Controlling Golog Programs against MTL Constraints

Till Hofmann[1], Stefan Schupp[2], Gerhard Lakemeyer[1]

[1]Knowledge-Based Systems Group
RWTH Aachen University

[2]Cyber-Physical Systems Group
TU Wien

At a Glance

- Golog is a high-level programming language for robots
- It allows abstract specification of agent behavior and supports nondeterministic operators

→ Golog program can be seen as a *partial behavior specification*

- On an actual robot, additional *platform requirements* need to be considered, e.g., hardware initialization, timing constraints → model as MTL constraints
- Some actions are not under the agent's control, e.g., action duration, exogenous events

⇒ Goal: synthesize a controller that executes the program while satisfying constraints

Foundations: *t-ESG* Basic Action Theories

- *t-ESG* is a modal variant of the situation calculus based on *ES* and *ESG*
- Extends *ESG* by *timed traces* and *clocks*
- Clocks can be compared against rational constants and may be reset by an action
- A *trace* is a sequence of action-time pairs,
  e.g., $\langle (start(bootCamera), 0), (end(bootCamera), 1)\rangle$
- Trace formulas akin to MTL allow temporal constraints with metric time
- Transition semantics defines the evolution of program configurations $\langle z, \rho\rangle$,
  e.g., $\langle z, \rho\rangle \xrightarrow{w} \langle z \cdot (a, t), \rho'\rangle$ is the execution of action a at time t
- A *basic action theory* defines the intial situation and the agent's actions
- Initial situation:

$$\neg CamOn \wedge \neg Grasping \wedge \forall a \neg Perf(a)$$

- Precondition axiom:

$$\begin{aligned}\Box \,\mathrm{Poss}(a) \equiv\ & \exists l, p{:}o.\ a = start(grasp(l, p))\\ & \vee a = start(bootCamera) \wedge \neg CamOn\\ & \vee \exists p.\ a = end(p) \wedge Perf(p)\end{aligned}$$

- Clock constraint axiom:

$$\begin{aligned}\Box \,\mathrm{g}(a) \equiv\ & \exists p.\ a = start(p)\\ & \vee \exists l, o.\ a = end(grasp(l, o)) \ \wedge\ c(grasp(l, o)) = 2\\ & \vee\ a = end(bootCamera) \wedge c(bootCamera) = 1\end{aligned}$$

- Successor state axioms:

$$\begin{aligned}\Box[a]\, CamOn &\equiv a = end(bootCamera) \vee CamOn\\ \Box[a]\, Grasping &\equiv a = start(grasp) \vee Grasping \wedge a \neq end(grasp)\\ \Box[a]\, Perf(p) &\equiv a = start(p) \vee Perf(p) \wedge a \neq end(p)\end{aligned}$$

- Clock reset axioms:

$$\Box[a]\,\mathrm{reset}(c) \equiv \exists p.\ a = start(p) \wedge c = c(p)$$

- Program:

$$\begin{aligned}\delta_h &:= start(grasp(m_2, o_1)); end(grasp(m_2, o_1))\\ \delta_m &:= start(bootCamera); end(bootCamera)\\ \delta &:= \delta_h \parallel \delta_m\end{aligned}$$

Foundations: MTL and Alternating Timed Automata

- MTL: temporal logic with timing constraints on temporal operators:

$$\phi = \top \mathbf{U}\, (\neg CamOn \wedge Grasping) \vee \top \mathbf{U}\, (\neg CamOn \wedge \top \mathbf{U}_{[0,2]}\, Grasping)$$

- An MTL formula can be translated to an *alternating timed automaton* (ATA) to check satisfaction (Ouaknine and Worrell 2005)
- The automaton tracks the unsatisfied parts of the formula
- Each location is a sub-formula of ϕ: $L = \{l_0, \phi_1, \phi_2, \phi_3\}$, where

$$\begin{aligned}\phi_1 &:= \top \mathbf{U}\, (\neg CamOn \wedge Grasping)\\ \phi_2 &:= \top \mathbf{U}\, (\neg CamOn \wedge \top \mathbf{U}_{[0,2]}\, Grasping)\\ \phi_3 &:= \top \mathbf{U}_{[0,2]}\, Grasping\end{aligned}$$

- An ATA has a single implicit clock x
- Transitions are defined by a transition function, e.g.,:

	$\{\}$	$\{CamOn\}$	$\{Grasping\}$	$\{CamOn, Grasping\}$
l_0	$\phi_1 \vee \phi_2$	$\phi_1 \vee \phi_2$	$\phi_1 \vee \phi_2$	$\phi_1 \vee \phi_2$
ϕ_1	ϕ_1	ϕ_1	$\top$	ϕ_1
ϕ_2	$\phi_2 \vee x.\phi_3$	ϕ_2	$\phi_2 \vee x.\phi_3$	ϕ_2
ϕ_3	ϕ_3	ϕ_3	$(x \geq 0 \wedge x \leq 2) \vee \phi_3$	$(x \geq 0 \wedge x \leq 2) \vee \phi_3$

The Control Problem

Given a program δ and a partition $A_E \dot\cup A_C$ of all actions in environment and controller actions. A controller CR maps a configuration to a set of timed actions, i.e., $CR(z, \rho) = \{(a_i, t_i)\}_{i \in I}$ such that

(C1) Each selected action is a possible transition of the program, i.e.,
for each i, $\langle z, \rho\rangle \xrightarrow{w} \langle z \cdot (a_i, t_i), \rho_i\rangle$ for some ρ_i;

(C2) The controller must not restrict environment actions, i.e.,
for each $a_e \in A_E$, if $\langle z, \rho\rangle \xrightarrow{w} \langle z \cdot (a_e, t), \rho'\rangle$, then
- $(a_e, t) \in CR(z, \rho)$, or
- there is $a_c \in A_C$ and $t_c < t$ such that $(a_c, t_c) \in CR(z, \rho)$;

(C3) The controller may only terminate in final configurations, i.e.,
$CR(z, \rho) = \{\}$ implies $\langle z, \rho\rangle \in \mathcal{F}^w$.

Given an MTL specification of *undesired behavior* ϕ, the *control problem* is to determine a controller CR such that for each finite program trace ψ: $\psi \models \neg\phi$.

Approach

1. Construct alternating timed automaton for the MTL specification
2. Construct the *synchronous product* of the program and the ATA

→ Resulting LTS is infinitely-branching and has infinite paths

3. *Regionalize* the synchronous product and compute determinization

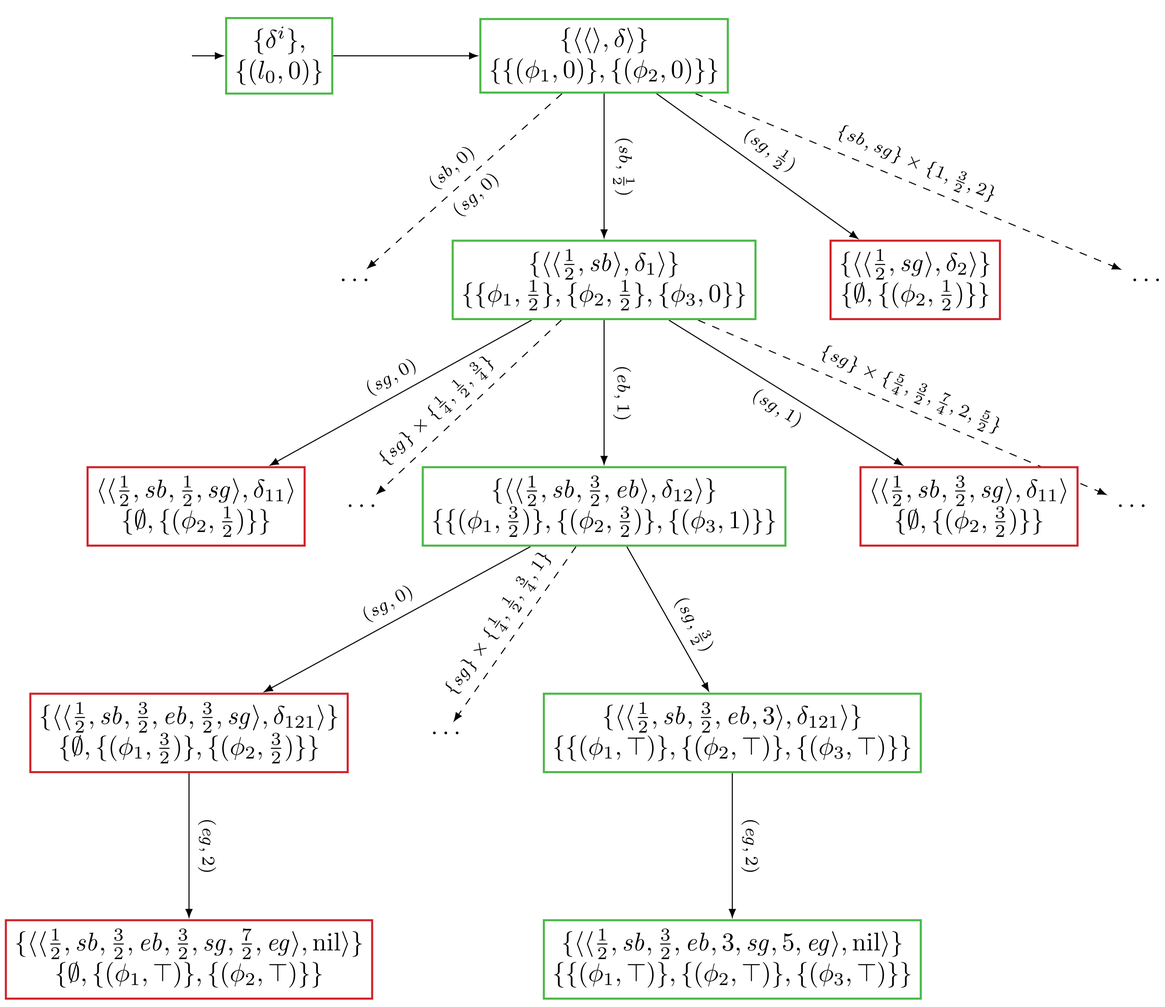
→ Resulting LTS is finitely-branching but has infinite paths

4. Based on a *well-quasi-ordering* on the LTS states, build a tree but prune infinite paths

→ Resulting tree is finite

5. Determine a control strategy on the tree

Example



$$\begin{aligned}sb &:= start(bootCamera) \qquad eb := end(bootCamera)\\ sg &:= start(grasp(m_2, o_1)) \qquad eg := end(grasp(m_2, o_1))\end{aligned}$$

$$\begin{aligned}\delta_1 &:= start(grasp(m_2, o_1)); end(grasp(m_2, o_1)) \parallel end(bootCamera)\\ \delta_2 &:= end(grasp(m_2, o_1)) \parallel start(bootCamera); end(bootCamera)\\ \delta_{11} &:= end(grasp(m_2, o_1)) \parallel end(bootCamera)\\ \delta_{12} &:= start(grasp(m_2, o_1)); end(grasp(m_2, o_1))\\ \delta_{121} &:= end(grasp(m_2, o_1))\end{aligned}$$

- Each node consists of a set of program configurations and a set of ATA configurations
- A leaf node is *bad* if it contains an accepting ATA configuration and a final program configuration, otherwise it is *good*
- An inner node is *good* if the controller can select a set of actions (satisfying C1-C3) such that each successor is good