safeAI | checking logical models

ANNELINE DAGGELINCKX, MATTHIJS KEMP, and OTTO MÄTTAS, Utrecht University, The Netherlands

1 WEEK 6 ASSIGNMENTS

1.1 Describing Concurrent Game Structures

Below, a description for concurrent game structure (CGS) is given. CGS incorporates multiple elements, include the set of agents and states and actions taken simultaneously, a valuation of propositions, specific actions available to a specific agent in a specific state and also a deterministic transition function that assigns outcome states to states and tuples of actions.

A concurrent game structure (CGS) is a tuple

$$M_{rps} = (\{a_1, a_2\}, \{q_0, q_1, q_2\}, v, \{rock, paper, scissors\}, d, o) \quad (1)$$

, where

V is defined as:

- $V(win_1) = \{q_1\}$
- $V(win_2) = \{q_2\}$

d is defined as:

- $d(a, q) = \{rock, paper, scissors\}$
- $\forall a \in \{1, 2\}, q \in \{q_0, q_1, q_2\}$

o is defined as:

- $o(q_i, rock, rock) = o(q_i, paper, paper) = o(q_i, scissors, scissors) = q_0$
- $o(q_i, rock, scissors) = o(q_i, paper, rock) = o(q_i, scissors, paper) = q_1$
- $o(q_i, scissors, rock) = o(q_i, rock, paper) = o(q_i, paper, scissors) = q_2$
- $\forall i \in \{0, 1, 2\}$

1.2 Expressions in Coalition Logic

Coalition Logic helps explain whether there is an action by a coalition of agents A such that whatever action other agents perform, A can make sure some property holds. Below, the given system is expressed in coalition logic.

$$\{a_1, a_2\} win_1 \wedge \{a_1, a_2\} \neg win_1 \quad (2)$$

The formula is true in q_0 in CGS M_{rps} because when we take two possible worlds, one where win_1 is true such as (paper, rock) and the inverse where win_2 is true such as (rock, paper).

1.3 Manipulating Concurrent Game Structures

Below, a representation of a fixed strategy of the first player is given in CGS.

$$M_{rps1f} = \langle\{1, 2\}, \{q_{ij}\}, V, \{rock, paper, scissors\}, d, o\rangle \quad (3)$$

, where $i \in 0, 1, 2$, $j \in r, p, s$ and

V is defined as:

- $V(win1) = \{q_{1j}\}, \forall j \in \{r, p, s\}$
- $V(win2) = \{q_{2j}\}, \forall j \in \{r, p, s\}$
- (q_{0j} represents a tie, a situation where no players win)

d is defined as:

- $d(1, q_{ir}) = rock, \forall i \in \{0, 1, 2\}$
- $d(1, q_{ip}) = paper, \forall i \in \{0, 1, 2\}$
- $d(1, q_{is}) = scissors, \forall i \in \{0, 1, 2\}$
- $d(2, q_{ij}) = \{rock, paper, scissors\}, \forall i \in \{0, 1, 2\}, j \in \{r, p, s\}$

o is defined as:

- $o(q_{ir}, rock, rock) = q_{0p}$
- $o(q_{ip}, paper, paper) = q_{0s}$
- $o(q_{is}, scissors, scissors) = q_{0r}$
- $o(q_{ir}, rock, scissors) = q_{1p}$
- $o(q_{ip}, paper, rock) = q_{1s}$
- $o(q_{is}, scissors, paper) = q_{1r}$
- $o(q_{ir}, rock, paper) = q_{2p}$
- $o(q_{ip}, paper, scissors) = q_{2s}$
- $o(q_{is}, scissors, rock) == q_{2r}$
- $\forall i \in \{0, 1, 2\}$

1.4 Expressions in Alternating-time Temporal Logic

Below, an expression of a game is given in Alternating-time Temporal Logic (ATL), where a required strategy for player 2 to always win has been described.

$$\langle\langle 2\rangle\rangle X\Big(\langle\langle 2\rangle\rangle G(win2)\Big) \quad (4)$$

Furthermore, a a memoryless strategy for player 2 that makes this formula true in any state in M_{rps1f} has been described as well:

- In q_{ir} play paper;
- In q_{ip} play scissors;
- In q_{is} play rock.

1.5 Properties in Alternating-time Temporal Logic

Below, an example of an ATL property and a CGS, where the bound 2 is not sufficient for a strategy to enforce the property is described.

$$M_5 = \langle\{\{a\}, \{q, r\}, V, \{stay, move\}, d, o\rangle \quad (5)$$

,where V is defined as:

- $V(p) = q$

d is defined as:

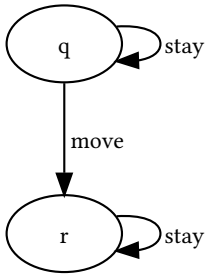
- $d(a, q) = \{stay, move\}$
- $d(a, r) = \{stay\}$

o defined as:

- $o(q, stay) = q$
- $o(r, stay) = r$
- $o(q, move) = r$

$$\varphi = \langle\langle a\rangle\rangle(Xp \wedge XXp \wedge XXX\neg p) \quad (6)$$

Authors' address: Anneline Daggelinckx, a.daggelinckx@students.uu.nl; Matthijs Kemp, m.g.r.kemp@students.uu.nl; Otto Mättas, o.mattas@students.uu.nl, Utrecht University, P.O. Box 80125, Utrecht, Utrecht, The Netherlands, 3508 TC.



2 WEEK 7 ASSIGNMENTS

2.1 Designing Concurrent Game Structures in ISPL

A description for an concurrent game structure in Interpreted Systems Programming Language (ISPL) has been added as an appendix A. The world domain (variables, actions, initial state, formulae) is given in the Coursework 3 document, assignment W7-1.

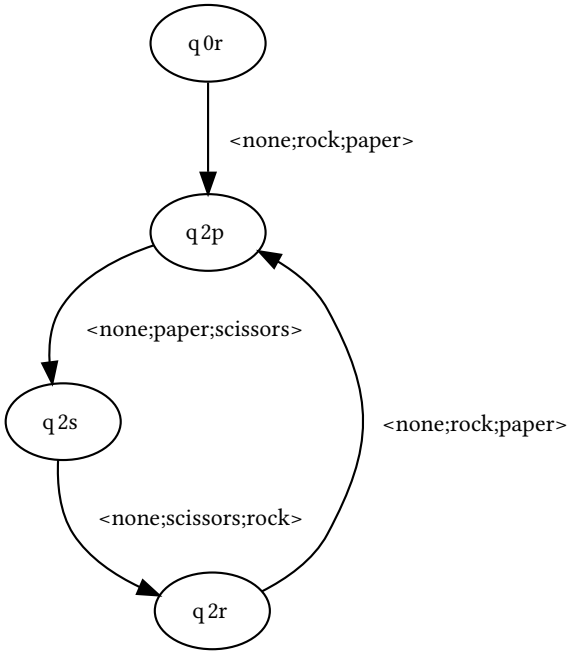
To continue, the witness strategy for the formula of exercise W6-4 is described as follows:

Let s_2 be the strategy function for agent 2, where

- $s_2(q_{2r}) = paper$
- $s_2(q_{2p}) = scissors$
- $s_2(q_{2s}) = rock$

The other states will never be reached if player 2 plays this strategy. To achieve completeness, a definition of a witness strategy can be as follows:

- $s_2(q_{0r}) = paper$
- $s_2(q_{ip}) = scissors$
- $s_2(q_{is}) = rock$
- $s_2(q_{ir}) = paper$
- $\forall i \in \{0, 1, 2\}$



2.2 Describing scenarios in ISPL

A description for an concurrent game structure in Interpreted Systems Programming Language (ISPL) has been added as an appendix B. The world domain (variables, actions, initial state, formulae) is given in the Coursework 3 document, assignment W7-2.

The thief can steal the painting and escape while not being caught:

$$\langle\langle Thief \rangle\rangle \Big((\neg caught) U (stolen \wedge escaped) \Big) \tag{7}$$

2.3 Modifying scenarios in ISPL

A description for an concurrent game structure in Interpreted Systems Programming Language (ISPL) has been added as an appendix C. The world domain (variables, actions, initial state, formulae) is given in the Coursework 3 document, assignment W7-3.

The guard can always prevent the picture from being stolen:

$$\langle\langle Guard \rangle\rangle G(\neg stolen) \tag{8}$$

2.4 Verifying the encoding

A coalition consisting of the thief and the guard can ensure that the painting is stolen and the thief exits without being caught:

$$\langle\langle Thief, Guard \rangle\rangle \Big(\neg caught U (stolen \wedge escaped) \Big) \tag{9}$$

2.5 Comments on encodings

A description for an concurrent game structure in Interpreted Systems Programming Language (ISPL) has been added as an appendix D. The world domain (variables, actions, initial state, formulae) is given in the Coursework 3 document, assignment W7-5.

There is a room with three switches; x, y and z. The switches can be on (1) or off (0). At each timestep, the agent (bert) has to turn one of the switches from on to off or the opposite. Bert has three actions; switch_x, switch_y and switch_z. Let us abbreviate the number of switches (which is equal to the number of variables) to nvars. For each switch we need two evolution statements. Therefore, the ISPL translation of the relation has size $2 * nvars$ and its size is of order $O(nvars)$. There are 2^{nvars} states and for each state there are three actions. The CGS transition relation is thus of size $2^{nvars} * nvars$ and therefore its size is of order $O(2^{nvars})$.

Since the ISPL representation of the transition relation has size $O(nvars)$ and the CGS representation has size $O(2^{nvars})$, the latter is exponentially larger than the first one.

A INTERPRETED SYSTEM IN ISPL

```
-- INFOMLSAI 2021 an interpreted system designed by
-- ANNELINE DAGGELINCKX, MATTHIJS KEMP and OTTO MATTAS

Agent Environment
   Vars:
      state: {q0r, q0p, q0s, q1r, q1p, q1s, q2r, q2p, q2s};
   end Vars
   Actions = { none };
   Protocol:
      Other : { none };
    end Protocol
   Evolution:
      state=q0r if (state=q0s or state=q1s or state=q2s) and
           (Player1.Action = scissors and Player2.Action =
           scissors);
      state=q1r if (state=q0s or state=q1s or state=q2s) and
           (Player1.Action = scissors and Player2.Action =
           paper);
      state=q2r if (state=q0s or state=q1s or state=q2s) and
           (Player1.Action = scissors and Player2.Action =
           rock);

      state=q0p if (state=q0r or state=q1r or state=q2r) and
           (Player1.Action = rock and Player2.Action = rock);
      state=q1p if (state=q0r or state=q1r or state=q2r) and
           (Player1.Action = rock and Player2.Action =
           scissors);
      state=q2p if (state=q0r or state=q1r or state=q2r) and
           (Player1.Action = rock and Player2.Action =
           paper);

      state=q0s if (state=q0p or state=q1p or state=q2p) and
           (Player1.Action = paper and Player2.Action =
           paper);
      state=q1s if (state=q0p or state=q1p or state=q2p) and
           (Player1.Action = paper and Player2.Action =
           rock);
      state=q2s if (state=q0p or state=q1p or state=q2p) and
           (Player1.Action = paper and Player2.Action =
           scissors);
   end Evolution
end Agent


Agent Player1
   Lobsvars = {state};
   Vars:
      -- this agent doesn't have other variables than the
           locally observable variables, but ispl forces you
           to define one
      dummy : {val};
   end Vars
    Actions = {rock, paper, scissors};
    Protocol:
        Environment.state=q0r or Environment.state=q1r or
             Environment.state=q2r : {rock};
        Environment.state=q0p or Environment.state=q1p or
             Environment.state=q2p : {paper};
        Environment.state=q0s or Environment.state=q1s or
             Environment.state=q2s : {scissors};
    end Protocol
    Evolution:
        dummy = val if dummy = val;
    end Evolution
end Agent


Agent Player2
   Lobsvars = {state};
   Vars:
      dummy : {val};
   end Vars
    Actions = {rock, paper, scissors};
    Protocol:
        Other: {rock, paper, scissors};
    end Protocol
    Evolution:
        dummy = val if dummy = val;
    end Evolution
end Agent


Evaluation
    win1 if (Environment.state=q1r or Environment.state=q1p
         or Environment.state=q1s);
    win2 if (Environment.state=q2r or Environment.state=q2p
         or Environment.state=q2s);
end Evaluation

InitStates
   Environment.state = q0r and
   Player1.dummy = val and
   Player2.dummy = val;
end InitStates

Groups
   g1 = {Player1};
   g2 = {Player2};
   g_all = {Player1,Player2};
end Groups

Formulae
   -- Test formulae
   <g2> X win2;
   <g2> F win1;
   --!<g1> X win1;
   <g_all> F win1;

   -- Formula w6-4
   <g2> X(<g2> G win2);
end Formulae
```

B INTERPRETED SYSTEM IN ISPL

```
-- INFOMLSAI 2021 an interpreted system designed by
-- ANNELINE DAGGELINCKX, MATTHIJS KEMP and OTTO MATTAS

Semantics=SingleAssignment;

Agent Environment
   Vars:
      guardRoom: {room0, room1, room2};
      painting: boolean;
   end Vars
   Actions = { none };
   Protocol:
      Other: { none };
    end Protocol
   Evolution:
      guardRoom = room0 if (guardRoom=room1 and
           Guard.Action=left);
      guardRoom = room1 if (guardRoom=room0 and
           Guard.Action=right) or (guardRoom=room2 and
           Guard.Action=left);
      guardRoom = room2 if (guardRoom=room1 and
           Guard.Action=right);

      painting = false if (Thief.Action=steal);
   end Evolution
end Agent


Agent Guard
   Lobsvars = {guardRoom};
   Vars:
      direction: {left, right};
   end Vars
    Actions = {left, right};
    Protocol:
      direction=left: {left};
      direction=right: {right};
    end Protocol
    Evolution:
        direction=right if Environment.guardRoom=room1 and
             Action=left;
        direction=left if Environment.guardRoom=room1 and
             Action=right;
    end Evolution
end Agent


Agent Thief
   Lobsvars = {guardRoom};
   Vars:
      room: {outside, room0, room1, room2};
   end Vars
   Actions = {left, right, wait, steal, exit};
   Protocol:
      room=room2 and Environment.guardRoom=room2: {left,
           wait, exit};
      room=room2 and (Environment.guardRoom=room0 or
           Environment.guardRoom=room2): {left, wait, steal,
           exit};
      room=room0 : {right, wait};
      room=room1 : {left, right, wait};
      Other: {left, right, wait};
   end Protocol
   Evolution:
      room = room0 if (room=room1 and Action=left);
      room = room1 if (room=room0 and Action=right) or
           (room=room2 and Action=left);
      room = room2 if (room=room1 and Action=right);
      room = outside if (room=room2 and Action=exit);
   end Evolution
end Agent


Evaluation
    caught if (Environment.painting = false) and
         (Environment.guardRoom = Thief.room);
    stolen if Environment.painting = false;
    escaped if Thief.room = outside;
end Evaluation

InitStates
   Environment.guardRoom = room0 and
   Guard.direction = right and
   Thief.room = room0 and
   Environment.painting = true;
end InitStates

Groups
   gG = {Guard};
   gT = {Thief};
end Groups

Formulae
   -- test formulae
   <gT> G !caught;
   <gT> F (stolen and escaped);
   --!<gT> X stolen;
   --!<gT> G escaped;
   --!<gT> X caught;

   -- exercise
   <gT> ((!caught) U (stolen and escaped));
end Formulae
```

C INTERPRETED SYSTEM IN ISPL

```
-- INFOMLSAI 2021 an interpreted system designed by
-- ANNELINE DAGGELINCKX, MATTHIJS KEMP and OTTO MATTAS

Semantics=SingleAssignment;

Agent Environment
   Vars:
      guardRoom: {room0, room1, room2};
      painting: boolean;
   end Vars
   Actions = { none };
   Protocol:
      Other: { none };
    end Protocol
   Evolution:
      guardRoom = room0 if (guardRoom=room1 and
           Guard.Action=left);
      guardRoom = room1 if (guardRoom=room0 and
           Guard.Action=right) or (guardRoom=room2 and
           Guard.Action=left);
      guardRoom = room2 if (guardRoom=room1 and
           Guard.Action=right);

      painting = false if (Thief.Action=steal);
   end Evolution
end Agent


Agent Guard
   Lobsvars = {guardRoom};
   Vars:
      dummy: {var};
   end Vars
    Actions = {left, right, wait};
    Protocol:
      Environment.guardRoom=room0 : {right, wait};
      Environment.guardRoom=room2 : {left, wait};
      Other : {left, right, wait};
    end Protocol
    Evolution:
        dummy = var if dummy = var;
    end Evolution
end Agent


Agent Thief
   Lobsvars = {guardRoom};
   Vars:
      room: {outside, room0, room1, room2};
   end Vars
   Actions = {left, right, wait, steal, exit};
   Protocol:
      room=room2 and Environment.guardRoom=room2: {left,
           wait, exit};
      room=room2 and (Environment.guardRoom=room0 or
           Environment.guardRoom=room2): {left, wait, steal,
           exit};
      room=room0 : {right, wait};
      room=room1 : {left, right, wait};
      Other: {left, right, wait};
   end Protocol
   Evolution:
      room = room0 if (room=room1 and Action=left);
      room = room1 if (room=room0 and Action=right) or
           (room=room2 and Action=left);
      room = room2 if (room=room1 and Action=right);
      room = outside if (room=room2 and Action=exit);
   end Evolution
end Agent


Evaluation
    caught if (Environment.painting = false) and
         (Environment.guardRoom = Thief.room);
    stolen if Environment.painting = false;
    escaped if Thief.room = outside;
end Evaluation

InitStates
   Environment.guardRoom = room0 and
   Thief.room = room0 and
   Environment.painting = true;
end InitStates

Groups
   gG = {Guard};
   gT = {Thief};
   gGT = {Guard, Thief};
end Groups

Formulae
   -- exercise 7.3:
   <gG> G(!stolen);

   -- exercise 7.4:
   -- <gGT> (!caught U (stolen and escaped));
   -- This last formula gives a segmentation fault but is
        equivalent to
   E (!caught U (stolen and escaped));
end Formulae
```

D INTERPRETED SYSTEM IN ISPL

```
-- INFOMLSAI 2021 an interpreted system designed by
-- ANNELINE DAGGELINCKX, MATTHIJS KEMP and OTTO MATTAS

Semantics=SingleAssignment;

Agent bert
   Vars:
      x: boolean;
      y: boolean;
      z: boolean;
   end Vars
   Actions = {switch_x, switch_y, switch_z};
   Protocol:
      Other: {switch_x, switch_y, switch_z};
   end Protocol
   Evolution:
      x = true if x = false and Action = switch_x;
      x = false if x = true and Action = switch_x;
      y = true if y = false and Action = switch_y;
      y = false if y = true and Action = switch_y;
      z = true if z = false and Action = switch_z;
      z = false if z = true and Action = switch_z;
   end Evolution
end Agent

-- the remainder of the file is written such that the
     program is runnable
Evaluation
   x_on if bert.x = true;
   y_on if bert.y = true;
   z_on if bert.z = true;
end Evaluation

InitStates
   bert.x = false and
   bert.y = false and
   bert.z = false;
end InitStates

Formulae
   EF (x_on and y_on and z_on);
end Formulae
```