Automatic Verification of Systems
Final Project - Symbolic LTL Model Checking

Omri Lifshitz
205490675
omrilifshitz@mail.tau.ac.il

Idan Berkovits
205404130
berkovits@mail.tau.ac.il

1 Abstract

2 Introduction

3 CTL and LTL Model Checking

Let us begin by describing the temporal logic CTL*, and from there continue to describe both Computation Tree Logic (CTL) and Linear Temporal Logic(LTL). CTL* contains two types of formulae: state formulae which hold for specific states and path formulae which hold along a specific path. State formulae are given by the following rules [1]:

- If $p \in AP$ (Atomic propositions) then p is a state formula.
- If $f,\ g$ are state formulae then $\neg f, f \vee g$ are state formulae.
- If f is a path formula, then Ef ("for some computation path") is a state formula.

Path formulae are defined by the following rules:

- If f is a state formula, it is also a path formula.
- If $f,\ g$ are path formulae then $\neg f,\ f \vee g,\ Xf$ and fUg are also path formulae.
- If f is a path formula, then Ef ("for some computation path") is a state formula.

Let $M = (S, R, L)$ be a Kripke structure where S is the set of states, R is the transition relation (we assume there is a total transition relation) and L be the labeling function.

Definition 1 *A path in Kripke structure M is an infinite sequence of states $\pi = s_0, s_1, \ldots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in R$.*

The notation $M, s \models f$ is used to state that f holds for state s in the Kripke structure M, and similarly for $M, \pi \models f$ for path π in M.

Definition 2 *Let f_1, f_2 be state formulae and g_1, g_2 path formulae. The relation $\models$ is defined recursively as follows:*

$$
\begin{aligned}
M, s \models p &\iff p \in L(s) && (1)\\
M, s \models \neg f_1 &\iff M, s \not\models f_1 && (2)\\
M, s \models f_1 \vee f_2 &\iff M, s \models f_1 or M, s \models f_2 && (3)\\
M, s \models E g_1 &\iff \text{there is a path } \pi \text{ starting at } s \text{ such that } M, \pi \models g_1 && (4)\\
M, \pi \models f_1 &\iff s \text{ is the first state of } \pi \text{ and } M, s \models f_1 && (5)\\
M, \pi \models \neg g_1 &\iff M, \pi \not\models g_1 && (6)\\
M, \pi \models g_1 \vee g_2 &\iff M, \pi \models g_1 \text{ or } M, \pi \models g_2 && (7)\\
M, \pi \models X g_1 &\iff M, \pi^1 \models g_1 && (8)\\
M, \pi \models g_1 U g_2 &\iff \exists k \geq 0 \text{ s.t. } M, \pi^k \models g_2 \text{ and } \forall 0 \leq j \leq k \; M, \pi^j \models g_1 && (9)
\end{aligned}
$$

Definition 3 *CTL is the subset of CTL* obtained by specifying the path formulae using the following rules[1]:*

- *If f, g are state formulae, then Xf and fUg are path formulae.*
- *If f is a path formulae, then so is $\neg f$.*

From the definition we can see that CTL is a subset of CTL* that only permits branching operators. Therefore, in this new logic, all linear-time operators can only appear if they are immediately preceded by path quantifiers.

Definition 4 *LTL is a subset of CTL restricted to all formulae of the form Af where f is a path formula in which the only state sub-formulae are atomic propositions[1]. More precisely, path formulae here are defined in the following recursive manner:*

- *An atomic proposition.*
- *If f, g are path formulae, then $\neg f$, $f \vee g$, Xf and fUg are also path formulae.*

4 Symbolic Model Checking - BDDs

5 Implementation

5.1 Parsing formulae

5.2 Creating the tableau

5.3 Creating the product structure

5.4 Finding Maximal SCC

As taught in class, in order to assert that there is a fair path in the product structure, we need to find a strongly connected component (SCC) that is reachable from the initial state and that intersects with each one of the fairness constraints.

In this section we will present the algorithm use to find the SCCs in a given model using BDDs. Our algorithm is based the algorithm explained in [2].

Algorithm 1 SCC Decomposition

procedure SCC_Decomp(N,V)

$V' \leftarrow V$

while $V' \neq 0$ **do**

$v \leftarrow random\_take(V')$

$B(v) \leftarrow backward_set(v, V', N)$

$SCC\_DECOMP\_RECUR(v, B(v), N)$

$V' \leftarrow V' \wedge \overline{v \vee B(v)}$

Algorithm 2 Finite maximum distance predecessors

procedure FMD_Pred(W, U, N)

$pred \leftarrow 0$

$front \leftarrow W$

$bound \leftarrow U$

while $front \neq 0$ **do**

$x \leftarrow \exists_Y front(Y) \wedge N(X,Y) \wedge bound(X)$

$y \leftarrow \exists_Y bount(Y) \wedge N(X,Y) \wedge bound(X)$

$front \leftarrow x \wedge \overline{y}$

$pred \leftarrow pred \vee front$

$bount \leftarrow bounts \wedge \overline{front}$

return $pred$

Algorithm 3 Recursive method to find SCCs

procedure SCC_DECOMP_RECUR(v, B(v), N)

$F(v) \leftarrow forward\_set(v, B(v), N)$

if $F(v) \neq 0$ **then**

report $F(v)$ an SCC

else

report v non- SCC

$x \leftarrow F(v) \vee v$

$R \leftarrow B(v) \wedge \overline{x}$

$y \leftarrow$ FMD_PRED(x, R, N)

report y non-SCC

$R \leftarrow R \wedge \overline{y}$

$IP \leftarrow \exists_Y (y \vee x)(Y) \wedge N(X, Y) \wedge R(X)$

while $R \neq 0$ **do**

$v \leftarrow random\_take(IP)$

$B(v) \leftarrow backward_set(v, R, N)$

$SCC\_DECOMP\_RECURE(v, B(v), N)$

$R \leftarrow R \wedge \overline{v \vee B(v)}$

$IP \leftarrow IP \wedge \overline{v \vee B(v)}$

5.5 Finding fair paths

6 Experiments and Results

7 Further Work

References

[1] Edmund M. Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. Another look at ltl model checking. *Formal Methods in System Design*, 10(1):47–71, Feb 1997.

[2] Aiguo. Xie and P A. Beerel. Implicit enumeration of strongly connected components. pages 37–40, Nov 1999.