Playing Tic-Tac-Toe Game Using Model Checking

CS 545 Course Project, Spring 04 (re-edited Feb 07)
Zheng Zhang, *zzhang18@uic.edu*
University of Illinois at Chicago

ABSTRACT

We consider the problem of designing a program to play tic-tac-toe game with a human. We tranform the original problem into a model checking problem. The game is modeled as a transition system where evey player's move transfers the system from one state to another. We designed a game strategy based on winning state. Then we formally define winning state in different model checking framework, including ATL, μ-calculus and bounded model checking with CTL, and finally solved the formalized model checking problem using a model checker SMV.

1. INTRODUCTION

In this section, we introduce some background and the problem to be solved.

1.1 What is Tic-Tac-Toe Game?

Tic-Tac-Toe (called TTT for short in the rest of this report) is a game played between two players on a 3×3 board. The two players take turns to put pieces on the board. A single piece is put for each turn and piece once put does not move. A player wins the game by first lining three of his or her pieces in a straight line, no matter horizonal, vertical or diagonal. Figure 1 shows one instance of a TTT game play where player 1 takes the first turn and finally wins the game by achieving a diagonal line.

1.2 Problem Statement

We consider a computer program playing TTT game with a human. The program makes his best to win the game. Min-max search is usually used to play games, but we focus on solving this problem using model checking.

1.3 Related Work

Madhusudan et al [1] summarized several model checking approaches that are used to play games. We applied their methods to a specific game and used model checker tools to solve the game. Our game model is slightly different from the model in [1]. Moreover, we used bounded model checking with CTL, while bounded model checking with QBF and SAT methods instead of CTL were proposed in [1].

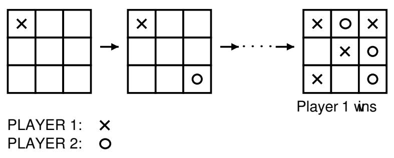

Figure 1: One instance of a TTT game play

1.4 Roadmap

Section 2 formulates the problem. Section 3 discusses different approaches to solve the problem using model checking, including ATL model checking, μ-calucus, and bounded model checking with CTL. Section 4 presents the experience and results of using existing model checker tools to solve this problem. Section 5 discusses some further issues on game and model checking. Section 6 concludes.

2. MODELING THE GAME

In this section, we model the game as a transition system. We define winning state and design the computer's game strategy based on winning state.

2.1 Transition System and Winning State

The TTT game can be viewed as a transition system, of which the state is the placement of all the pieces on the board. Because of the finite size of game board (3×3) and finite types of pieces (two types, i.e. "X" and "O"), the transition system has finite states. Each "put" made by the player is a transition, which transfers the game from one state to another. A state can transite to multiple states because a player may have multiple choices of putting his or her piece.

A state is a *winning* state of a player if the player can win the game by following a certain sequence of proper choices, no matter how his or her opponent puts his or her pieces afterwards. That is to say, once game enters this state, the player will surely win the game. It can be seen from the definition that each player has a set of win states and they do not overlap.

2.2 Game Strategy

Suppose we are writing a program to play TTT game with a human and the program makes his best to win the game. Because the number of states is finite, we can enumerate all the winning states for the computer and all the winning states for the human. Then we write the program with following instructions:

1. Avoid let the game enter a winning state of the human in the next step
2. Make a choice such that the game will transit to a winning state of the computer in the next step
3. If such a choice does not exist, pick up a choice randomly

The remaining problem is how to determine the winning states. We first formalize what is a winning state and then determine the winning states using model checker.

2.3 Transition System Formulation

Grids on the board is labled with numbers from 0 to 8, with the grids on the uppermost horizontal line labeled 0, 1, 2 from left to right, and so on with other grids. A set of variables X_i ($i = 0, 1, ..., 8$) represent the pieces on the grids. That is,

$$X_i = \begin{cases} 0, & \text{if grid i is empty} \\ 1, & \text{if grid i is occupied by player 1} \\ 2, & \text{if grid i is occupied by player 2} \end{cases}$$

Variable T represents who would take the comming turn. That is,

$$T = \begin{cases} 1, & \text{if player 1 would take the coming turn} \\ 2, & \text{if player 2 would take the coming turn} \end{cases}$$

Thus the combination of X_i and T completely represent the state of the game. The successive states of a given current state are the states that have one more "1" on an empty grid if $T = 1$, or the states that have one more "2" on an empty grid if $T = 2$. A win state is a state where the following propositional formula Win is satisfied,

$$\begin{aligned} P1Win \;:=\; & \{ \bigvee_{i=0,3,6} [(X_i = 1) \wedge (X_{i+1} = 1) \wedge (X_{i+2} = 1)]\} \\ \vee\; & \{ \bigvee_{i=0,1,2} [(X_i = 1) \wedge (X_{i+3} = 1) \wedge (X_{i+6} = 1)]\} \\ \vee\; & [(X_0 = 1) \wedge (X_4 = 1) \wedge (X_8 = 1)] \\ \vee\; & [(X_2 = 1) \wedge (X_4 = 1) \wedge (X_6 = 1)] \end{aligned}$$

$$\begin{aligned} P2Win \;:=\; & \{ \bigvee_{i=0,3,6} [(X_i = 2) \wedge (X_{i+1} = 2) \wedge (X_{i+2} = 2)]\} \\ \vee\; & \{ \bigvee_{i=0,1,2} [(X_i = 2) \wedge (X_{i+3} = 2) \wedge (X_{i+6} = 2)]\} \\ \vee\; & [(X_0 = 2) \wedge (X_4 = 2) \wedge (X_8 = 2)] \\ \vee\; & [(X_2 = 2) \wedge (X_4 = 2) \wedge (X_6 = 2)] \end{aligned}$$

$$Win := P1Win \vee P2Win$$

Note that a win state has no successors, even if there are empty grids.

We define TTT game as a finite state transition system $S_{TTT} = \langle V, Q, \pi, \delta, Q_0 \rangle$. A transition system has following components.

- V is a set of variables. $D(V)$ is the set product of the value domain of each variable in X.
- Q is a set of states.
- π labels the each state with a distinct element in $D(V)$. $\pi : Q \to D(V)$, is a injection, mapping every state to a set of variable values.
- $\delta : Q \to 2^Q$, denotes transitions, which associates each state with its successive states.
- $Q_0 \in Q$ is the set of inital states.

The finite state transition system for TTT game is as follows,

(Finite State Transition System for the TTT game)

- $V = \{X_0, X_1, ..., X_8, T\}$
- $Q = \{q_0, ..., q_{n-1}\}$, where $n = 3^9 \times 2$
- $\pi : q_i \to (x_0, x_1, ..., x_8, t)$, where $i = 3^9(t-1) + \sum_{j=0}^{8} 3^j x_j$
- $\delta : q_i \to R \subseteq Q$. R is $\{q_i\}$ if $q_i \models \neg Terminate$, where $Terminate$ is formulated as follows,

$$\begin{aligned} Boardfull \;&:=\; \bigwedge_{i=0}^{8} (X_i \neq 0) \\ Terminate \;&:=\; P1Win \vee P2Win \vee Boardfull \end{aligned}$$

  Otherwise, suppose $\pi(q_i) = (x_0, x_1, ..., x_8, t)$, then δ maps q_i to the set of all states $q_i{}'$, where $\pi(q_i{}') = (x_0{}', x_1{}', ..., x_8{}', t')$, such that the following proposition formula is satisfied,

$$\begin{aligned} &\tau(q_i, q_i{}') := [(t = 1) \to (t' = 2)] \wedge [(t = 2) \to (t' = 1)] \\ &\wedge \bigvee_{k=1}^{8} [(x_k = 0) \wedge (x_k{}' = t) \bigwedge_{j=0,1,\ldots,8; i \neq k} (x_j = x_j{}')] \end{aligned}$$

- $Q = \{q_i \mid \pi(q_i) = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1)\}$

In the following sections, a state q_i and its labeling $(x_0, x_1, ..., x_8, t)$ are used exchangeably.

3. WINNING STATE FORMULATION

In this section, we formally define winning state in different model checking framework, including ATL, μ-calculus and bounded model checking with CTL.

3.1 Alternating-time Temporal Logic

It seems that a winning state can be expressed in a CTL formula. In order to decide whether a state $(x_0, x_1, ..., x_8, t = 1)$ is a winning state for player 1, we check whether the following formula is satisfied at this state,

$$EXAXEXAXEXAX \cdots P1Win$$

However, this is NOT a valid CTL formula, because CTL does not allow a "$\cdots$". This made us resort to Alternating-time Temporal Logic (ATL) proposed by R. Alur et al in [2], which is more expressive than CTL. CTL allows existentital and universal quantification over all paths. In contrast, ATL offers selective quantification over those paths that are possible outcomes of games, such as the game in which the system and the enviroment alternate moves. Although the motivation of ATL is for open system and exploring modularity in model checking, not solving games, we can use it to solve our problem.

We model TTT game as a turn-based synchronous ATS. This ATS is the same as the finite state transition system mentioned above, except that in this ATS, there are two agents: $player1$ and $player2$, which are scheduled to take turns to put pieces on the board. In [2], a turn-based synchronous ATS is represented by a 6-tuplet $S = \langle \Pi, \Sigma, Q, \pi, \sigma, R \rangle$. We follow the convention of finite state transition system and use a $ATS_{TTT} = \langle V, Q, \pi, \delta, Q_0, \Sigma, \sigma \rangle$, where Σ and σ have the same meaning as in [2] and the other symbols have the same meaning as above mentioned. That is, Σ and σ are defined as follows,

$$\begin{aligned} \Sigma \;&:\; \{player1, player2\} \\ \sigma \;&:\; (x_0, x_1, ..., x_8, t) \to player1, \text{ if } t = 1; player2, \text{ if } t = 2. \end{aligned}$$

Thus a state $(x_0, x_1, ..., x_8, t)$ is a winning state for $player1$ iff the following ATL formula is satisfied at this state,

$$P1Winning1 := \ll player1 \gg F\ P1Win$$

3.2 μ-calculus

The μ-calculus citeUVAis a proposition modal logic extended with the least fixpoint operator and is interpreted over Kripke structure. It can express temporal logic such as LTL and CTL. In μ-calculus, a formula have another meaning. It can represent the set of all states such that the formula is satisfied at it. We can compute the set of all winning states for player 1 as follows. First of all, all win states for player 1 are winning states. That is,

$$Z_0 := P1Win$$

Secondly, all states at which player 1 have a strategy of letting the system enter states in Z_0 in the next step, no matter how player 2 behaves, are winning states. That is,

$$Z_1 := \{(t=1) \wedge \langle a \rangle Z_0\} \vee \{(t=2) \wedge [a] Z_0\}$$

("$\langle\rangle$" and "[]" are two operators in μ-calculus.) Similarly, we have,

$$\begin{aligned} Z_2 &:= \{(t=1) \wedge \langle a \rangle Z_1\} \vee \{(t=2) \wedge [a] Z_1\} \\ &\ldots \\ &\ldots \\ Z_{k+1} &:= \{(t=1) \wedge \langle a \rangle Z_k\} \vee \{(t=2) \wedge [a] Z_k\} \\ &\ldots \end{aligned}$$

The computation terminates if there exists some j such that $Z_{j+1} = Z_j$. Then Z_j is the set we are looking for. In μ-calculus, we are computing a least fix point, so we have,

$$P1Winning2 := \mu X.\{P1Win \vee ((t=1) \wedge \langle a \rangle X) \vee ((t=2) \wedge [a] X)\}$$

3.3 Bounded Model Checking with CTL

In bounded model checking, given a transition system S, a temporal logic formula f and a user-supplied bound $k \in N$, we construct a formula f_k, which is satisfiable if and only if the formula f is valid along some path of length k. In our problem, we have some knowledge about this k. Any path would at a length of no more than 9, run into a state in which the path loops forever, simply because the board has only $3 \times 3 = 9$ grids. Thus, we can set $k = 9$ and construct the f_k, which is a CTL formula as follows,

$$\begin{aligned} P1Winning3 \;&:=\; [(t=1) \wedge (EXAX)^4 EXP1Win] \\ &\vee\; [(t=2) \wedge (AXEX)^4 AXP1Win] \end{aligned}$$

where the superscript 4 means repeating the quoted sequence 4 times.

By model-checking this CTL formula, we can solve our problem.

4. PROBLEM SOLVING

We have discussed three methods to solve the problem by model checking. In this section, we use model checkers to check the formulas.

SMV is a tool for checking CTL. We wrote a script in SMV and found out all the winning states. SMV works as expected. As an example, the initial state is expected not to be a winning state for player 1, and SMV gives a "false". The state (1, 2, 0, 0, 0, 0, 0, 0, 0, 1) is expected to be a winning state for player 1, and SMV gives a "true". In terms of performance, SMV is very quick in model checking. For most of the cases, the tool spends less than a second producing the results.

Unfortunately, we could not get Mocha[4] and μcke[5] working. The former one is a tool for checking ATL and the latter one for μ-calculus.

5. DISCUSSIONS

In this section, we dicuss the limitations of our model checking approach in games.

5.1 Chess

It is interesting whether our model checking approach applies to general games. In section 2, we said that the program randomly pick up a choice if no choice that leads to a winning state exists. However if we apply it to a more complex game like chess, this approach needs to be improved. The key problem lies in the states which are not the winning state of either player. For example, we have a state which has 10 successors, 8 of which are winning states; we have another state which also has 10 successors, 8 of which are winning states of the opponent. The former state is more advantageous than the latter one, because in the former state the human has larger chance of making a mistake that leads the game into the computer's winning state. But in our program, we do not distinguish those two states. The regularly used min-max search is a good method in games, in which a sophisticated evaluatin function rather than winning state is used to evaluate states. However model checking winning state is not meaningless in this case. Knowing which states are winning states can be helpful in evaluting the states in Min-Max search.

5.2 Nim

Our approach cannot deal with infinite-state game such as Nim. Nim game is a game played between two players. It begins with several piles of matches. The players alternate their moves. On each move the player selects one of the piles and removes at least one match from that pile. The last player to remove a match wins. The "standard version" of Nim begins with three piles with 3, 4 and 5 matches respectively.

It is known that the player who pick up the first turn has a strategy to win the game. This can be verified by showing the initial state of a Nim game, e.g. the standard version, is a winning state with the model checking method.

However, can we prove that for all kinds of Nim game (all Nim game such as three piles with 3, 4, 5, and four piles with 1, 2, 3, 4 and so on), the first player has a winning strategy? It is true that for all Nim games, the first player has a winning strategy? Can we verify this with model checking? We can image a "super-Nim" game: the first player select a initial configuration, say 4 piles with 1, 2, 3, 4, than the players begin a normal Nim game with the second player taking the first turn. Thus we are to verify the initial state is a winning state for the second player. However, the initial state has infinite number of successors, which goes beyond our finite model checking. Generally speaking, model checking can not be done on infinite state system, except that there is abstrations or bisimulation from this infinite state system to a finite state system. But as for this "super-Nim" problem, we currently do not see any possibility of abstration or bisimulation to a finite state system. It is mentioned in [7] that some infinite game have been lifted to pushdown graphs.

6. CONCLUSION

We consider the problem of designing a program to play tic-tac-toe game with a human. We tranform the original problem into a model checking problem. The game is modeled as a transition system where evey player's move transfers the system from one state to another. We designed a game strategy based on winning state. Then we formally define winning state in different model checking framework, including ATL, μ-calculus and bounded model checking with CTL, and finally solved the formalized model checking problem using a model checker SMV.

7. REFERENCES

[1] P. Madhusudan, W. Nam, R. Alur, *Symbolic Computational*

Techniques for Solving Games, Workshop on Bounded Model Checking, 2003.

[2] R. Alur, T.A. Henzinger, O. Kupferman. *Alternating-time temporal logic*, 38th IEEE Symposium on Foundations of Computer Science, pp. 100-109, 1997

[3] *Modal Logic Course Spring 2003*, http://staff.science.uva.nl/$\sim$yde/teaching/ML/mu/

[4] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. *MOCHA: Modularity in model checking*. In Proc. of the 10th Int'l Conf. on Computer-aided Verification, vol. 1427 of LNCS, pages 521-525. Springer-Verlag, 1998.

[5] A. Biere. *μcke - Efficient μ-calculus model checking*. In Proc. of the 9th Int'l Conf. on Computer-Aided Verification, vol. 1254 of LNCS, pages 468-471. Springer-Verlag, 1997.

[6] K.L.McMilan, *SMV*, http://www-2.cs.cmu.edu/$\sim$modelcheck/smv.html

[7] Wolfgang Thomas, *Infinite Games and Verification (Extended Abstract of a Tutorial)*, CAV 2002: 58-64.