Two-player Game Structures for Service Composition, Synthesis and Generalized Planning

Fabio Patrizi
SAPIENZA – Università di Roma
patrizi@dis.uniroma1.it
www.dis.uniroma1.it/~patrizi
Solving Composition Problems

• Service Composition problems can be solved using a variety of approaches, e.g.,:
  
  – PDL-based
    [Berardi, Calvanese, De Giacomo, Lenzerini, Mecella@ICSOC03]
  – Direct simulation computation
    [Ströder, Pagnucco@IJCAI09]
  – LTL synthesis
    [Sardina, DeGiacomo@ICAPS08; P@Phd09]

• 2-GS: powerful framework to capture and solve all above
Two-player Game Structures (2-GS)

• Inspired by game structures for LTL synthesis
  [Piterman,Pnueli,Sa’ar@VMCAI06; Alur,Hentzinger,Kupferman@JACM-02]

• Model the rules of a game (e.g., Chess) between players:
  – Controller (the good)
  – Environment (the bad)

• With the game at hand, we can:
  – define a problem (e.g., can we checkmate from a starting situation?)
  – (Try to) solve the problem
2-GS: Definition

\[ G = \langle \mathcal{X}, \mathcal{Y}, \text{start}, \rho_e, \rho_c \rangle, \text{ where:} \]

- \( \mathcal{X} \): set of environment (uncontrolled) variables \( x_1, \ldots, x_n \), ranging over \( X = X_1 \times \ldots \times X_n \)
- \( \mathcal{Y} \): set of controller (controlled) variables \( y_1, \ldots, y_m \), ranging over \( Y = Y_1 \times \ldots \times Y_m \)
- \( \text{start} = \langle x_0, y_0 \rangle \in X \times Y \) is the initial game state
- \( \rho_e \subseteq X \times Y \times X \) is the environment transition relation
- \( \rho_c \subseteq X \times Y \times X \times Y \) is the controller transition relation
2-GS: Rounds

- Each round consists of an environment move and a controller reply.
- Moves and replies must be compliant with $\rho_e$ and $\rho_c$.
2-GS MC Example: TIC-TAC-TOE

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $\mathcal{X} = \{x_{A1}, \ldots, x_{A3}, \ldots, x_{C1}, x_{C3}\}$: propositional
- $\mathcal{Y} = \{y_{A1}, \ldots, y_{A3}, \ldots, y_{C1}, y_{C3}\}$: propositional
- Start: all variables are initially false
- $\rho_e$: assign true exactly one $\mathcal{X}$ variable $x_{ij}$ s.t. $y_{ij}$ is false
- $\rho_c$: assign true exactly one $\mathcal{Y}$ variable $y_{ij}$ s.t. $x_{ij}$ is false
Example(2)

• 2-GSs capture (ND) planning domains:
  – The controller executes an action
  – The environment chooses the outcome

• $\rho_c$ accounts for preconditions
• $\rho_e$ accounts for (ND) effects
Goals for 2GS

• When does a player win G?
• It depends on the goal
• In planning, we have reachability goals, e.g.:
  – checkmate the opponent’s king
• In general, we can define complex goals, e.g.:
  – The controller can always reach a state where the coin can be tossed
μ-calculus over 2-GS

• To define goals, we use a variant of the μ-calculus [Emerson96], whose formulae are:
  – atoms of the form $x_i = x$ or $y_i = y$
  – $\bigcirc \Psi$ (next), if $\Psi$ is a formula
  – $\mu Z.\Psi$ (least fixpoint), if $\Psi$ is a formula
  – $\nu Z.\Psi$ (greatest fixpoint), if $\Psi$ is a formula
  – Boolean combinations of above formulae
\[\mu\text{-calculus over 2-GS (2)}\]

For complete semantics, see [Emerson96]

- Key operator *next*

\[
\langle x,y \rangle \models \circ \Psi \iff \\
\exists x'. \rho_e (x,y,x') \wedge \\
\forall x'. \rho_e (x,y,x') \rightarrow \exists y'. \rho_c (x,y,x',y') \text{ s.t. } \langle x',y' \rangle \models \Psi
\]

(\text{Player controller is able to force the game to reach, in one step, a state where } \Psi \text{ holds, no matter how the environment moves})
Defining Goals

• Given a ($\mu$-calculus) goal formula $\phi$, player controller wins iff $\langle x_0, y_0 \rangle \models \phi$

• We use particular goal patterns

  $\lozenge \phi \doteq \mu Z. \phi \lor \lozenge Z$ (C. can force the game to eventually reach $\phi$)

  $\square \phi \doteq \nu Z. \phi \land \lozenge Z$ (C. can force the game to always satisfy $\phi$)

  $\square \lozenge \phi$ (C. can force the game to always satisfy $\lozenge \phi$)
2-GS Model Checking

- **DEF:** Given a 2-GS $G$ and a goal formula $\phi$, $G \models \phi$ iff $\langle x_0, y_0 \rangle \models \phi$

- The **MC problem** requires to check if, given $G$ and $\phi$, $G \models \phi$

- If so, the controller has a *strategy* to enforce $\phi$, (no matter how the environment plays)

- **Strategy:** function of histories

- We are not only interested in the checking problem, but in computing the strategy
2-GS Model Checking (2)

- The computational cost of 2-GS MC is \( O((|G| \cdot |\phi|)^k) \), where:
  - \( |G| = |S_G| + |\rho_e| + |\rho_c| \)
  - \( k \) is the number of fixpoint nestings in \( \phi \)
Conditional Planning with 2-GS (2)

Domain: Tossing a coin
Predicates: inHand, head
Actions:
  toss(pre: inHand,
       eff: oneof(head,¬head) and ¬inHand
  )
  turn(
    pre: ¬inHand
    eff: (when (head)(¬head) and when (¬head)(head))
  )
  nop()

Init: inHand
Goal: head
Coin Tossing as a 2-GS

GS:

• $X = \{\text{inHand}, \text{head}\}$: propositional
• $Y = \{\text{act}\}$, over: $\{\text{toss, turn, nop, init}\}$
• $\text{start}$: $\text{inHand} = \text{head} = \bot$; $\text{act} = \text{init}$
• $\rho_e$: selects action effects (according to current $\text{act}$)
• $\rho_c$: chooses next action, among those executable in current state
• Special action $\text{init}$ for initialization only

Goal formula: $\Psi = \Diamond (\text{head} = \top)$
Patrizi, F., Two-Player Game Structures for Service Composition, Synthesis and Generalized Planning
Solution Approach

- We applied a MC algorithm for μ-calc
- Time cost: \( O(|2^P| |A| + |\rho|) \)
- During the check, we saved additional information to extract a witness, i.e., a strategy for the controller
- A strategy for \( \Diamond (\text{head} = \top) \) corresponds, in fact, to a conditional plan
- Explicit state manipulation not required: symbolic approaches (e.g., BDD-based) can be used
Solving Multi-Target Composition using 2-GS

• Encoding similar to Planning Programs

• Player Environment features:
  – the execution of target programs
  – (A target program is advanced only after the controller declares its request fulfilled)

• Player Controller:
  – delegates, at each step, an action requested by some target, to some available service able to execute it
  – At some point, based on action outcomes, declares some targets fulfilled
  – (When no more target requests are pending, at least one target must be declared satisfied, so as to get a new request)
Solving Multi-Target Composition using 2-GS (2)

- \( \mathcal{X} = \{s_1, \ldots, s_n, t_1, \ldots, t_m\} \)
  - \( s \): state of available service
  - \( t \): requested transition
- \( \mathcal{Y} = \{act, ser, full_1, \ldots, full_m\} \)
  - \( act \): action to execute
  - \( ser \): delegated service
  - \( full_i \): fulfilled?
- \( \text{start: } act = ser = \text{init}; full_i = \bot \) (all state initial)
- \( \rho_e \): selects action effects, according to current \( act \) and \( ser \), and advances the target, according to \( full_i \)
- \( \rho_e \): chooses next action \( act \), according to \( t_i \), delegates to \( ser \), and, when needed, declares targets’ fulfillment, assigning \( full_i \)
- (Special action \( \text{init} \) for initialization only)

Goal formula: \( \Psi = \square (\diamond full_1 = \top) \land \ldots \land \square (\diamond full_m = \top) \)
Solution

• Still an exponential bound
• Optimal as the problem is EXPTIME-complete
Solving Agent Planning Programs using 2-GS

Domain: TVworld

Predicates: on, broken, mute

Actions:
- `muteTV`:
  ```
  pre: on
  eff: when(mute)(¬mute) \land when(¬mute)(mute))
  ```
- `switchOn`:
  ```
  pre: off \land ¬broken
  eff: on
  ```
- `switchOff`:
  ```
  pre: on
  eff: off
  ```
- `throwTV`:
  ```
  eff: off \land broken
  ```

Init: off \land ¬broken \land ¬mute
Solving Agent Planning Programs using 2-GS (2)

• Player **Environment** features:
  – the **planning domain**
  – the target service **evolution**, i.e., its requests
  – (The target service advances only when its current request is fulfilled)

• Player **Controller**:
  – selects, at each step, the actions needed to **fulfill current request**
  – **announces** current request fulfillment to the environment (which advances the target)
Solving Agent Planning Programs using 2-GS (3)

TV Domain:
• \( \mathcal{X} = \{\text{on}, \text{broken}, \text{mute}, \text{tr}\} \)
• \( \mathcal{Y} = \{\text{act}, \text{last}\} \)
• In the start state, \( \text{last} = \bot \) (special action \text{init} also used)
• \( \rho_e: \)
  – according to current \text{act} changes \{\text{on}, \text{broken}, \text{mute}\}
  – If \( \text{last} = \top \), changes \text{tr} according to the target service
• \( \rho_c: \)
  – chooses next action, among those executable in current state
  – sets \( \text{last} = \top \) only if current \text{tr} is actually realized
• Goal formula: \( \Psi = \Box \Diamond (\text{last} = \top) \)
Example
Observation

• Each target transition is realized by a (conditional) plan
• However, plans cannot be computed as usually done in planning
• Realizability of possible future transitions must be guaranteed
• TV cannot be switched off by throwing it because this prevents future requests for on
Solution

• Again, we use a MC algorithm (in fact, the same as before) for $\mu$-calc

• (This time, two non-nested fixpoint computations are needed, for $\Diamond$ and $\Box$)

• Time cost: $O(2^p \cdot (|\rho| + |\delta|))$

• Optimal, as the problem is EXPTIME-complete
Agent Planning Programs and Services

• How planning programs are related to services?

• Given a set of available services, we compose a high-level procedure, instead of a new service
Generalized Planning w/ loops under strong fairness constraints

• 2-GS and the μ-calc are also useful to tackle generalized forms of planning

• Sample ND Domain:
Generalized Planning w/ loops under strong fairness constraints (2)

• **Conditional Plan:**
  Acyclic plan that reaches a goal, e.g., “head”, no matter how nondeterminism is resolved at runtime

• **Strong Cyclic Plan** [Cimatti, Pistore, Roveri, Traverso-AI 03]:
  Cyclic plan that reaches a goal, under the assumption that loops iterate only a finite (though unbounded) number of times
Fairness Constraints

Strong cyclic plans work under a specific fairness assumption, required to hold for all loops.

We want to be able to

1. Asserting explicitly general fairness constraints on domain evolutions
2. Finding plans that work under such constraints
Fairness Constraints (2)

\[ \square \Diamond (\text{act=toss}) \rightarrow \square \Diamond (\text{head}) \]
Generalized Planning w/ loops under strong fairness constraints (3)

- Can be reduced to MC a 2GS (not done, yet)
- (Currently: reduction to LTL synthesis problem [DeGiacomo,P,Sardina@KR10])
- The problem is EXPTIME-complete
Conclusion

• “Local” MC techniques can be useful for optimization?