TLSF Examples of Abstraction-based Control Synthesis

Zexiang Liu

Abstraction-based control synthesis techniques have been developed for many years in order to synthesize controllers formally for control systems such as robotics and vehicles. The synthesized controllers should be able to guarantee that the behavior of the system satisfies the LTL specification specified by users, such as safety requirements for the autonomous vehicles. In the problem setting of control synthesis, the dynamics of the control system is usually known and modeled by a finite transition system (FTS), called the abstraction of the control system. In the physical world, the control systems are usually governed by ordinary differential equations. Many techniques have been developed in this field to obtain the abstraction from the differential equation description of the system, for instance [1, 2]. In this note, we consider the specification of the form

$$\Box A \wedge \Diamond\Box B \wedge \left(\bigwedge_{i\in I} \Box\Diamond R^i\right) \tag{1}$$

where the atomic propositions A, B and R^i describe the *invariance*, *persistence* and *recurrence* properties of desired behavior of a control system. These types of requirements are quite common in the context of control design. For example, a user may want its robot vacuum cleaner to (i) not move out of his apartment (*invariance*) (ii) go to his bedroom and stay there (*persistence*) and (iii) clean different regions in his bedroom in turn (*recurrence*). A synthesis algorithm for this specific type of problems is developed in [2]. In general, this is a synthesis problem which can be solved by full LTL solvers. In this note, we present two examples on abstraction-based control synthesis, which are written in the format of TLSF. We would like to know the performance of existing LTL solvers on this type of problems.

Let's take a toy example to see how the synthesis problem stated above is converted into TLSF format. The abstraction of this example can be easily visualized in Figure 1. There are 4 states $\{s_1, s_2, s_3, s_4\}$ in the nodes and 9 actions $\{a, b, ..., i\}$ marked on each edge.

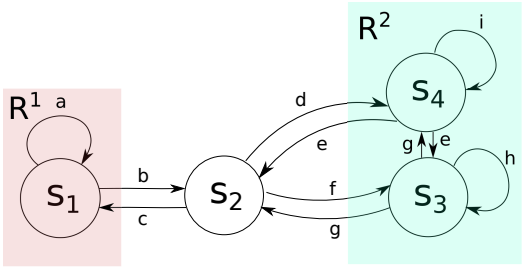

Figure 1: toy example

From the perspective of the two-player game, at each time point, the environment (control system) picks one state, and the system (controller) picks one action. Not all actions are feasible for the system at each state (i.g. only e and i are feasible at s_4). The evolution of the state is governed by the abstraction, i.e. there exists at least one transition in the abstraction from the current state under the current action to the next state. For instance, if at this time the environment and system pick s_4 and e respectively, then the environment can only pick a state in $\{s_2, s_3\}$ at next time. **In the context of the TLSF, the input signal is the state, and the output signal is the action given by the system (controller).**

Note that Figure 1 is a non-deterministic transition system: there are two possible transitions at state s_3 (or s_4) under action g (or e), which models the uncertainty from the environment. To prevent the state from being trapped within $\{s_3, s_4\}$, we assume that the FTS has one progress property[1] $\neg\Diamond\Box(U \wedge G(U))$, where $U = \{d, f\}$ and $G(U) = \{s_3, s_4\}$. It says that the state cannot stay within $\{s_3, s_4\}$ forever by just applying action e or g at each time. The spec of control synthesis for this example is $\Box A \wedge \Diamond\Box B \wedge (\Box\Diamond R^1 \wedge \Box\Diamond R^2)$, where $A = \{s_1, s_2, s_3, s_4\}$, $B = \{s_1, s_2, s_3, s_4\}$, $R^1 = \{s_1\}$ and $R^2 = \{s_3, s_4\}$.

In this note, there are two types of specs: One is the control synthesis spec in the form of (1) which shows the desired behavior of the control system.

[1]The tuple $(U, G(U))$ is called progress group in [2]. This is a kind of fairness assumption. This type of assumptions can be extracted from the properties of the underlying differential equations.

The other one is the TLSF spec converted from the control synthesis spec plus the abstraction. In this toy example, the control synthesis spec is realizable no matter which state the environment picks initially. This is not true in general. Assume that W contains all the feasible initial states, from which the control synthesis spec is realizable[2]. Then to make sure that the TLSF spec is realizable, the initial value of the input signal should be picked from W.

The semantics chosen to form the spec of control synthesis plus the abstraction is *Mealy, Strict*, that is:

$$\phi_i^a \rightarrow (\phi_i^g \wedge (\phi_s^g \; \mathcal{W} \; \neg \phi_s^a) \wedge (\; G \; \phi_s^a \wedge \phi_l^a \rightarrow \phi_l^g)) \tag{2}$$

where $\phi_i^a, \phi_s^a, \phi_l^a$ are the assumptions about the environment, and the $\phi_i^g, \phi_s^g, \phi_l^g$ are guarantees on the behaviors of the control system. The subscripts i, s, l refer to initialization, safety and liveliness assumptions (or guarantees) respectively. The inputs are states picked by the environment and governed by the abstraction and progress properties, and the outputs are actions given by the system (controller). We use enumeration identifiers STATE and ACTION to encode the states and actions. For instance,

```
enum STATE =
        s1: 00
        s2: 01
        s3: 10
        s4: 11
```

Assume that the set of feasible initial states $W \subseteq A$ is known. (We abuse the notation a little bit to denote the sets A, B, R^i by the sets of states satisfying the propositions A, B, R^i respectively.) Input and output are defined as follows.

```
INPUTS {
        STATE S;
}
OUTPUTS {
        ACTION ACT;
}
```

Each section in the MAIN part are defined as follows:

[2] W is called the winning set for the control synthesis spec, and can be computed by the algorithm in [2]

- ϕ_i^a (INITIALLY) enforces that the initial value of the state (input) belongs to W.
- ϕ_s^a (REQUIRE) enforces that the state (input) evolves following the abstraction. For instance, the transition from state s_1 to s_2 under action b written in the TLSF is $(S == s1 \;\&\&\; ACT == b) \rightarrow X\ S == s2$.
- ϕ_l^a (ASSUME) enforce the progress properties of the control system. The LTL expression is in the form of $\bigwedge_{U \in 2^{\mathcal{U}}} \bigwedge_{G \in \mathcal{G}(U)} \square\lozenge((S \notin G) \vee (ACT \notin U))$, where U is a subset of actions and G is a transient set of states under actions in U.
- ϕ_i^g (PRESET) is $true$.
- ϕ_s^g (ASSERT) enforces that (a) feasible actions at each state. For instance, at state s_1, the feasible actions are a and b, that is $S == s_1 \rightarrow (ACT == a \;||\; ACT == b)$. (b) the state always belongs to A.
- ϕ_l^g (GUARANTEE) enforces that (a) $\lozenge\square S \in B$ (b) $\square\lozenge S \in R^i$, for all i.

The attached file **toy_example.tlsf** contains the full spec. Since every state satisfies B, the persistence term $\lozenge\square B$ can be ignored. Then the TLSF spec is in GR(1). It is solved by *slugs* after being transferred to *slugsin* by *syfco*.

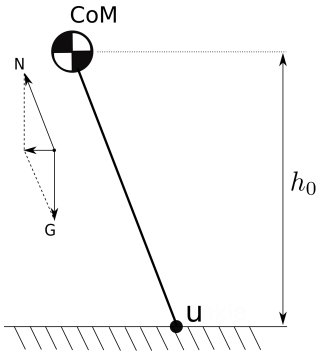

Figure 2: The Linear Inverted Pendulum Model

In the second example, we consider the controller design for a 2D Linear Inverted Pendulum Model (LIPM), which is usually used as a simplified

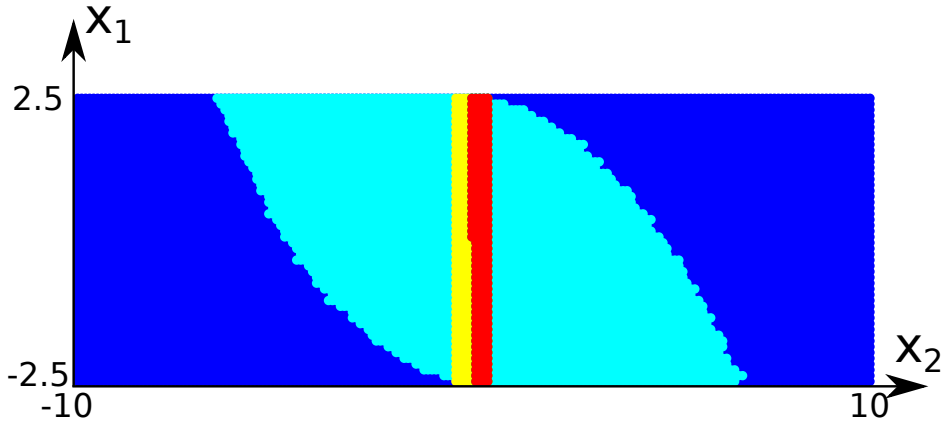

Figure 3: The regions of atomic propositions holds: A (all colored space), B (red+yellow), R^1 (red), R^2 (yellow), the set of feasible initial state W (cyan+red+yellow)

model of walking robots:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ g/h_0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -g/h_0 \end{bmatrix} u \tag{3}$$

where x_1 and x_2 are position and velocity of the center of mass (CoM). g and h are gravitational acceleration and height of the CoM. The state space is $Q = [-2.5, 2.5] \times [-10, 10]$, and the action space is $U = [-3.5, 3.5]$. We discretize the state space and action space uniformly and compute transitions among discrete states under different actions using the method in [1], which gives the abstraction of the dynamical system (3).

The control synthesis spec is still in the form of (1). The sets where A, B, R^i holds are shown in Figure 3. The feasible set of initial states W are computed by the algorithm in [2]. We transfer the spec here into the TLSF format in the same way as in the toy example. The transferred spec is saved in **lipm_wt_B.tlsf**.

Note that the spec described above is not in GR(1). For now we do not know if any existing solver can solve this problem efficiently. If we set $B = Q$, which cancel the persistence term in the spec. Then it becomes GR(1) again, which can be transferred to *slugsin* format by *syfco* and solved by *slugs* within several minutes. The spec file for this setting is **lipm_wo_B.tlsf**.

References

[1] Li, Y., Liu, J., & Ozay, N. (2015). *Computing finite abstractions with robustness margins via local reachable set over-approximation.* IFAC-PapersOnLine, 48(27), 1-6. https://doi.org/10.1016/j.ifacol.2015.11.144

[2] Nilsson, P., Ozay, N., & Liu, J. (2017). *Augmented finite transition systems as abstractions for control synthesis.* Discrete Event Dynamic Systems: Theory and Applications, 27(2), 301?340. https://doi.org/10.1007/s10626-017-0243-z