Picobot

PHILIPPON BOSSUT BAILLEUL

3 avril 2015

Table des matières

1 Introduction

Picobot est un programme utilisé comme support au cours d'informatique du Dr. Zachary Dodds au Harvey Mudd College de Californie.

Son but est de simuler le parcours d'un robot dans un environnement donné en utilisant des règles dictant son comportement.

Ce projet utilise le langage de spécification déclaratif Alloy pour générer ces règles. L'intérêt est de chercher le plus petit ensemble de règles qui complète le niveau, quelque soit la position initiale du robot.

2 Picobot

2.1 Environnement

Un environnement est modélisé sous forme d'un tableau où le robot doit parcourir au moins une fois sur chaque case grâce aux règles pré-établies. Comme le robot n'est pas doté de mémoire interne alors il ne peut pas savoir où il se trouve. Il ne peut donc pas utiliser d'algorithmes de recherche de chemin (Dijkstra, A*).

2.2 Règles

Une règle est formée de la façon suivante :
CurrentState Surroundings -> MovementDirection NewState

— *CurrentState* et *NewState* sont deux nombres entiers (comme 0,1,2, etc).
— *MovementDirection* est soit N, E, W, S, ou X. «X» signifie "ne pas bouger".

Une fois ces quatre facteurs déterminés, la formule suivante est calculée et son résultat est sauvegardée en mémoire. Quand toutes les méthodes sont évaluées, l'algorithme renvoie la liste des dix méthodes les plus suspectes.

2.3 Exemple

3 Résultats

4 Conclusion

5 Exemple Java

```
package examples

/** This examples illustrates the use of Guarded
*  Algebraic Data Types in Scala. For more information
*  please refer to the Scala specification available
*  from http://scala-lang.org/docu/.
*/
object gadts extends Application {

  /* The syntax tree of a toy language */
  abstract class Term[T]

  /* An integer literal */
  case class Lit(x: Int) extends Term[Int]

  /* Successor of a number */
  case class Succ(t: Term[Int]) extends Term[Int]

  /* Is 't' equal to zero? */
  case class IsZero(t: Term[Int]) extends Term[Boolean]

  /* An 'if' expression. */
  case class If[T](c: Term[Boolean],
  t1: Term[T],
  t2: Term[T]) extends Term[T]

  /** A type-safe eval function. The right-hand sides can
  *  make use of the fact that 'T' is a more precise type,
  *  constraint by the pattern type.
  */
  def eval[T](t: Term[T]): T = t match {
    case Lit(n)      => n

    // the right hand side makes use of the fact
    // that T = Int and so it can use '+'
    case Succ(u)     => eval(u) + 1
    case IsZero(u)   => eval(u) == 0
    case If(c, u1, u2) => eval(if (eval(c)) u1 else u2)
  }
  println(
  eval(If(IsZero(Lit(1)), Lit(41), Succ(Lit(41)))))
}
```