A Generic Framework and Solver for Synthesizing Finite-State Controllers

Toby Hu (University of Toronto, Canada)
Giuseppe De Giacomo (SAPIENZA Università di Roma, Italy)

AAAI'11 Workshop on Generalized Planning,
San Francisco, California
August 8, 2011

Introduction

Finite-state controllers (FSC) are widely used in AI

- Achieve reachability goals for planning with incomplete knowledge
- Control long-running agents to satisfy temporally-extended goals
- ...

Introduction

Finite-state controllers (FSC) are widely used in AI

- Achieve reachability goals for planning with incomplete knowledge
- Control long-running agents to satisfy temporally-extended goals
- ...

FSAPLANNER partly solves the first type of problems, can we generalize its idea for FSC synthesis in general?

Introduction

Finite-state controllers (FSC) are widely used in AI

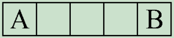
- Achieve reachability goals for planning with incomplete knowledge
- Control long-running agents to satisfy temporally-extended goals
- ...

FSAPLANNER partly solves the first type of problems, can we generalize its idea for FSC synthesis in general?

In this work, we present a generic framework and solver for synthesizing FSCs, and show its application in

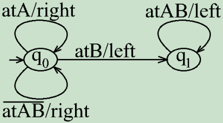
- "Generalized planning" [Bonet *et al.* 1999; Pralet *et al.* 2010]
- Service composition [Calvanese *et al.* 2008]
- Planning programs [De Giacomo *et al.* 2010]

Example 1: Generalized Planning



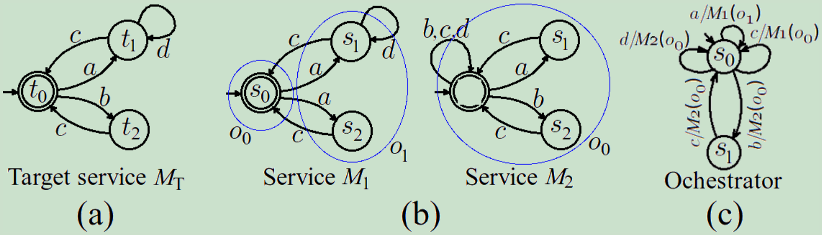
(a)

(b)

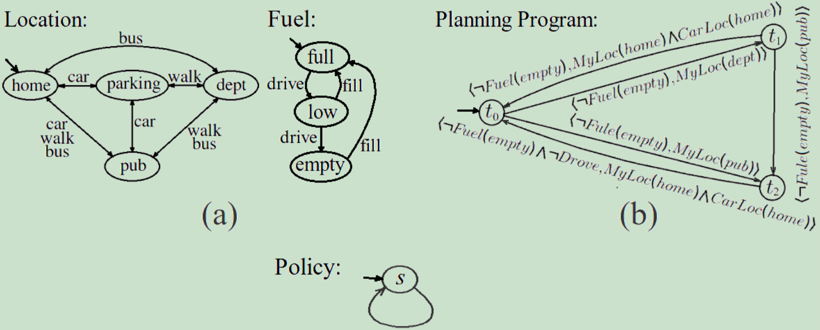


(c)

Example 2: Service Composition



Example 3: Planning Programs



$\langle\{\cdots, MyLoc(home)\}, t_0 \rightarrow t_1\rangle$ / $goByBus(dept)$
$\langle\{\cdots, MyLoc(home)\}, t_0 \rightarrow t_2\rangle$ / $walk(pub)$
$\langle\{\cdots, MyLoc(dept)\}, t_1 \rightarrow t_0\rangle$ / $goByBus(home)$
$\langle\{\cdots, MyLoc(dept)\}, t_1 \rightarrow t_2\rangle$ / $goByBus(pub)$
$\langle\{\cdots, MyLoc(pub)\}, t_2 \rightarrow t_0\rangle$ / $walk(home)$

(c)

A Generic Framework for Controller Synthesis

Controller Synthesis: given a dynamic environment and a behavior specification, find a finite-state controller so that the behavior is realized.

A Generic Framework for Controller Synthesis

Controller Synthesis: given a dynamic environment and a behavior specification, find a finite-state controller so that the behavior is realized.

A *dynamic environment* is a tuple $\mathcal{E} = \langle \mathcal{A}, \mathcal{O}, \mathcal{S}, \mathcal{I}, \Delta, \Omega \rangle$, where

- $\mathcal{A}$ is a finite set of actions,
- $\mathcal{O}$ is a finite set of observations,
- $\mathcal{S}$ is a finite set of world states (the state space),
- $\mathcal{I} \subseteq \mathcal{S}$ is a set of possible initial states,
- $\Delta \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation, and
- $\Omega : \mathcal{S} \to \mathcal{O}$ is the observation function.

We use the notation $s \xrightarrow{a} s'$ to denote $\langle s, a, s' \rangle \in \Delta$.

A Generic Framework for Controller Synthesis

Controller Synthesis: given a dynamic environment and a behavior specification, find a finite-state controller so that the behavior is realized.

An *(N-bounded) finite-state controller* for an environment $\mathcal{E} = \langle \mathcal{A}, \mathcal{O}, \mathcal{S}, \mathcal{I}, \Delta, \Omega \rangle$ is a tuple $\mathcal{C} = \langle Q, q_0, T \rangle$, where

- $Q = \{1, \ldots, N\}$ is the finite set of control states,
- $q_0 = 1$ is the initial control state,
- $T : \langle Q \times \mathcal{O} \rangle \to \langle Q \times \mathcal{A} \rangle$ is the transition function.

We use $q \xrightarrow{o/a} q'$ to denote $T(q, o) = \langle q', a \rangle$.

A Generic Framework for Controller Synthesis

Controller Synthesis: given a dynamic environment and a behavior specification, find a finite-state controller so that the behavior is realized.

An *execution history* of controller $\mathcal{C}$ in environment $\mathcal{E}$ is a finite sequence $h = \langle q_0, s_0\rangle, \langle q_1, s_1\rangle, \cdots, \langle q_n, s_n\rangle$, such that there is a sequence of actions $r = a_1 a_2 \cdots a_n$, satisfying

- $s_0 \in \mathcal{I}$ (recall that $q_0 = 1$ by definition),
- $q_i \xrightarrow{\Omega(s_i)/a_{i+1}} q_{i+1}$ and $s_i \xrightarrow{a_{i+1}} s_{i+1}$.

A *controller specification* for $\mathcal{E}$ and control states Q is a function $\beta : (Q \times \mathcal{S})^* \rightarrow \{\text{true}, \text{false}, \text{unknown}\}$

A Generic Framework for Controller Synthesis

Controller Synthesis: given a dynamic environment and a behavior specification, find a finite-state controller so that the behavior is realized.

An *execution history* of controller $\mathcal{C}$ in environment $\mathcal{E}$ is a finite sequence $h = \langle q_0, s_0 \rangle, \langle q_1, s_1 \rangle, \cdots, \langle q_n, s_n \rangle$, such that there is a sequence of actions $r = a_1 a_2 \cdots a_n$, satisfying

- $s_0 \in \mathcal{I}$ (recall that $q_0 = 1$ by definition),
- $q_i \xrightarrow{\Omega(s_i)/a_{i+1}} q_{i+1}$ and $s_i \xrightarrow{a_{i+1}} s_{i+1}$.

A *controller specification* for $\mathcal{E}$ and control states Q is a function $\beta : (Q \times \mathcal{S})^* \to \{\text{true}, \text{false}, \text{unknown}\}$ satisfying the following condition: if a history h' contains two identical configurations $\langle q_i, s_i \rangle$ and $\langle q_j, s_j \rangle$, then at least one of its prefix h satisfy $\beta(h) \in \{\text{true}, \text{false}\}$.

A Generic Algorithm for Controller Synthesis

$\mathcal{C} =$ **synthesize**$_{\mathcal{E},N}(\mathcal{I})$
    return AND_step$_{\mathcal{E},N}(\emptyset, 1, \mathcal{I}, \emptyset)$;

AND_step$_{\mathcal{E},N}(C, q, S, h)$
    for each $s \in S$
        $C :=$ **OR_step**$_{\mathcal{E},N}(C, q, s, h \cdot \langle q, s \rangle)$;
    return C;

OR_step$_{\mathcal{E},N}(C, q, s, h)$
    if $\beta(h) = \mathsf{true}$ **return** C;
    else if $\beta(h) = \mathsf{false}$ **fail**;
    else if $(q \xrightarrow{\Omega(s)/a} q') \in C$
        $S' := \{s' \mid s \xrightarrow{a} s'\}$;
        return AND_step$_{\mathcal{E},N}(C, q', S', h)$;
    else
        NON-DETERMINISTICALLY CHOOSE
            $a \in \mathcal{A}$ and $1 \leq q' \leq N$;
        $S' := \{s' \mid s \xrightarrow{a} s'\}$;
        $C' := C \cup \{q \xrightarrow{\Omega(s)/a} q'\}$;
        return AND_step$_{\mathcal{E},N}(C', q', S', h)$;

A Generic Algorithm for Controller Synthesis

This algorithm strategically enumerates all valid controllers with up to N states

A Generic Algorithm for Controller Synthesis

This algorithm strategically enumerates all valid controllers with up to N states by avoiding

- controllers with unreachable states, and
- isomorphic (identical by state renaming) controllers with a simple state ordering.

A Generic Algorithm for Controller Synthesis

This algorithm strategically enumerates all valid controllers with up to N states by avoiding

- controllers with unreachable states, and
- isomorphic (identical by state renaming) controllers with a simple state ordering.

Theorem (Soundness and Completeness)

Given environment $\mathcal{E}$ with initial states $\mathcal{I}$, and a behavior specification β, $\mathcal{C} = \textbf{synthesize}_{\mathcal{E},N}(\mathcal{I})$ iff $\mathcal{C}$ is an N-bounded finite-state controller in $\mathcal{E}$ that satisfies β, up to isomorphism.

Instantiations: Generalized Planning

Behavior specification:

$$\beta(h) = \begin{cases} \mathsf{true} & \text{if the last state in } h \text{ satisfies the goal } G; \\ \mathsf{false} & \text{if } h \text{ cannot be extended or} \\ & \text{has two identical configurations;} \\ \mathsf{unknown} & \text{otherwise.} \end{cases}$$

Instantiations: Generalized Planning

Behavior specification:

$$\beta(h) = \begin{cases} \mathsf{true} & \text{if the last state in } h \text{ satisfies the goal } G; \\ \mathsf{false} & \text{if } h \text{ cannot be extended or} \\ & \text{has two identical configurations;} \\ \mathsf{unknown} & \text{otherwise.} \end{cases}$$

We implemented the adapted solver in SWI-Prolog, and obtained promising preliminary experimental results.

Instantiations: Generalized Planning

		BPG	Dyncode		Our solver	
Problem	N	Solve	Solve	Proof	Solve	Proof
Hall-A 1×4	2	0.0	0.01	0.02	0.01	0.0
Hall-A 4×4	4	5730.5	0.26	2.35	0.21	1.86
Hall-R 1×4	1	0.0	0.01	0	0.01	0
Hall-R 4×4	1	0.0	0.02	0	0.01	0
Prize-A 4×4	1	0.0	0.02	0	0.01	0
Corner-A 4×4	1	0.1	0.02	0	0.01	0
Prize-R 3×3	2	0.1	0.03	0.03	0.04	0.01
Prize-R 5×5	3	2.7	2.37	0.97	2.71	1.3
Corner-R 2×2	1	0.0	0.01	0	0.01	0
Corner-R 5×5	1	1.6	0.02	0	0.01	0
Prize-T 3×3	1	0.1	0.05	0	0.01	0
Prize-T 5×5	1	0.3	0.34	0	0.02	0
Blocks 6	2	0.8	0.02	0.02	0.02	0.0
Blocks 20	2	34.8	0.04	0.02	0.02	0.0
Visual-M $(8, 5)$	2	1289.5	3.59	0.27	0.02	0.0
Gripper $(3, 5)$	2	4996.1	0.06	0.02	0.01	0.0

Data obtained on different hardware. Performance not to compare.

Instantiations: Service Composition

The state space is the cross-product of the target and service states together with the requested action.

Instantiations: Service Composition

The state space is the cross-product of the target and service states together with the requested action.

The behavior specification:

$$\beta(h) = \begin{cases} \mathsf{true} & \text{if } h \text{ contains two identical configurations;} \\ \mathsf{false} & \text{if the last requested action cannot be provided by any service, or the target is at a final state but some of the services are not;} \\ \mathsf{unknown} & \text{otherwise.} \end{cases}$$

Instantiations: Service Composition

The state space is the cross-product of the target and service states together with the requested action.

The behavior specification:

$$\beta(h) = \begin{cases} \mathsf{true} & \text{if } h \text{ contains two identical configurations;} \\ \mathsf{false} & \text{if the last requested action cannot be} \\ & \text{provided by any service, or the target is at a} \\ & \text{final state but some of the services are not;} \\ \mathsf{unknown} & \text{otherwise.} \end{cases}$$

We experimented the adapted solver on 18 benchmark problems that require the composition of target services of 2–4 states from 2–5 available services ranging from 2 to 10 states.
All problems are either solved or proved unsolvable within less than 0.01 second.

Instantiations: Planning Programs

The state space is the cross product of the planning domain's state space and the goal transitions in the planning program.

Instantiations: Planning Programs

The state space is the cross product of the planning domain's state space and the goal transitions in the planning program.

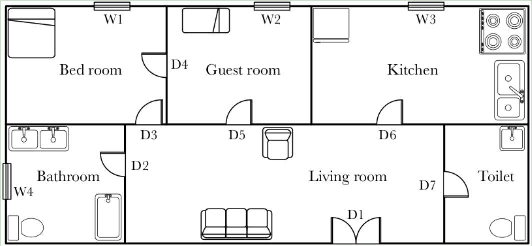
The behavior specification:

$$\beta(h) = \begin{cases} \mathsf{false} & \text{if the maintenance goal is violated in } h\text{, or} \\ & h \text{ cannot be extended, or} \\ & h \text{ contains two identical configurations} \\ & \text{while achieving a goal transition;} \\ \mathsf{true} & \text{if the last goal transition in } h \text{ has been achieved} \\ & \text{in a proper prefix of } h\text{;} \\ \mathsf{unknown} & \text{otherwise.} \end{cases}$$

Instantiations: Planning Programs

The adapted solver finds a solution to the researcher's world example in 0.04 second.

[De Giacomo *et al.* 2011] reports the application of this method to a practical smart-home application, where a policy is found in less than 4 minutes for a complex, 5-state planning program in the following environment:



Summary

Conclusions

- We presented a generic framework and solver for synthesizing finite-state controllers.
- Preliminary experimental results show the effectiveness of our approach in three diverse types of synthesis problems.

Future work:

- More detailed experimental evaluation
- Real-world applications
- Explore LTL-synthesis