, MAX_QUEUE_ELEMENTS, ENQUEUE, NTH et REMOVE sur les files d'attente, définies dans le fichier `QUEUE.lnt`.

3.3 Simulation interactive de MEDIUM_PROCESS

Pour vérifier un programme LNT, une première solution est de le simuler de manière interactive. Pour cela :

- lancer la commande xeuca en tâche de fond, ce qui ouvre l'interface graphique de CADP (Eucalyptus)
- Cliquer sur le fichier MEDIUM.lnt et choisir "Execute/Advanced Simulation".
- Une fenêtre s'ouvre demandant de choisir un processus parmi ceux définis dans le module, cliquer sur MEDIUM_PROCESS puis cliquer sur OK.
- Le fichier est alors compilé : corriger vos erreurs et recommencer jusqu'à ce que la compilation réussisse.
- Lorsque le code est syntaxiquement correct, le simulateur OCIS s'ouvre. On peut alors explorer de manière interactive un arbre d'exécution du processus en sélectionnant des actions parmi celles qui sont possibles dans l'état courant, proposées dans le cadran du bas. Vérifier que le medium semble bien fonctionner en mode FIFO avec pertes.

3.4 Comparaison de STE

Le STE (système de transitions étiquetées) attendu pour MEDIUM_PROCESS est donné dans le fichier `MEDIUM_PROCESS.bcg`. Pour s'assurer de la correction de votre code, il est possible de comparer le processus proposé à la question précédente avec ce STE.

Pour cela, cliquer sur le fichier `MEDIUM.lnt` puis choisir "Compare...". Sélectionner MEDIUM_PROCESS, puis conserver les paramètres par défaut ("Compare on the fly using Bisimulator", "Strong equivalence", "Equivalence (=)", "Breadth-first search" et garder le fichier `diagnostic.bcg` comme nom de fichier de diagnostic, qui pourra vous servir à identifier les erreurs éventuelles).

N.B. : Ne restez pas bloqués... Si vous ne parvenez pas à écrire ce processus correctement, la solution est disponible dans le fichier `Corige/MEDIUM.lnt`.

3.5 Génération du STE du système complet, première tentative

Pour générer le STE du système à partir d'Eucalyptus, cliquer sur le fichier `SCENARIO_A.lnt`, puis choisir "Generate Labelled Transition System...". Sélectionner le processus principal (MAIN), garder les options par défaut (BCG et "Real-time monitoring") puis cliquer sur OK.

Qu'observez-vous ?

Il est recommandé d'interrompre la génération avant de saturer la machine...

3.6 Génération du STE des composants du système

De la même manière que précédemment, générer le système de transitions étiquetées des processus TFTP_A, TFTP_B, MEDIUM_A et MEDIUM_B. A chaque fois, cela produit un fichier nommé `SCENARIO_A.bcg`, qu'il faut donc renommer (File/Move dans Eucalyptus) afin d'obtenir finalement quatre fichiers que l'on nommera `TFTP_A.bcg`, `TFTP_B.bcg`, `MEDIUM_A.bcg` et `MEDIUM_B.bcg`.

Noter les tailles de chacun de ces STE. Cette taille est affichée dans la fenêtre de monitoring à la fin de la génération. Elle peut également être obtenue (ainsi que d'autres informations) en cliquant sur le fichier BCG, puis en choisissant "Properties".

Le nombre d'états du STE du système complet est majoré par le produit des nombres d'états de ses composants parallèles. Quel est ce nombre ?

3.7 Minimisation modulo la bisimulation forte

Réduire modulo la bisimulation forte chacun des quatre STE générés à la question précédente. Pour cela, cliquer sur le fichier BCG, puis choisir "Reduce". Garder les paramètres par défaut ("Reduce using BCG_MIN", "Normal LTS", "Strong Equivalence" et ne pas modifier la valeur du champ "Output file").

Noter la taille de chacun de ces STE et calculer le majorant du produit comme à la question précédente.

Quel est le ratio entre les deux chiffres calculés ?

3.8 Génération du STE du système, bis

On tente à nouveau de générer le STE du système en composant les STE réduits obtenus à la question précédente. Pour cela on utilise le langage et l'outil EXP.OPEN [5], qui permet de composer des STE avec les opérateurs de composition parallèle et de masquage d'actions de LNT, ainsi que des vecteurs de synchronisation et d'autres opérateurs issus des langages CCS, CSP, mCRL, etc.

Cliquer sur le fichier `SCENARIO_A_COMPO.exp`, puis "Generate Labelled Transition System...". Ceci génère un fichier nommé `SCENARIO_A_COMPO.bcg`.

La génération du STE est-elle rapide ? Quelle est la taille du STE généré ? On pourra comparer cette taille au majorant calculé à la question précédente pour juger du ratio entre le nombre d'états théorique et le nombre d'états atteignables.

Réduire le STE `SCENARIO_A_COMPO.bcg` pour la bisimulation forte (cette opération prend environ une minute). Quelle est la taille du STE généré ? Observe-t-on une réduction supplémentaire ?

3.9 Vérification d'une propriété de logique temporelle

On considère la propriété MCL (*Model Checking Language*) décrite dans le fichier `prop.mcl`. Essayer de comprendre cette propriété.

L'outil EVALUATOR 4 [6] permettant de vérifier cette propriété n'étant pas encore parfaitement intégré dans CADP, il est nécessaire de faire quelques renommages d'étiquettes dans le STE du système. Ces renommages sont définis dans le fichier `flattening.ren`. Cliquer sur le fichier `SCENARIO_A_COMPO_s.bcg` puis choisir "Rename Labels...". Choisir le mode de renommage "Single partial" et le fichier `flattening.ren` et ne pas modifier le nom du fichier résultat. Cliquer sur OK. Vérifier que les étiquettes du fichier `SCENARIO_A_COMPO_s.bcg` ont bien été renommés conformément aux règles de renommages ("Display Labels").

De plus, pour vérifier la formule, il est nécessaire de taper la commande suivante dans un terminal :

```
bcg_open SCENARIO_A_COMPO_s.bcg evaluator4 prop.mcl
```

Cette propriété est-elle vraie ?

3.10 Réduction dépendant de la propriété

On a vérifié cette propriété sur un système dont toutes les étiquettes sont restées visibles. En fait, cette propriété permet de ne conserver que certaines étiquettes pertinentes et de réduire le STE modulo une relation d'équivalence faible, en l'occurrence la bisimulation de branchement sensible aux divergences (ou divbranching bisimulation).

On procède en deux étapes :

1. Cliquer sur le fichier `SCENARIO_A_COMPO_s.bcg` et choisir "Hide Labels...". Conserver le mode de masquage "Total", choisir le fichier `maximal.hid` et ne pas modifier le nom du fichier résultat. Cliquer sur OK. Vérifier que le fichier `SCENARIO_A_COMPO_s.bcg` ne contient plus que cinq étiquettes visibles dont les portes sont RECEIVE_A et SEND_A.
2. Réduire `SCENARIO_A_COMPO_s.bcg` modulo la divbranching bisimulation, ce qui produit un fichier nommé `SCENARIO_A_COMPO_s_d.bcg`.

Quel est la taille du STE après réduction ? Comparer avec les taille du STE réduit modulo la bisimulation forte. Quel est le gain de taille ?

Vérifier à nouveau la propriété sur `SCENARIO_A_COMPO_s_d.bcg`. Est-elle toujours vraie ?

3.11 Utilisation d'un script SVL

Utiliser l'interface graphique ou des appels en ligne de commande pour faire des vérifications poussées est assez laborieux. CADP contient un langage de script nommé SVL [1], qui permet d'écrire des scénarios de vérification en s'affranchissant d'une multitude de micro-tâches qui sont gérées automatiquement, telles que la gestion des fichiers intermédiaires.

Le fichier `demo.svl` décrit une grande partie des manipulations effectuées au cours de ce TP, et même un peu plus.

Regarder ce fichier et le comprendre.

On peut exécuter ce fichier en tapant la commande suivante :

```
svl demo
```

ou plus simplement :

```
svl
```

Une fois l'exécution terminée, le répertoire courant peut être automatiquement débarrassé des fichiers intermédiaires générés lors de l'exécution du script en tapant la commande suivante :

```
svl -clean demo
```

ou plus simplement :

```
svl -clean
```

Ajouter une ligne au script pour vérifier que le STE `SCENARIO_A_DB_MAXIMAL.bcg` est équivalent modulo la divbranching bisimulation à `SCENARIO_A_STRONG.bcg` une fois toutes les les étiquettes autres que celles présentes dans `SCENARIO_A_DB_MAXIMAL.bcg` ont été masqués.

References

[1] Hubert Garavel and Frédéric Lang. SVL: a Scripting Language for Compositional Verification. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2001 (Cheju Island, Korea)*, pages 377–392. IFIP, Kluwer Academic Publishers, August 2001. Full version available as INRIA Research Report RR-4223.

[2] Hubert Garavel, Frédéric Lang, and Radu Mateescu. Compositional Verification of Asynchronous Concurrent Systems using CADP. *Acta Informatica*, 52(4-5):337–392, 2015.

[3] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 15(2):89–107, April 2013.

[4] Hubert Garavel and Damien Thivolle. Verification of GALS Systems by Combining Synchronous Languages and Process Calculi. In Corina Pasareanu, editor, *Model Checking Software, Proceedings of the 16th International SPIN Workshop on Model Checking of Software SPIN'2009 (Grenoble, France)*, volume 5578 of *Lecture Notes in Computer Science*, pages 241–260. Springer-Verlag, June 2009.

[5] Frédéric Lang. EXP.OPEN 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-fly Verification Methods. In Jaco van de Pol, Judi Romijn, and Graeme Smith, editors, *Proceedings of the 5th International Conference on Integrated Formal Methods IFM'2005 (Eindhoven, The Netherlands)*, volume 3771 of *Lecture Notes in Computer Science*, pages 70–88. Springer Verlag, November 2005. Full version available as INRIA Research Report RR-5673.

[6] Radu Mateescu and Damien Thivolle. A Model Checking Language for Concurrent Value-Passing Systems. In Jorge Cuellar, Tom Maibaum, and Kaisa Sere, editors, *Proceedings of the 15th International Symposium on Formal Methods FM'08 (Turku, Finland)*, volume 5014 of *Lecture Notes in Computer Science*, pages 148–164. Springer Verlag, May 2008.

[7] Damien Thivolle. *Langages modernes pour la vérification des systèmes asynchrones.* Thèse de Doctorat, Université Joseph Fourier (Grenoble, France) and Universitatea Politehnica din Bucuresti (Bucharest, Romania), April 2011.