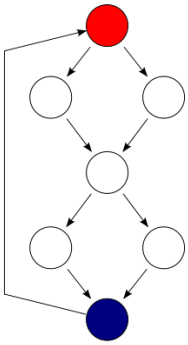
Projet SSC 2024-2025 : Vérification de formules CTL



Ce projet est à réaliser en trinômes. Indiquer absolument vos noms dans un fichier `README.md`**.**

L'objectif du projet est de programmer en Java un logiciel de *model checking* qui prendra en entrée un automate fini (donné dans une syntaxe donnée ci-dessous) et une formule CTL (donnée dans une syntaxe donnée ci-dessous), et qui va vérifier la validité de ladite formule sur l'état initial dudit automate.

Compilation et appel

Votre logiciel doit pouvoir être compilé puis exécuté sous un système Linux sans bibliothèque particulière préinstallée. Pour information, votre projet sera évalué dans un environnement Linux Mint 21.1. La syntaxe de compilation doit être indiquée dans votre `README.md`, idéalement sous la forme d'une unique commande et/ou d'un script.

Appel La syntaxe d'appel de base de votre logiciel doit être la suivante :

```
java VotreOutil automate.dot formule.ctl
```

Ce qu'affiche votre programme est relativement libre mais **doit** au moins afficher « `VRAI :-)` » si la formule est vraie, ou « `FAUX :-(` » si la formule est fausse. Une troisième valeur est « `ERREUR` » si votre logiciel contient une erreur (cas non prévu, erreur de syntaxe, etc.). Rendre « `ERREUR` » vaudra 0, mais rendre une mauvaise valeur coûtera des points négatifs.

```
java VotreOutil automate.dot formule.ctl
> VRAI :-)
```

Syntaxe d'entrée

Syntaxe du modèle

Le modèle (automate fini) utilise la syntaxe suivante, qui est aussi la syntaxe de dot.

```
digraph mon_systeme {
  /* Liste des nœuds avec leurs labels séparés par des espaces */
  q1 [label="P Q"];
  q2 [label="P"];
  q3 [label="Q"];
  q4 [label=""];

  /* Un seul nœud initial */
  q1 [shape=box];

  /* Transitions */
    q1 -> q2 -> q3;
    q2 -> q4;
    q4 -> q4;
}
```

Le système correspondant à ce modèle est représenté graphiquement sur la figure 1.

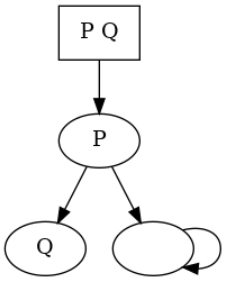


FIGURE 1 – Exemple de modèle

Pour des raisons de simplification, les hypothèses suivantes sont effectuées :

— les propositions atomiques sont indiquées dans le « `label` » du nœud et séparées par des espaces ;
— chaque proposition atomique est une chaîne de caractères commençant par une lettre (minuscule ou majuscule) et suivie d'un nombre arbitraire de lettres ou chiffres ;
— le modèle contient exactement un et un seul nœud initial, identifié par la syntaxe « `nomdunoeud[shape=box]` » (NB : cette syntaxe est également retenue pour vous aider à générer graphiquement et visualiser aisément vos modèles) ;
— les commentaires doivent être acceptés (syntaxe : `/* commentaire */`) et donc éliminés par votre outil ;
— on fait l'hypothèse que, dans le modèle d'entrée, les informations sont données dans l'ordre suivant :

1. la liste des nœuds avec leurs propositions atomiques ;
2. l'état initial ;
3. puis enfin les transitions.

(Si votre outil gère n'importe quel ordre, c'est évidemment mieux.)

Ne pas hésiter à utiliser d'éventuelles bibliothèques de manipulation de dot.

Syntaxe de la formule

Les formules peuvent être récursives, et ne peuvent inclure dans un premier temps que :

— les propositions atomiques du modèle
— `TRUE`, `FALSE`
— les connecteurs booléens (`OR`, `AND`, `NOT`), au niveau des propositions atomiques uniquement
— le `EX (...)`
— le `E (...) U (...)`
— le `A (...) U (...)`

Exemples de formules CTL dans cette syntaxe :

```
EX (P)
EX (P AND Q)
E (NOT Q) U (P OR Q)
E (P OR Q) U (Q AND NOT P)
A (EX (P)) U (A (P) U (NOT Q ))
```

Exemple 1

Reprenons le modèle donné sur la figure 1 et les formules ci-dessus.

Alors, une sortie attendue (sauf erreur du sujet !) est :

```
java VotreOutil automate.dot formules5.ctl
> Formule 1 : VRAI :-)
> Formule 2 : FAUX :-(
> Formule 3 : VRAI :-)
> Formule 4 : VRAI :-)
> Formule 5 : VRAI :-)
```

(on suppose ici pour simplifier l'exemple que le fichier `.ctl` prend plusieurs formules — ce n'est pas demandé)

Voir l'exemple 2 (premières formules) pour d'autres exemples.

Il n'est pas demandé de gérer les cas où la syntaxe des fichiers d'entrée serait incorrecte.

NB : en cas de difficulté, la récursion des formules peut être omise (c'est-à-dire que la 5e formule de l'exemple ci-dessus ne serait alors pas reconnue, car elle contient un `EX` et un `AU` à l'intérieur d'un `AU`). Cela occasionnera un retrait de points, mais sans donner une note catastrophique pour autant.

Afin de vous aider, un parser complet pour les formules CTL, rédigé en Java, est disponible à cette adresse :

`git@github.com:etienneandre/CTLproject.git`

Parties optionnelles (bonus)

Les améliorations suivantes feront l'objet de points attribués en sus du barème.

Plusieurs formules (difficulté : très facile)

Donner le résultat de la validité de plusieurs formules présentes consécutivement dans le fichier d'entrée. Par exemple, pour un fichier `formules.ctl` contenant 5 formules, le résultat pourrait être :

```
java VotreOutil automate.dot formules.ctl
> Formule 1 : VRAI :-)
> Formule 2 : VRAI :-)
> Formule 3 : FAUX :-(
> Formule 4 : FAUX :-(
```

```
> Formule 5 : VRAI :-)
```

CTL complet (difficulté : assez facile)

Améliorer votre outil afin que l'intégralité de CTL soit supportée, c'est-à-dire y compris les autres opérateurs omis précédemment :

— le =>, le parenthésage des connecteurs booléens, au niveau des propositions atomiques *et* des formules complexes ;

— la composition complète des `A` et `E` d'une part avec `F`, `G`, `U` et `X` d'autre part.

Exemples de formules dans cette syntaxe étendue :

```
NOT(AX (P))
EF (P AND Q)
AX ((P => (Q OR P)) OR AG(Q))
E (EF (P)) U (A (P) U (AG (Q => P)))
```

Exemple 2

Reprenons le modèle donné sur la figure 1.

Soit le fichier `formules6.ctl` suivant :

```
/* Formules simples */
EX (Q)
E (P) U (NOT P)
A (P) U (Q)
/* Formules complexes */
AF (NOT P)
AG (NOT P OR (EF (NOT Q)))
AX (AF (NOT (P OR Q)))
```

Alors, une sortie attendue (sauf erreur du sujet!) est :

```
java VotreOutil automate.dot formules6.ctl
> Formule 1 : FAUX :-(
> Formule 2 : VRAI :-)
> Formule 3 : VRAI :-)
> Formule 4 : VRAI :-)
> Formule 5 : VRAI :-)
> Formule 6 : FAUX :-(
```

Il est bien entendu possible de ne prendre en considération qu'une partie de cette extension. La partie de la syntaxe prise en compte devra alors être explicitée dans le `README.md`.

Réseau d'automates (difficulté : modérée à difficile)

Autoriser un *réseau* d'automates (au lieu d'un seul automate) dans le fichier d'entrée, dont le produit est effectué par synchronisation sur les étiquettes de transitions partagées.

Exemple :

```
digraph mon_systeme {
  subgraph aut1{
    /* Liste des nœuds avec leurs labels séparés par des espaces */
    q1 [label="P Q"];
    q2 [label="P"];
    q3 [label="Q"];
    q4 [label=""];

```

```
    /* Un seul nœud initial */
    q1 [shape=box];

    /* Transitions */
    q1 -> q2 -> q3 [label="a"];
    q2 -> q4;
    q4 -> q4 [label="b"];
  }
  subgraph aut2{
    /* Liste des nœuds avec leurs labels séparés par des espaces */
    l1 [label="P"];
    l2 [label="P"];
    l3 [label="Q R"];

    /* Un seul nœud initial */
    l1 [shape=box];

    /* Transitions */
    l1 -> l2 -> l3 [label="a"];
    l2 -> l1 [label="b"];
  }
}
```

Le système correspondant à ce modèle est représenté graphiquement sur la figure 2.

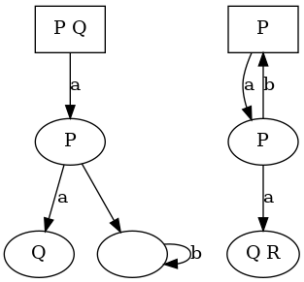


FIGURE 2 – Exemple de modèle synchronisé

Deux solutions recommandées :

1. effectuer une composition *statique* des différents automates afin d'obtenir un unique automate, puis appeler votre outil dessus (relativement facile à implémenter) ; ou
2. effectuer la construction *dynamique* (à la volée) de vos automates, pendant l'appel à l'algorithme de CTL (plus difficile à implémenter… mais beaucoup plus efficace pour de gros automates !).

Modalités de rendu

Le langage de programmation doit être Java.

Le rendu s'effectuera sur Moodle sous la forme d'une archive unique (`.zip` ou `.tar.gz` uniquement) contenant l'ensemble de vos sources, ainsi qu'un fichier `README.md` décrivant brièvement l'utilisation de l'outil. Aucun fichier superflu (fichiers de gestionnaire de version, fichiers compilés…) n'est autorisé.

Calendrier

Présentation du sujet	octobre 2024
Date limite de rendu	lundi 11 novembre 2024 23h59 *strict*

Source : https://en.wikipedia.org/wiki/File:Control_flow_graph_of_function_with_two_if_else_statements.svg

Version : 21 octobre 2024