EFREI Master 2

Mini-Project : Implementation of a CTL Model chekcer

1 Preamble

Model checking is a powerful formal verification approach. Given a formal model of a system and a temporal formula expressing a correctness property, the model checking technique allows to check whether the system satisfies the formula. It is based on an exhaustive exploration of the reachable state space of the system. Thus, it suffers from the well known state explosion problem. In this project, we aim to implement a CTL model checker based on the algorithms seen in class. We represent the system with a *Kripke structure* : a graph where the nodes/states are labeled with a subset of satisfied atomic propositions.

1.1 Preliminaries

1.1.1 Kripke structure

Definition 1 (Kripke structure)
Let AP *be a finite set of atomic propositions. A Kripke structure (*KS for short*) over* AP *is a 4-tuple* $\langle \Gamma, L, \rightarrow, s_0 \rangle$ *where :*

— Γ is a finite set of states *;*
— $L : \Gamma \rightarrow 2^{AP}$ is a labeling (or interpretation) function *;*
— $\rightarrow \subseteq \Gamma \times \Gamma$ is a transition relation *;*
— $s_0 \in \Gamma$ is the initial state *;*

1.1.2 CTL Logic

Syntax :
CTL state formulae over the set AP of atomic propositions are formed according to the following grammar :

— $\Phi ::= true \mid p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists\varphi \mid \forall\varphi \mid$

Where $p \in AP$ and φ is a path formula : $\varphi ::= X\varphi \mid \varphi_1 U \varphi_2$
Abreviations :

— $\varphi_1 \implies \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
— $F\varphi$: now or sometimes in the future
  — $F\varphi \equiv true\ U\varphi$
— $G\varphi$: now and always in the future
  — $G\varphi \equiv \neg F\neg\varphi$

Semantics :
Let s be a state, Φ and Ψ are state CTL formulae and φ a CTL path formula. $Paths(s)$ denotes the set of paths starting from s.

— $s \models p$ iff $p \in L(s)$
— $s \models \Phi \wedge \Psi$ iff $s \models \Phi$ and $s \models \Psi$
— $s \models \neg\Phi$ iff not $s \models \Phi$
— $s \models \exists\varphi$ iff $\pi \models \varphi$ for some $\pi \in Paths(s)$
— $s \models \forall\varphi$ iff $\pi \models \varphi$ for all $\pi \in Paths(s)$

For path π, the satisfaction relation for path formulae is defined by :

— $\pi \models X\varphi$ iff $\pi[1] \models \phi$

— $\pi \models \varphi_1 U \varphi_2$ iff $\exists i \geq 0$ s.t. $\pi[i] \models \varphi_2$ and $\forall 0 \leq j < i \wedge \pi[j] \models \varphi_1$

Where for path $\pi = s_0.s_1 \ldots$ and $i \geq 0$, $\pi[i]$ denotes the $(i+1)$th state of π i.e., $\pi[i] = s_i$

A Kripke Structure satisfies a CTL formula Φ iff $s_0 \models \Phi$

1.1.3 Algorithms

In the following, we recall the algorithms to check CTL formulae. These algorithms can be combined and recursively called on sub-formulae to compute the set of states of a Kripke Structure that satisfy the given formula. If the initial state of the system KS belongs to this set, then the system satisfies the formula. Only a subset of possible sub-formulae are considered here.

You must design the algorithms corresponding to the other sub-formulae such as $\phi = AX\psi$, $\phi = AF\psi$, $\phi = AG\psi$, $\phi = \psi_1 \vee \psi_2$, $\phi = \psi_1 \implies \psi_2$ ...

— ***Case 1 :*** $\phi = p$:

Procedure marking(ϕ)

case $\phi = p$ **do**
  forall $q \in Q$ **do**
    if $p \in l(q)$ **then**
      $q.\phi$:=`true`
    else
      $q.\phi$:=`false`

— ***Case 2 :*** $\phi = \neg\psi$

marking(ψ);
forall $q \in Q$ **do**
  $q.\phi$:=$\neg q.\psi$

— ***Case 3 :*** $\phi = \psi_1 \wedge \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  $q.\phi$:=$q.\psi_1 \wedge q.\psi_2$

— ***Case 4 :*** $\phi = EX\psi$

marking(ψ);
forall $q \in Q$ **do**
  $q.\phi$:=`false`
forall $(q, \_, q') \in T$ **do**
  if q'.ψ=`true` **then**
    $q.\phi$:=`true`

— ***Case 5 :*** $\phi = E\psi_1 U \psi_2$

```
marking(ψ1);
marking(ψ2);
forall q ∈ Q do
    q.ϕ :=false;
    q.seenbefore :=false
L :=∅;
forall q ∈ Q do
    if q.ψ2=true then
        L :=L∪{q}
while L≠ ∅ do
    pick q from L ; L :=L\{q};
    q.ϕ :=true;
    forall (q′, _, q) ∈ T do
        if q'.seenbefore=false then
            q'.seenbefore :=true;
            if q'.ψ1=true then
                L :=L∪{q'}
```

— ***Case 6 :*** $\phi = A\psi_1 U \psi_2$

```
marking(ψ1);
marking(ψ2);
L :=∅;
forall q ∈ Q do
    q.nb :=degree(q);
    q.ϕ :=false
forall q ∈ Q do
    if q.ψ2=true then
        L :=L∪{q}
while L≠ ∅ do
    pick q from L ; L :=L\{q};
    q.ϕ :=true;
    forall (q′, _, q) ∈ T do
        q'.nb :=q'.nb - 1;
        if q'.nb=0 and q'.ψ1=true
        and q'.ϕ=false then
            L :=L∪ {q'}
```

2 Requirements

The aim of the project is to implement a CTL model checker. The target tool has two imputs :

1. A Kripke structure describing the behavior of the system
2. a CTL fromula

The output of the model checker is whether the formula is satisfied by the system or not. To achieve this task you have to solve the following issues :

1. Clearly determine the syntax of the inputs (the KS and the CTL formula)
2. Choose the appropriate date structures of the inputs
3. Implement the parsers that will check the correctness of the inputs and fill the corresponding data structures
4. Implement the algorithms corresponding to the different possible sub-formulae

You are free to choose the programming language to be used for the implementation of the model checker.

3 Improvements

Some improvements can be considered through optional functionalities.

— Graphical interface : Allows to draw a graph (KS) and enter a CTL formula
— A random generator of KSs from a number of states and a set of atomic propositions. This allows to test the limits of your model checker when the size of the KS is big.
— A random generator of CTL properties.
— A plugin allowing to generate the KS from a formal models such as Petri nets or state machines.

4 Project planning

1. Publication of project subject : January 3d, 2022
2. Project follow-up sessions : January 4th, 2022, January 12th, 2022
3. Project intermediary defense : January 19th, 2022 (SE2) and January 25th, 2022 (SE1)
4. Submission of code sources and report (on the moodle) : February 6th, 2022 at 11 :59 pm.

You will work in trinomial (at most) and you have to upload your work on the moodle as a compressed folder containing the following elements :

— A structured pdf document (around 10 pages) containing the description of your input formats, the chosen data structures and the design of the complementary verification algorithms. The document should also contain a discussion on the encountered difficulties, the possible improvements and a conclusion.
— The source code,
— A readme file explaining the technical points allowing the execution of your code,
— A Sample folder containing some examples of systems and CTL formulae to be used to test your implementation.

Have fun