Modélisation et vérification

Hamza Benkabbou, Kevin Bourgeois, Jérémy Morosi,
Jean-Baptiste Perrin, Zo Rabarijaona

20 janvier 2013

Justification

Cette partie décrit la manière dont le programme calcule et mémorise la justification d'une formule CTL pour ensuite en donner une représentation visuelle.

Preuve

L'interface `IPreuve` (listing 1) définie les méthodes nécessaires pour prouver qu'une formule CTL est vraie.

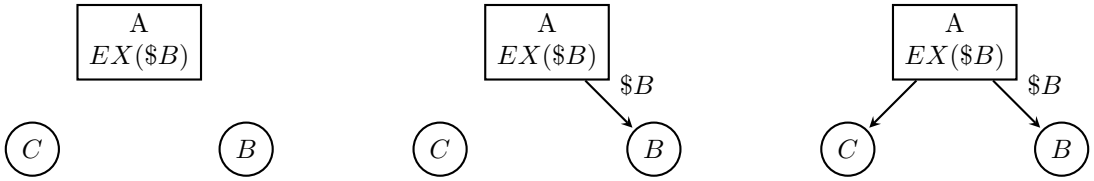
La structure d'une preuve est très simple, puisqu'elle consiste en la formule qui lui est associée (`getFormule`), en la liste des états qui sont vrais (`getMarquage`, `setMarquage`) et en la liste des sous-preuves pour chaque morceau de la formule (`getPreuves`).

Une preuve dispose aussi de plusieurs méthodes pour la visualiser. On peut récupérer la formule qui lui est associée au format textuel grâce à la méthode `formuleToString`. On peut obtenir un affichage sous forme d'arbre de la preuve et de ses sous-preuves grâce à la méthode `toTree`. On peut récupérer la formule sous la forme d'un label coloré pour l'exportation au format `.dot` grâce à la méthode `toDotLabel`. Et finalement, on peut exporter la preuve au format `.dot` grâce aux méthodes `toDotRacine` et `toDot`.

Lors du calcul de la preuve, les méthodes `couperRacine` et `couper` permettent de supprimer les états qui sont inutiles car ils n'ont pas de prédécesseurs dans la preuve parent.

Affichage au format .dot

L'exportation de la preuve au format `.dot` se fait au travers des méthodes `toDotRacine` et `toDot`. Il y a deux méthodes car l'affichage de la preuve complète est différent de celui des sous-preuves. Pour l'état de départ (celui pour lequel on doit justifier la formule CTL), on affiche son marquage ainsi que le label de la formule complète (figure 1a). On appelle ensuite `toDotRacine` qui va appeler à son tour la méthode `toDot` pour chaque sous-preuve de la preuve complète. Chaque sous-preuve va alors ajouter au graphe des flèches partants de l'état parent, allants vers les états validants la sous-preuve et ayants le label de la formule associée à côté (figure 1b). Les sous-preuves appellent également la méthode `toDot` pour leurs sous-preuves. Une fois que toutes les sous-preuves ont été ajoutées au graphe, on le complète en ajoutant les flèches manquantes entre deux états (figure 1c).



(a) Preuve complète (b) Sous-preuves (c) Flèches manquantes

FIGURE 1 – Affichage de la preuve pour $EX(\$B)$

Pour l'ajout des flèches manquantes, on utilise une map de type `Map<Integer, Set<Integer>>` où les clefs sont les numéros des états et les valeurs sont les listes des états auxquels ils sont reliés. Pour s'assurer qu'une même preuve ne soit affichée qu'une fois (figure 2), on utilise une liste de type `Set<String>` où les éléments sont les flèches (au format `.dot`) du graphe.

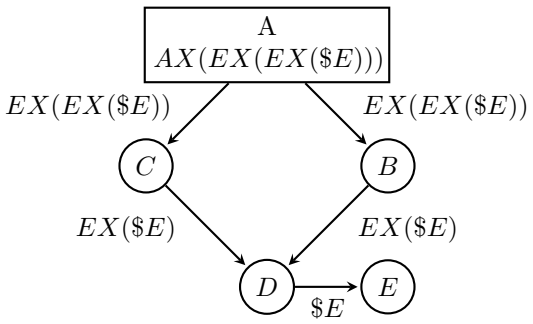


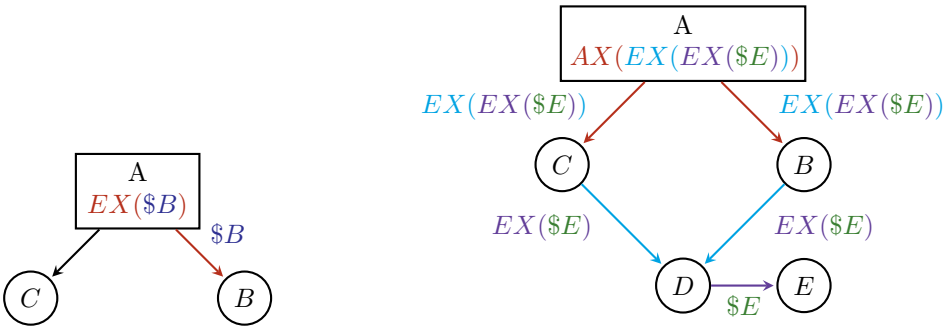
FIGURE 2 – Affichage de la preuve pour $AX(EX(EX(\$E)))$

Coloration

La coloration des preuves est gérée par la classe `Coloration` (listing 2). Son but est d'associer une couleur unique (si possible) et un label coloré (pour l'exportation au format `.dot`) à une formule. Lors de l'exportation, les accesseurs `getCouleur` et `getLabel` permettront alors de récupérer les informations liées à la formule donnée.

Les couleurs sont générées par la méthode `genererCouleur`. Dans l'idéal, toutes les couleurs utilisées doivent être uniques et ne doivent pas trop se ressembler pour que la preuve soit suffisamment claire. Ici, on génère juste des couleurs au format HSV avec $s = 1$, $v = 0.75$ et en incrémentant le h d'une certaine valeur à chaque appel de la méthode.

La couleur d'une formule est utilisée pour colorer une flèche partant d'un état validant la formule et allant vers un état validant une sous-formule alors que le label de la sous-formule est affiché sur la flèche (figure 3).



(a) $EX(\$B)$ (b) $AX(EX(EX(\$E)))$

FIGURE 3 – Exemples de coloration

Preuves atomiques

Les preuves atomiques sont les preuves associées aux formules atomiques p, $true$, $false$, $dead$ et $initial$. Elles correspondent respectivement aux classes `Atom`, `True`, `False`, `Dead` et `Initial`. Comme leur implémentation est assez simple puisqu'elle n'ont pas de sous-preuves, on montrera juste les affichages obtenus au format textuel (figure **??**) et au format `.dot` (figure **??**).

Création de la preuve

ht

```java
public interface IPreuve {

  public Tree getFormule();
  public boolean[] getMarquage();
  public void setMarquage(boolean[] marquage);
  public List<IPreuve> getPreuves();

  public void couperRacine(CTL ctl, int[][] pred, int etat);
  public void couper(CTL ctl, int[][] pred, boolean[] parents);

  public String toTree();
  public String toTree(String indent);

  public void toDotRacine(Map<Integer, Set<Integer>> fleches,
      Set<String> justifications, IPreuve parent, int etatParent,
      Coloration couleurs);
  public void toDot(Map<Integer, Set<Integer>> fleches,
      Set<String> justifications, IPreuve parent, int etatParent,
      Coloration couleurs);

  public String toDotLabel(Coloration couleurs);
  public IPreuve clone();
  public String formuleToString();

}
```

Listing 1 – Interface `IPreuve` commune à toutes les preuves

```java
public class Coloration {

  private Map<Tree, String> couleursFormules;
  private Map<Tree, String> labelsFormules;

  public String getCouleur(Tree formule);
  public String getLabel(Tree formule);

  public void ajouter(Tree formule, String couleur, String label);
  public FakeTree ajouter(String label);

  public String genererCouleur();

}
```

Listing 2 – Classe `Coloration`