Informatique *TP*
Logique- Ilias Mouline Ipesup

1 Monsieur Bizarre et les portes quantifiées

Ce problème étudie la modélisation et la résolution d'énigmes logiques complexes faisant intervenir des quantifications sur un domaine fini, avec implémentation d'un solveur automatisé.

I. Le labyrinthe de Monsieur Bizarre

Monsieur Bizarre a conçu un labyrinthe comportant quatre portes numérotées de 1 à 4. Chaque porte porte une inscription et mène soit à la sortie (succès), soit à un piège (échec). Les inscriptions sont les suivantes :

— Porte 1 : « *Toutes les portes mentent.* »
— Porte 2 : « *Exactement deux portes disent la vérité.* »
— Porte 3 : « *La porte 1 dit la vérité si et seulement si la porte 4 ment.* »
— Porte 4 : « *Au moins une porte mène à la sortie.* »

On suppose que chaque inscription est soit vraie, soit fausse. Une porte qui mène à la sortie ne dit pas nécessairement la vérité, et inversement.

1. Modélisation formelle

On introduit les prédicats suivants pour $i \in \{1, 2, 3, 4\}$:

— $V(i)$: « La porte i dit la vérité. »
— $S(i)$: « La porte i mène à la sortie. »

(a) Traduire chacune des quatre inscriptions sous forme de formules logiques utilisant les prédicats $V(i)$ et $S(i)$.
(b) On suppose de plus que *exactement une porte mène à la sortie*. Exprimer cette contrainte sous forme d'une formule logique.
(c) Donner la formule complète Φ modélisant l'ensemble des connaissances.

2. Résolution par déduction

(a) Montrer que la porte 1 ne peut pas dire la vérité.
(b) En déduire la valeur de vérité de l'inscription de la porte 4.
(c) Déterminer combien de portes disent la vérité. Justifier soigneusement.
(d) Identifier la porte qui mène à la sortie.

3. Vérification par table de vérité

(a) Construire la table de vérité complète du système (16 lignes correspondant aux valuations de $V(1), V(2), V(3), V(4)$).
(b) Ajouter une colonne pour la contrainte « exactement une porte mène à la sortie » (on ne précise pas encore laquelle).
(c) Identifier les lignes compatibles avec toutes les contraintes et vérifier le résultat de la question précédente.

II. Généralisation à n portes

On considère maintenant un labyrinthe à n portes ($n \geq 2$). Chaque porte i porte une inscription qui est une formule du premier ordre close (sans variables libres) construite à partir :

— des prédicats unaires $V(j)$ et $S(j)$ pour $j \in \{1, \ldots, n\}$,
— des connecteurs logiques $\neg, \wedge, \vee, \rightarrow$,
— des quantificateurs $\forall j$ et $\exists j$ sur l'ensemble des portes.

Par exemple, l'inscription de la porte i pourrait être :

$$\forall j,\ (j \neq i) \rightarrow \neg S(j).$$

4. Modélisation algorithmique

On représente une inscription par un arbre de syntaxe abstraite.

(a) Proposer en `OCaml` un type `expr` pour représenter ces formules.
(b) Écrire une fonction

```
eval : bool array -> bool array -> expr -> bool
```

qui prend en argument deux tableaux de booléens v et s (de longueur n) indiquant les valeurs de $V(i)$ et $S(i)$, une expression e, et retourne la valeur de vérité de e.

5. Problème de décision

On note φ_i l'inscription de la porte i. Le système est cohérent s'il existe des valuations $(V(i))_{1 \leq i \leq n}$ et $(S(i))_{1 \leq i \leq n}$ telles que pour tout i :

$$V(i) \leftrightarrow \varphi_i \text{ est vraie.}$$

(a) Montrer que pour n fixé, le problème de décider si un tel système est cohérent est dans NP.
(b) Montrer que le problème SAT se réduit à ce problème pour un n approprié. En déduire que le problème est NP-complet.

6. Solveur par force brute

Écrire en `OCaml` une fonction

```
solve : expr array -> (bool array * bool array) option
```

qui prend un tableau d'expressions (φ_i) et retourne une solution si elle existe.

(a) Décrire l'algorithme utilisé.
(b) Quelle est sa complexité en fonction de n ?
(c) Proposer une optimisation simple permettant d'éliminer rapidement des valuations inconsistantes.

III. Analyse et extensions

7. Portes autoréférentes

Une inscription est dite *autoréférente* si elle fait référence à la véracité de la porte elle-même (par exemple, $V(i)$ apparaît dans φ_i).

(a) Montrer que tout système sans autoréférence et sans quantificateurs peut être résolu en temps polynomial.
(b) Donner un exemple de système avec autoréférence mais sans quantificateurs qui n'a pas de solution.
(c) Donner un exemple de système avec autoréférence et quantificateurs qui a exactement deux solutions.

8. Quantification restreinte

On modifie le langage des inscriptions en n'autorisant que des quantifications sur les autres portes :

$$\forall j \neq i \qquad \text{et} \qquad \exists j \neq i.$$

(a) Montrer que tout système de ce type peut être réduit à un système sans quantificateurs (mais avec plus de portes).
(b) En déduire que le problème de cohérence reste NP-complet même avec cette restriction.

9. Programmation dynamique

Pour certaines classes d'inscriptions, on peut résoudre le problème plus efficacement.

(a) On suppose que chaque φ_i est une conjonction de littéraux sur les $V(j)$ et $S(j)$, sans quantificateurs. Proposer un algorithme en $O(2^n \cdot n^2)$ pour résoudre le système.
(b) Améliorer cet algorithme en $O(2^n \cdot n)$ en utilisant une approche par programmation dynamique.

Exercices

1.1 Implémentation optimisée

Écrire en C une fonction :

```
bool solve_optimized(expr *phi, int n, bool *V, bool *S)
```

qui implémente l'algorithme de la question précédente.

(a) Décrire la structure de données utilisée pour représenter les expressions.
(b) Expliquer comment évaluer rapidement une conjonction de littéraux pour une valuation donnée.
(c) Analyser la complexité en temps et en mémoire.

1.2 Juge

Un juge veut libérer de la place dans ses prisons en donnant une chance de sortir aux prisonniers qui sont les plus à même de réfléchir sur leur forfait. Il présente un prisonnier devant deux portes et déclare :

> « Voici deux portes. Chacune porte une indication. »

Le prisonnier le constate en effet :

— sur la première porte, il est indiqué : *« Cette porte conduit à la liberté, l'autre à ta cellule. »*
— sur la seconde porte, il est indiqué : *« L'une des portes conduit à la liberté, l'autre à ta cellule. »*

Le prisonnier demande :

> « Est-ce que c'est vrai ? »

Alors le juge, qui veut privilégier les prisonniers ayant un jugement fiable, ajoute :

> « L'une des indications est vraie, l'autre est fausse. Sauras-tu reconnaître celle qui dit vrai ? Montre-le moi en choisissant la bonne porte. »

1. On modélise ce problème en utilisant l'algèbre de Boole. On prendra deux valeurs booléennes l_1 et l_2, qui indiquent respectivement que la porte 1 et la porte 2 conduisent à la liberté.
   Les indications sur les portes peuvent également être modélisées comme deux propositions logiques p_1 et p_2.
   Écrire les expressions booléennes de ces deux propositions.
2. Pour modéliser l'affirmation du juge, c'est-à-dire : *« l'une des indications est vraie, l'autre est fausse »*, on utilisera deux nouvelles propositions logiques p_3 et p_4.
   Écrire les expressions booléennes de ces deux propositions.
3. Simplifier la proposition p_3 en utilisant les lois de la logique propositionnelle.
4. Simplifier la proposition p_4 en utilisant les lois de la logique propositionnelle.
5. Construire la table de vérité correspondant aux propositions p_3 et p_4.
6. Déterminer la porte que le prisonnier doit choisir pour atteindre la liberté. Justifier.

7. Vérifier la table de vérité des propositions p_3 et p_4 en utilisant `OCaml` en mode interpréteur.
   Le code doit être écrit dans un fichier nommé :

   `Nom_Booleens_Juge.ml`

   dont l'exécution est rappelée ci-dessous :

   ```
   # #use "Nom_Booleens_Juge.ml";;
   ```

8. Créer le circuit correspondant aux propositions p_3 et p_4.

1.3 Jeu télévisé

La présentatrice d'un jeu télévisé présente deux boîtes à un participant et explique :

> « Dans chaque boîte vous trouvez soit un gain de 1 000 €, soit une perte des gains accumulés jusque-là, mais rien d'autre. A priori, le contenu de chaque boîte est indépendant de ce que contient l'autre. Mais des indications sur les boîtes vous en diront davantage. »

— Sur la boîte bleue, il est indiqué : *« La boîte bleue vous fait perdre vos gains. »*
— Sur la boîte rouge, il est indiqué : *« Au moins une des boîtes contient un gain de 1 000 €. »*

La présentatrice précise :

> « Soit les deux indications sont correctes, soit elles sont toutes les deux fausses. »

Puis elle propose au participant d'ouvrir la boîte de son choix, les deux, ou aucune.

1. Proposer une modélisation à l'aide de l'algèbre de Boole du problème posé.
2. En simplifiant les propositions à l'aide de l'algèbre de Boole, expliquer quelle devrait être la stratégie gagnante du joueur.
3. Faire la table de vérité correspondant aux propositions.
4. Calculer la valeur de vérité de la modélisation en utilisant `OCaml`.
5. Créer le circuit logique correspondant à la modélisation.

La présentatrice du jeu télévisé propose toujours une boîte rouge et une boîte bleue, portant chacune la même indication :

> « Les deux boîtes sont gagnantes. »

Cependant, cette fois-ci les règles sont différentes.

— Pour la boîte bleue, l'indication qu'elle porte :
  — est fausse si la boîte contient un gain de 1 000 € ;
  — est vraie si la boîte contient une perte des gains accumulés jusque-là.
— À l'inverse, pour la boîte rouge, l'indication qu'elle porte :
  — est vraie si la boîte contient un gain de 1 000 € ;
  — est fausse si la boîte contient une perte des gains accumulés jusque-là.

Dire ce que doit jouer le participant.

Justifier ce choix avec une méthode choisie après avoir modélisé le problème à l'aide de l'algèbre de Boole.

1.4 Complément à deux

L'additionneur binaire permet l'addition de deux nombres binaires représentés sous la forme d'une série de booléens allant de a_0 à a_{n-1} et de b_0 à b_{n-1}.

Il produit une série de booléens résultants notés de r_0 à r_{n-1} comme cela a été étudié précédemment.

On s'intéresse ici à la synthèse d'un circuit qui réalise le complément à deux d'un nombre.

Le complément à deux d'un nombre binaire est un codage astucieux de l'information de telle sorte que les additionneurs binaires fonctionnent aussi bien avec les nombres positifs qu'avec les nombres négatifs (voir section 4.2.1, Représentation des nombres entiers, page 92).

Selon cette représentation, le nombre opposé $-a$ d'un nombre a positif de n bits est obtenu en prenant l'opposé de chaque bit de a puis en ajoutant 1 au nombre résultant de cette inversion.

C'est cette fonction booléenne qu'il s'agit de concevoir ici.

On note les bits du nombre binaire a codé sur n bits de a_0 à a_{n-1} et le nombre produit $r = -a$ décomposé lui aussi en n bits de r_0 à r_{n-1}.

Le bit de poids fort de cette représentation en complément à deux (c'est-à-dire a_{n-1} et r_{n-1}) indique qu'un nombre est positif s'il vaut 0 et négatif s'il vaut 1.

On note aussi c_i la retenue qu'il convient de propager du i-ième bit au $(i+1)$-ième bit.

1. Donner la formule booléenne qui permet de calculer r_0 en fonction de a_0.
2. Donner la formule booléenne qui donne c_0 en fonction de a_0.
3. Donner la formule booléenne qui permet de calculer r_i en fonction de a_i et de c_{i-1}, pour $i \in [1, n-1]$.
4. Donner la formule booléenne qui permet de calculer c_i en fonction de a_i et de c_{i-1}, pour $i \in [1, n-1]$.
5. Donner le circuit logique associé à la formule booléenne de r_0.
6. Donner le circuit logique associé à la formule booléenne de c_0.
7. Donner le circuit logique associé à la formule booléenne de r_i, $\forall i \in [1, n-1]$.
8. Donner le circuit logique associé à la formule booléenne de c_i, $\forall i \in [1, n-1]$.
9. Démontrer qu'il n'y a qu'un nombre positif a composé de n bits dont l'opposé en complément à deux est lui aussi positif.

   Indication : Sachant que $a_{n-1} = 0$, déterminer dans quel cas $r_{n-1} = 0$.
10. Démontrer qu'il n'y a qu'un nombre négatif a composé de n bits dont l'opposé en complément à deux est lui aussi négatif.

    Indication : Sachant que $a_{n-1} = 1$, déterminer dans quel cas on a $r_{n-1} = 1$.