Vérification logicielle et synthèse de programmes – TP1

0 Installation

Pour ce TP, vous aurez besoin d'installer le solveur SMT Z3 et son API Python. Z3 est disponible dans les dépôts de paquets de la plupart des distributions Linux (par exemple sur Ubuntu, l'installation peut se faire avec la commande `sudo apt-get install z3`). L'API Python peut être installée via `pip` avec la commande `pip install z3-solver`.

1 Résolution d'énigme

Dans cet exercice, nous allons utiliser un solveur SMT pour résoudre l'énigme suivante :

> — Un habitant du manoir Dreadbury a tué Agatha.
> — Agatha, le majordome et Charles sont les trois seuls habitants du manoir Dreadbury, et sont trois personnes distinctes.
> — Un tueur déteste nécessairement sa victime, et n'est jamais plus riche qu'elle.
> — Charles ne déteste aucune des personnes qu'Agatha détestait.
> — Agatha détestait tout le monde à l'exception du majordome.
> — Le majordome déteste toutes les personnes qui ne sont pas plus riches qu'Agatha, ainsi que toutes les personnes qu'Agatha détestait.
> — Personne ne déteste tout le monde.
> — **Qui a tué Agatha ?**

1. Le fichier `agatha.smt2` contient une transcription au format SMT-LIB2 de cette énigme, mais il manque les faits liées à la richesse. Complétez le fichier en déclarant un prédicat binaire `richer` et en ajoutant les hypothèses manquantes.
2. Formalisez les faits suivants en logique du premier ordre, puis utilisez le solveur SMT Z3 pour les prouver :

— Agatha se détestait elle-même, et le majordome la détestait aussi. Charles ne la détestait pas.
— Le majordome est plus riche qu'Agatha.
— Agatha s'est tuée elle-même.

3. À première vue, le domaine de discours (les personnages) ne semble contenir que trois éléments. Toutefois l'hypothèse "personne ne déteste toute le monde" introduit potentiellement d'autres personnes dans le problème. Expliquez comment ces autres personnes sont représentées formellement en logique du premier ordre (considérez la traduction de l'hypothèse en logique du premier ordre, puis sa version Skolemisée).

2 API Python

Pour cet exercice et les suivants, nous allons utiliser l'API Python de Z3, qui nous permettra de générer des contraintes de manière programmatique et d'invoquer Z3 sur celles-ci. Le fichier `z3_tutorial.py` fournit des exemples de l'utilisation de cette API. Étudiez-le et exécutez les exemples.

3 Sudoku

L'objectif de cet exercice est d'utiliser Z3 pour trouver la solution de la grille de Sudoku ci-dessous. Pour ce faire, complétez le programme fourni dans le fichier `sudoku.py` avec les contraintes manquantes.

								6
			8	6		2		
		3	9					8
	2						7	
6	8	9						
	3						2	5
					5	1	4	
9					3			
		1		2				

4 Chaînes de mots

Soit le jeu suivant : étant donné un mot de départ et un mot d'arrivée, essayez de former une chaîne de mots, telle que chaque mot ne diffère du précédent que par une seule lettre. Par exemple pour le mot de départ *hope* et le mot d'arrivée *dawn*, on peut former la chaîne suivante :

hope - hips - lips - laps - laws - lawn - dawn

Le programme `wordgame.py` contient la base de l'encodage du jeu. Un mot est représenté par un tableau d'entiers, où chaque entier représente un caractère [1] et le passage d'un mot à un autre se fait via un `store`, puisqu'un seul élément du tableau peut être modifié.

1. Complétez ce programme pour vérifier qu'on peut passer du mot *hope* au mot *hips* avec un seul mot intermédiaire.
2. Généralisez le programme pour qu'il trouve la plus courte chaîne permettant de passer d'un mot à un autre. Pour cela, vous devrez résoudre le problème plusieurs fois de suite, en incrémentant le nombre de mots intermédiaires autorisés à chaque fois, jusqu'à trouver une solution.

1. La fonction Python `ord` permet d'obtenir l'entier associé à un caractère, tandis que la fonction `chr` effectue l'opération inverse.