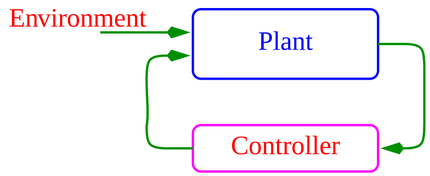
Lectures Outline

- Overview of System Synthesis.
- Fair Discrete Systems and their Computations.
- Model Checking Invariance and feasibility.
- Temporal Testers and general LTL Model Checking.
- Controller Synthesis via Games.
- Synthesis from Recurrence Specifications.
- Synthesis from Reactivity Specifications. – The general case.

The Control Framework

Classical (Continuous Time) Control



Required: A design for a controller which will cause the plant to behave correctly under all possible (appropriately constrained) environments.

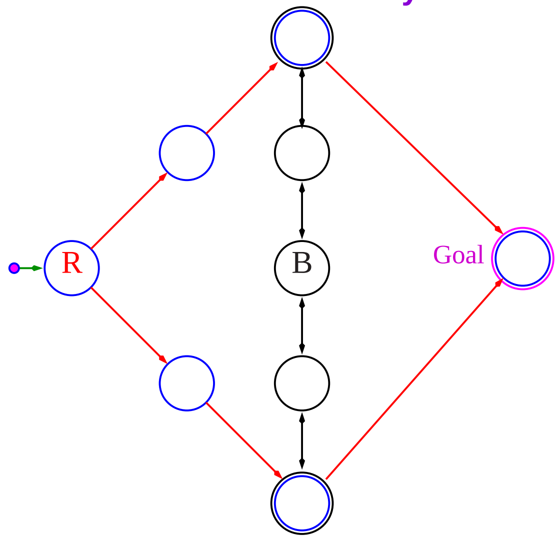
Discrete Event Systems Controller: [Ramadge and Wonham 89]. Given a Plant which describes the possible events and actions. Some of the actions are controllable while the others are not.

Required: Find a strategy for the controllable actions which will maintain a correct behavior against all possible adversary moves. The strategy is obtained by pruning some controllable transitions.

Application to Reactive Module Synthesis: [PR88], [ALW89] — The Plant represents all possible actions. Module actions are controllable. Environment actions are uncontrollable.

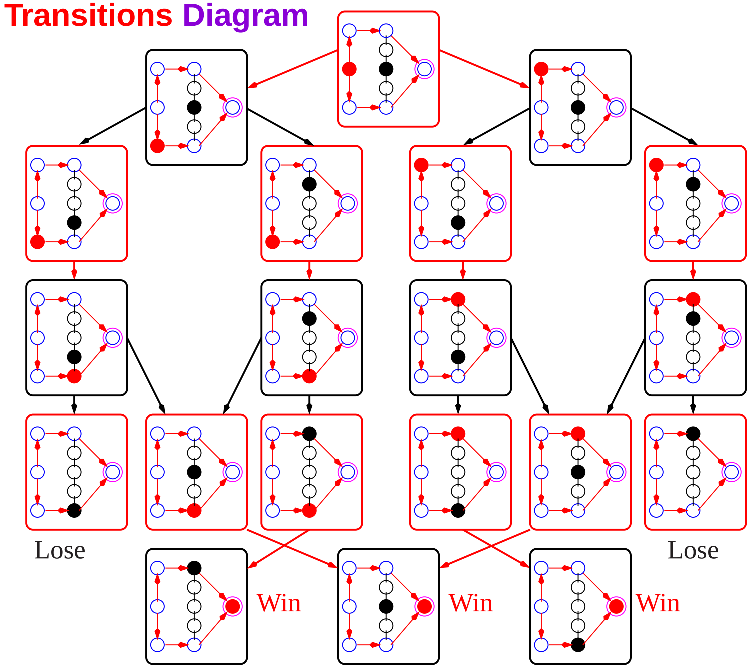
Required: Find a strategy for the controllable actions which will maintain a temporal specification against all possible adversary moves. Derive a program from this strategy. View as a two-persons game.

The Runner Blocker System



The runner R tries to reach the goal. The blocker B tries to intercept and stop the runner.

State Transitions Diagram



Is the Goal Reachable?

All of our algorithms will be computing sets of states out of the state-transition diagram. Let $\|win\|$ denote the set of states labeled by the *win* proposition. Let ρ be the transition relation, such that $\rho(s_1, s_2)$ holds whenever s_2 is a direct successor of the state s_1 in the state-transition diagram.

For a state-set S, we introduce the predecessor operator $Pre_\exists$ which computes the set of all one-step predecessors of the states in S. That is,

$$Pre_\exists(S) \quad = \quad \{s \mid s \text{ has a } \rho\text{-successor in } S\}$$

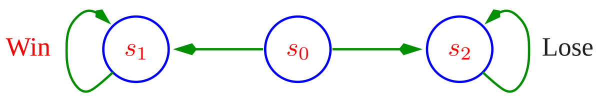
Recursively, we define a state s to be goal reaching if either $s \in \|win\|$ or s has a goal reaching successor. That is,

$$R = \|win\| \ \cup \ Pre_\exists(R)$$

We may expect that the solution to this fix-point equation, will give us the set of all states from which $\|win\|$ is reachable.

Problem: Not Every Solution is Satisfactory

Consider the diagram:



Both $R_{0,1} = \{s_0, s_1\}$ and $R_{0..2} = \{s_0, s_1, s_2\}$ satisfy the equation

$$R = \|win\| \ \cup \ Pre_\exists(R)$$

but only $R_{0,1} = \{s_0, s_1\}$ captures the set of states from which $\|win\|$ is reachable.

Conclusion: We should take the minimal solution of the fix-point equation $R = \|win\| \ \cup \ Pre_\exists(R)$ which we denote by

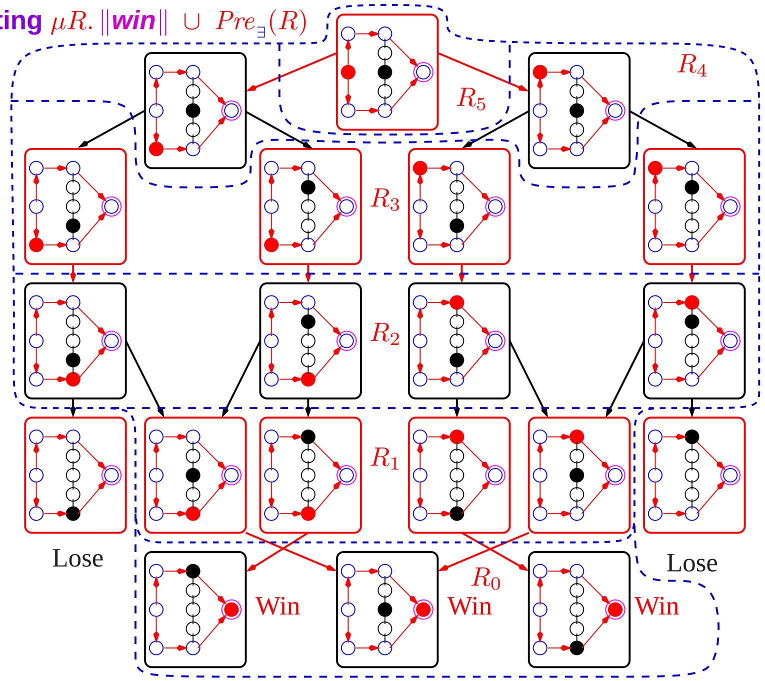
$$\mu R.\, (\|win\| \ \cup \ Pre_\exists\, R)$$

This minimal solution can be effectively computed by the iteration sequence:

$$\begin{aligned} R_0 &= \emptyset \\ R_1 &= \|win\| \\ R_2 &= R_0 \ \cup \ Pre_\exists\, R_0 \\ R_3 &= R_1 \ \cup \ Pre_\exists\, R_1 \\ &\cdots \end{aligned}$$

Consequently, the goal is reachable from an initial state s_0 iff $s_0 \in \mu R.\, (\|win\| \ \cup \ Pre_\exists\, R)$.

Computing $\mu R.\, \|win\| \ \cup \ Pre_\exists(R)$



Controller Synthesis

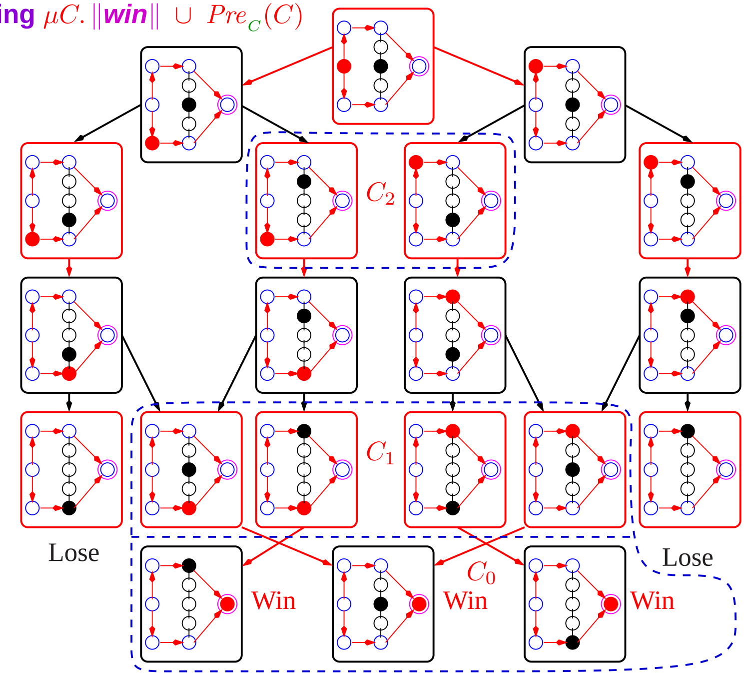
Distinguishing between the two players, we define

$$Pre_{\exists}(S) = \{s \mid \text{Some red successor of } s \text{ is in } C\}$$
$$Pre_{\forall}(S) = \{s \mid \text{All \textbf{black} successors of } s \text{ are in } C\}$$

The two operators can be combined, and the expression $Pre_{C}(C) = Pre_{\exists}(Pre_{\forall}(C))$ denotes the set of states s which have at least one red successor s_1 all of whose **black** successors belong to C. If we think about the moves as taken in turn by two players, then $Pre_{C}(C)$ denotes the states from which the red player can force the game after a complete round (each player making one move) into a C-state.

The expression $control(win) = \mu C. \|win\| \cup Pre_{C}(C)$ characterizes all the states from which the red player can force a visit to a win state in a finite number of moves.

Computing $\mu C. \|win\| \cup Pre_{C}(C)$

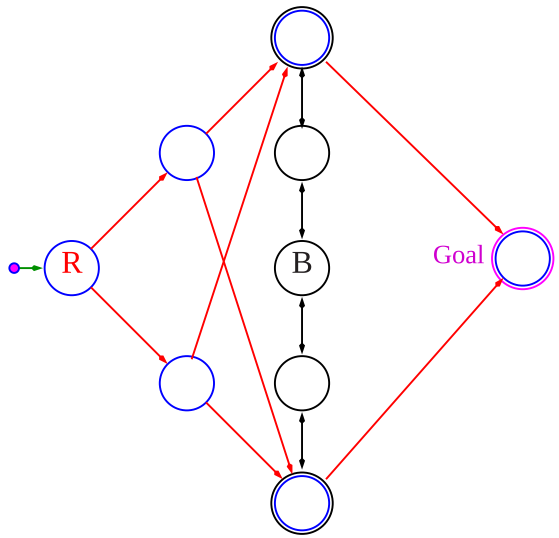


Conclusions

The runner and the blocker can cooperate to reach a winning state for R.

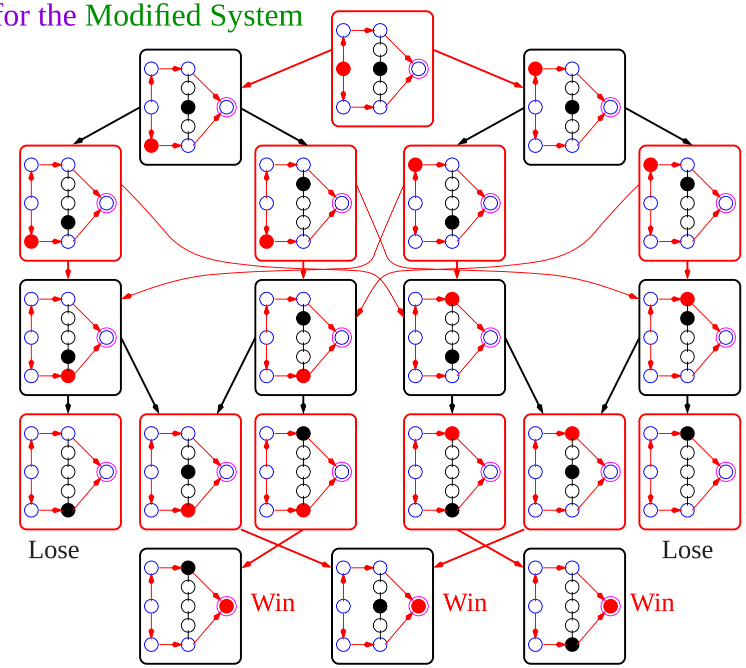
However, R cannot force a win.

The Modified Runner Blocker System

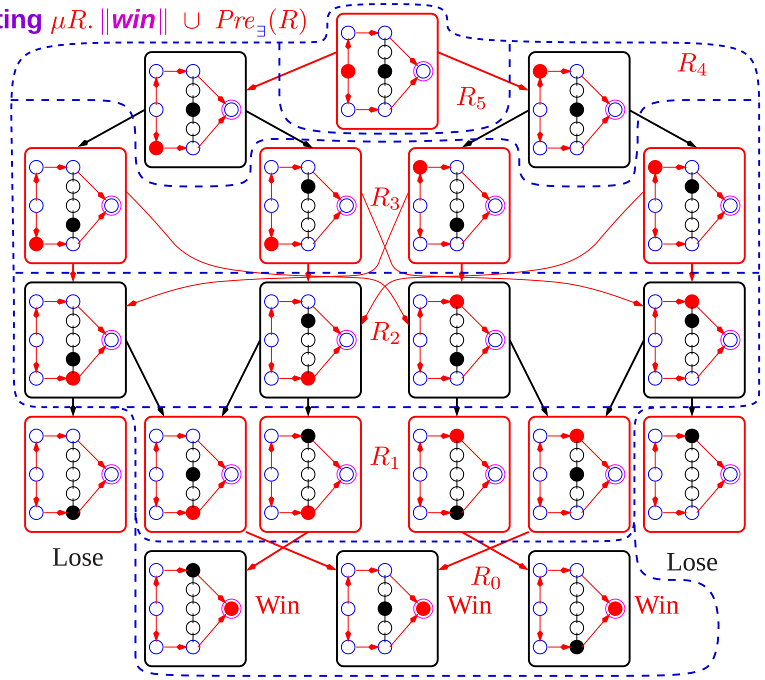


Additional transitions have been added to the runner.

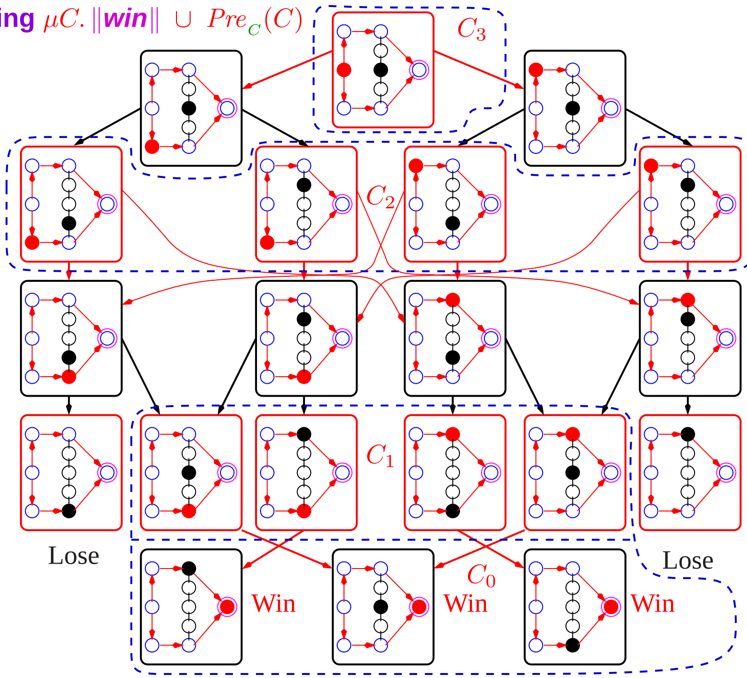
Game Tree for the Modified System



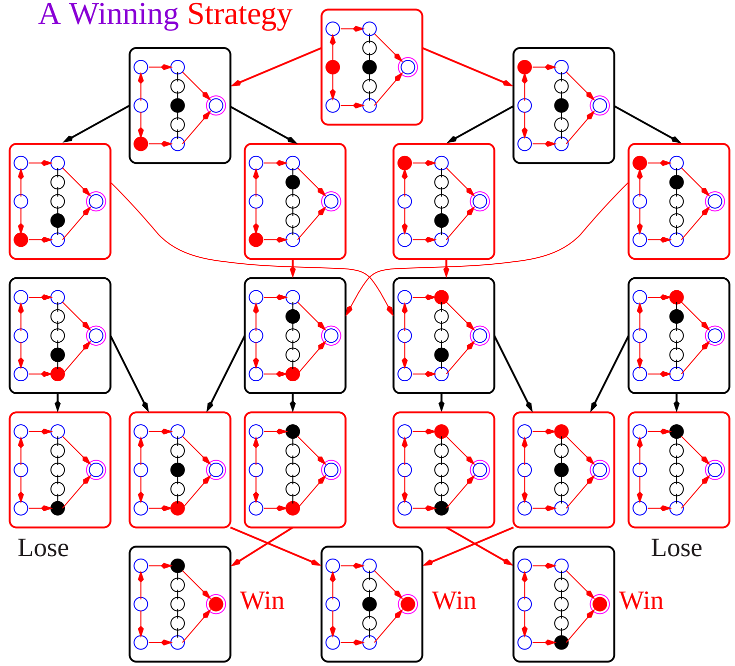
Computing $\mu R.\ \|win\| \cup Pre_{\exists}(R)$



Computing $\mu C.\ \|win\| \cup Pre_{C}(C)$



A Winning Strategy

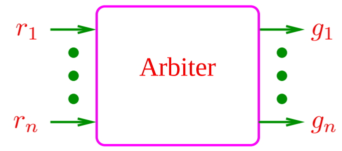


Apply to Programs

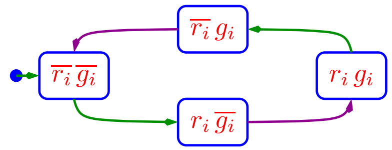
Let us apply the controller synthesis paradigm to synthesis of programs (or designs, in general).

Example Design: Arbiter

Consider a specification for an arbiter.



The protocol for each client:

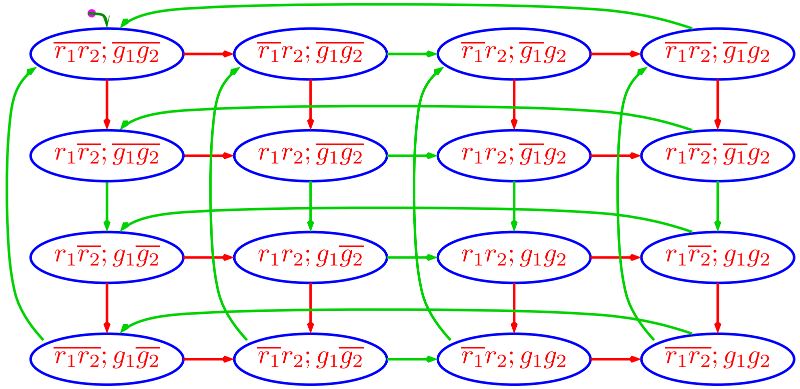


Required to satisfy

$$\bigwedge_{i \neq j} \square \neg(g_i \wedge g_j) \quad \wedge \quad \bigwedge_i \square \diamond (g_i = r_i)$$

Start by Controller Synthesis

Assume a given platform (plant), identifying controllable (system) and uncontrollable (environment) transitions:



By default every node is connected to itself by both green and red transitions. A complete move consists of a red edge followed by a green edge, visiting at most two different states. Also given is an LTL specification (winning condition):

$$\varphi : \quad \square \neg(g_1 \wedge g_2) \ \wedge \ \square \diamond (g_1 = r_1) \ \wedge \ \square \diamond (g_2 = r_2)$$

Synthesis Via Game Playing

A game is given by $\mathcal{G} : \langle V = X \cup Y, \Theta, \rho_1, \rho_2, \varphi \rangle$, where

- $V = X \cup Y$ are the state variables, with X being the environment's (player 1) variables, and Y being the system's (player 2) variables. A state of the game is an interpretation of V. Let Σ denote the set of all states.
- Θ — the initial condition. An assertion characterizing the initial states.
- $\rho_1(X, Y, X')$ — Transition relation for player 1 (Environment).
- $\rho_2(X, Y, X', Y')$ — Transition relation for player 2 (system).
- φ — The winning condition. An LTL formula characterizing the plays which are winning for player 2.

A state s_2 is said to be a $\mathcal{G}$-successor of state s_1, if both $\rho_1(s_1[V], s_2[X])$ and $\rho_2(s_1[V], s_2[V])$ are true.

We denote by D_X and D_Y the domains of variables X and Y, respectively.

Plays and Strategies

Let $\mathcal{G} : \langle V, \Theta, \rho_1, \rho_2, \varphi \rangle$ be a game. A play of $\mathcal{G}$ is an infinite sequence of states

$$\pi : \quad s_0, s_1, s_2, \ldots,$$

satisfying:

- Initiality: $s_0 \models \Theta$.
- Consecution: For each $j \geq 0$, the state s_{j+1} is a $\mathcal{G}$-successor of the state s_j.

A play π is said to be winning for player 2 if $\pi \models \varphi$. Otherwise, it is said to be winning for player 1.

A strategy for player 1 is a function $\sigma_1 : \Sigma^+ \mapsto D_X$, which determines the next set of values for X following any history $h \in \Sigma^+$. A play $\pi : s_0, s_1, \ldots$ is said to be compatible with strategy σ_1 if, for every $j \geq 0$, $s_{j+1}[X] = \sigma_1(s_0, \ldots, s_j)$.

Strategy σ_1 is winning for player 1 from state s if all s-originated plays compatible with σ_1 are winning for player 1. If such a winning strategy exists, we call s a winning state for player 1.

Similar definitions hold for player 2 with strategies of the form $\sigma_2 : \Sigma^+ \times D_X \mapsto D_Y$.

From Winning Games to Programs

A game $\mathcal{G}$ is said to be winning for player 2 (player 1, respectively) if all (some) initial states are winning for 2 (1, respectively).

We solve the game, attempting to decide whether the game is winning for player 1 or 2. If it is winning for player 1 the specification is unrealizable. If it is winning for player 2, we can extract a winning strategy which is a working implementation.

When applying controller synthesis, the platform provides the transition relations ρ_1 and ρ_2, as well as the initial condition.

Thus, the essence of synthesis under the controller framework is an algorithm for computing the set of winning states for a given platform and specification φ.

The Controlled Predecessor

As in symbolic model checking, computing the winning states involves fix-point computations over a basic predecessor operator. For model checking the operator is $\mathbf{E}\bigcirc p$ satisfied by all states which have a p-state as a successor.

For synthesis, we use the controlled predecessor operator $\boxed{\diamond}p$. Its semantics can be defined by

$$\boxed{\diamond}p : \quad \forall X' : \rho_1(V, X') \rightarrow \exists Y' : \rho_2(V, V') \wedge p(V')$$

where ρ_1 and ρ_2 are the transition relations of the environment and system, respectively.

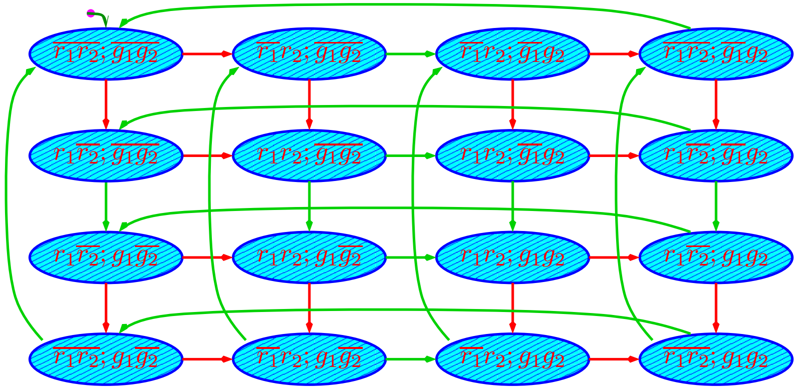
In our graphic notation, $s \models \boxed{\diamond}p$ iff s has at least one green p-successor, and all red successors different from s satisfy p.

Solving $\Box\, p$ Games, Iteration 0

The set of winning states for a specification $\Box\, p$ can be computed by the fix-point expression:

$$\nu Y. p \wedge \boxed{\diamond}Y \quad = \quad p \wedge \boxed{\diamond}p \wedge \boxed{\diamond}\boxed{\diamond}p \wedge \cdots$$

We illustrate this on the specification $\Box\, \neg(g_1 \wedge g_2)$.



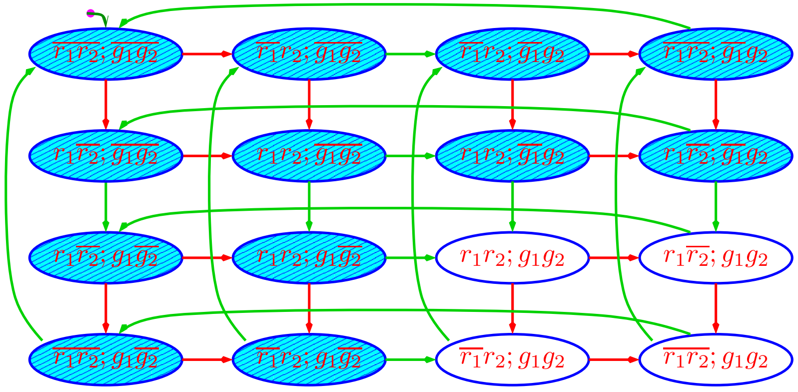
Iteration 0, $\quad Y_0 : 1$

Solving $\Box\, p$ Games, Iteration 1

The set of winning states for a specification $\Box\, p$ can be computed by the fix-point expression:

$$\nu Y.\, p \;\wedge\; \boxed{\Diamond} Y \quad = \quad p \wedge \boxed{\Diamond} p \wedge \boxed{\Diamond}\boxed{\Diamond} p \wedge \cdots$$

We illustrate this on the specification $\Box\, \neg(g_1 \wedge g_2)$.



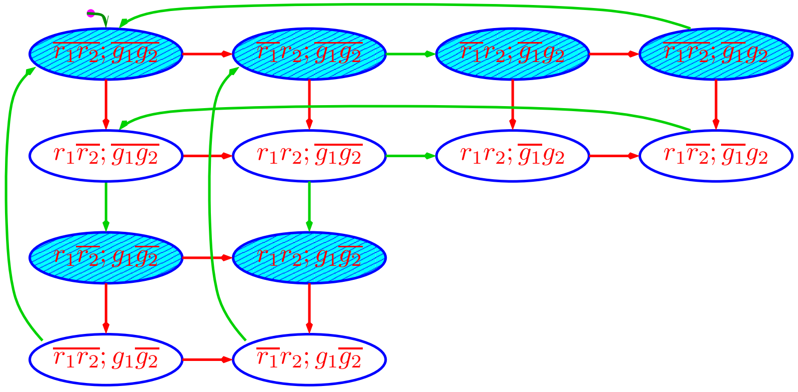
Iteration 1, $\quad Y_1 : \neg(g_1 \wedge g_2) \wedge \boxed{\Diamond} 1$

Solving $\Diamond\, q$ Games, Iteration 1

The set of winning states for a specification $\Diamond\, q$ can be computed by the fix-point expression:

$$\mu Y.\, q \;\vee\; \boxed{\Diamond} Y \quad = \quad q \vee \boxed{\Diamond} q \vee \boxed{\Diamond}\boxed{\Diamond} q \vee \cdots$$

We illustrate this on the specification $\Diamond\, (g_1 = r_1)$.



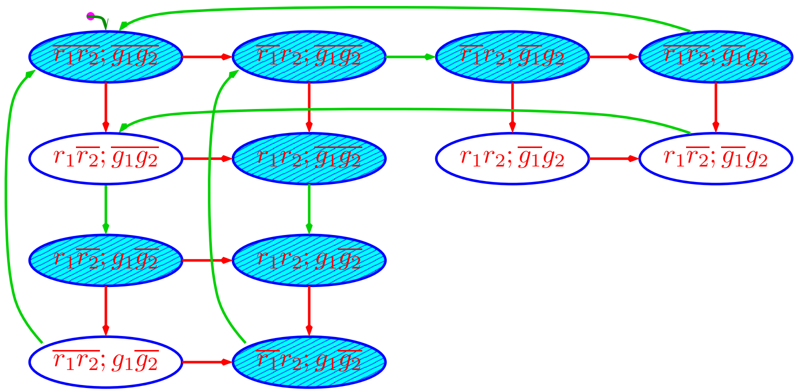
Iteration 1, $\quad Y_1 : (g_1 = r_1)$

Solving $\Diamond\, q$ Games, Iteration 2

The set of winning states for a specification $\Diamond\, q$ can be computed by the fix-point expression:

$$\mu Y.\, q \;\vee\; \boxed{\Diamond} Y \quad = \quad q \vee \boxed{\Diamond} q \vee \boxed{\Diamond}\boxed{\Diamond} q \vee \cdots$$

We illustrate this on the specification $\Diamond\, (g_1 = r_1)$.



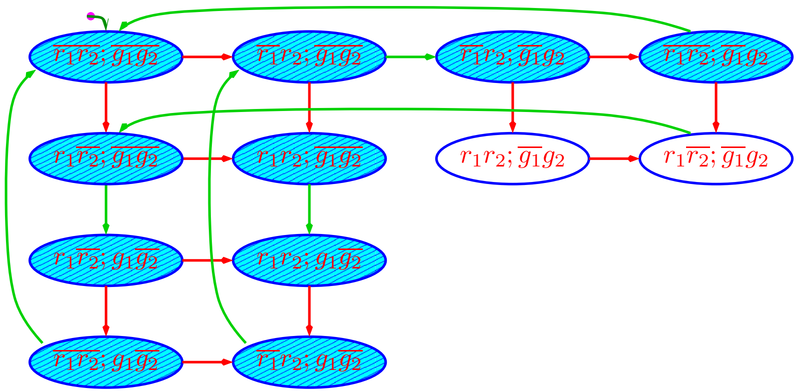
Iteration 2, $\quad Y_2 : Y_1 \vee \boxed{\Diamond} Y_1$

Solving $\Diamond\, q$ Games, Iteration 3

The set of winning states for a specification $\Diamond\, q$ can be computed by the fix-point expression:

$$\mu Y.\, q \;\vee\; \boxed{\Diamond} Y \quad = \quad q \vee \boxed{\Diamond} q \vee \boxed{\Diamond}\boxed{\Diamond} q \vee \cdots$$

We illustrate this on the specification $\Diamond\, (g_1 = r_1)$.



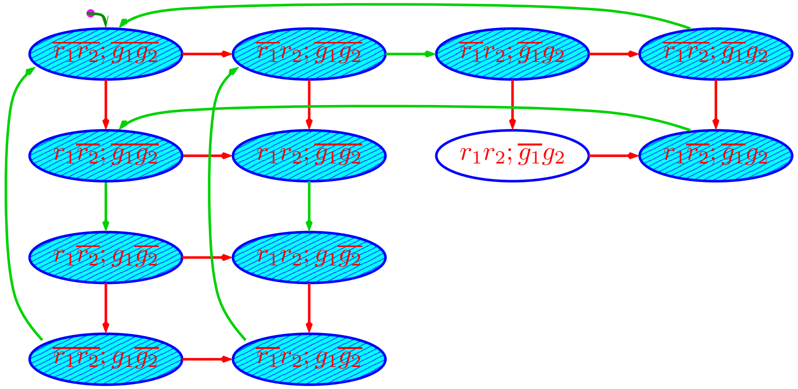
Iteration 3, $\quad Y_3 : Y_2 \vee \boxed{\Diamond} Y_2$

Solving $\diamond\, q$ Games, Iteration 4 (Final)

The set of winning states for a specification $\diamond\, q$ can be computed by the fix-point expression:

$$\mu Y.\, q \;\vee\; \boxed{\diamond} Y \quad = \quad q \vee \boxed{\diamond} q \vee \boxed{\diamond}\boxed{\diamond} q \vee \cdots$$

We illustrate this on the specification $\diamond\, (g_1 = r_1)$.



Iteration 4, $\quad Y_4 : Y_3 \vee \boxed{\diamond} Y_3$

Solving $\square\, \diamond\, q$ Games

A game for a winning condition of the form $\square\, \diamond\, q$ can be solved by the fix-point expression:

$$\nu Z \mu Y.\, q \wedge \boxed{\diamond} Z \vee \boxed{\diamond} Y$$

This is based on the maximal fix-point solution of the equation

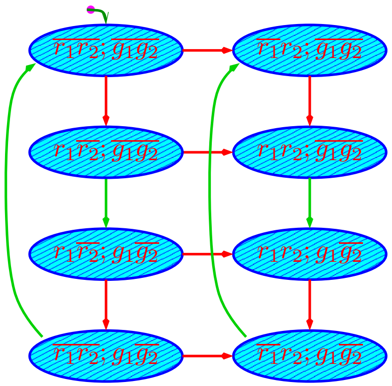
$$Z \quad = \quad \mu Y.\, (q \wedge \boxed{\diamond} Z) \vee \boxed{\diamond} Y$$

This nested fix-point computation can be computed iteratively by the program:

$Z := 1$
Fix (Z)
$\left[\begin{array}{l} G := q \;\wedge\; \boxed{\diamond} Z \\ Y := 0 \\ \textbf{Fix } (Y) \\ \quad [Y := G \;\vee\; \boxed{\diamond} Y] \\ Z := Y \end{array}\right]$

Solving $\square\, \diamond\, (g_1 = r_1)$ for the Arbiter Example

Applying the above fix-point iterations to the Arbiter example, we obtain:



Note that the obtained strategy, keeps $g_2 = 0$ permanently. This suggests that we will have difficulties finding a solution that will maintain

$$\square\, \diamond\, (g_1 = r_1) \;\wedge\; \square\, \diamond\, (g_2 = r_2)$$

Generalized Response (Büchi)

Solving the game for $\square\, \diamond\, q_1 \;\wedge\; \cdots \;\wedge\; \square\, \diamond\, q_n$.

$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left((q_1 \wedge \boxed{\diamond} Z_2) \;\vee\; \boxed{\diamond} Y \right) \\ \mu Y \left((q_2 \wedge \boxed{\diamond} Z_3) \;\vee\; \boxed{\diamond} Y \right) \\ \vdots \\ \vdots \\ \mu Y \left((q_n \wedge \boxed{\diamond} Z_1) \;\vee\; \boxed{\diamond} Y \right) \end{bmatrix}$$

Iteratively:

For $(i \in 1..n)$ **do** $[Z[i] := 1]$
Fix $(Z[1])$
$\left[\begin{array}{l} \textbf{For } (i \in 1..n) \textbf{ do} \\ \left[\begin{array}{l} Y := 0 \\ \textbf{Fix } (Y) \\ \quad \left[Y := (q[i] \wedge \boxed{\diamond} Z[i \oplus_n 1]) \;\vee\; \boxed{\diamond} Y\right] \\ Z[i] := Y \end{array}\right] \end{array}\right]$
Return $Z[1]$

Specification is Unrealizable

Applying the above algorithm to the specification

$$\Box\Diamond(g_1 = r_1) \;\wedge\; \Box\Diamond(g_2 = r_2)$$

we find that it fails. Conclusion:

The considered specification is unrealizable

Indeed, without an environment obligation of releasing the resource once it has been granted, the arbiter cannot satisfy any other client.

Property-Based System Design

While the rest of the world seems to be moving in the direction of model-based design (see System-C, UML), some of us persist with the vision of property-based approach.

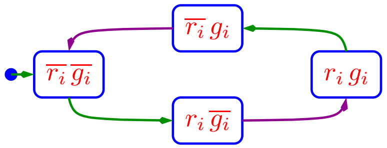
Specification is stated declaratively as a set of properties, from which a design can be extracted.

This is currently studied in the project PROSYD.

Design synthesis is needed in two places in the development flow:

- Automatic synthesis of small blocks whose time and space efficiency are not critical.
- As part of the specification analysis phase, ascertaining that the specification is realizable.

A Realizable Specification



Assumptions (Constraints on the Environment)

$$A: \quad \bigwedge_i \left(\overline{r_i} \;\wedge\; (r_i \neq g_i) \Rightarrow (\bigcirc r_i = r_i) \;\wedge\; r_i \wedge g_i \Rightarrow \Diamond\, \overline{r_i}\right)$$

Guarantees (Expectations from System)

$$G: \quad \bigwedge_{i\neq j} \Box\neg(g_i \wedge g_j) \;\wedge\; \bigwedge_i \left(\overline{g_i} \;\wedge\; \begin{pmatrix} r_i = g_i & \Rightarrow & \bigcirc g_i = g_i & \wedge \\ r_i \wedge \overline{g_i} & \Rightarrow & \Diamond\, g_i & \wedge \\ \overline{r_i} \wedge g_i & \Rightarrow & \Diamond\, \overline{g_i} & \end{pmatrix}\right)$$

Total Specification

$$\varphi: \quad A \rightarrow G$$

Program Sythesis from LTL Specification

Assume we are given a set of LTL specifications. We construct a game as follows:

- As Θ we take all the non-temporal specification parts which relate to the initial state.
- As ρ_1 and ρ_2, we can take True. A more efficient choice is to include in ρ_1 (similarly ρ_2) all local limitations on the next values of X (resp. Y), such as

$$r_i \;\wedge\; \neg g_i \quad\rightarrow\quad r_i'$$

- We place in φ all the remaining properties that have not already been included in Θ, ρ_1, and ρ_2.

We solve the game, attempting to decide whether the game is winning for player 1 or 2. If it is winning for player 1 the specification is unrealizable. If it is winning for player 2, we can extract a winning strategy which is a working implementation.

The Game for the Sample Specification

For the specification

$$\bigwedge_i \left(\overline{r_i} \;\wedge\; (r_i \neq g_i) \Rightarrow (\bigcirc r_i = r_i) \;\wedge\; r_i \wedge g_i \Rightarrow \Diamond\, \overline{r_i}\right) \;\to\; \bigwedge_{i \neq j} \square\, \neg(g_i \wedge g_j) \;\wedge\; \bigwedge_i \left(\overline{g_i} \;\wedge\; \begin{pmatrix} r_i = g_i & \Rightarrow & \bigcirc g_i = g_i & \wedge \\ r_i \wedge \overline{g_i} & \Rightarrow & \Diamond\, g_i & \wedge \\ \overline{r_i} \wedge g_i & \Rightarrow & \Diamond\, \overline{g_i} & \end{pmatrix}\right)$$

We take the following game components:

$$\begin{aligned} X \cup Y :&\quad \{r_i \mid i = 1, \ldots, n\} \cup \{g_i \mid i = 1, \ldots, n\} \\ \Theta :&\quad \bigwedge_i (\overline{r_i} \;\wedge\; \overline{g_i}) \\ \rho_1 :&\quad \bigwedge_i ((r_i \neq g_i) \to (r_i' = r_i)) \\ \rho_2 :&\quad \bigwedge_{i \neq j} \neg(g_i' \;\wedge\; g_j') \;\wedge\; \bigwedge_i ((r_i = g_i) \to (g_i' = g_i)) \\ \varphi :&\quad \bigwedge_i (r_i \wedge g_i \Rightarrow \Diamond\, \overline{r_i}) \;\to\; \bigwedge_i ((r_i \wedge \overline{g_i} \Rightarrow \Diamond\, g_i) \;\wedge\; (\overline{r_i} \wedge g_i \Rightarrow \Diamond\, \overline{g_i})) \end{aligned}$$

Solving in Polynomial Time a Doubly Exponential Problem

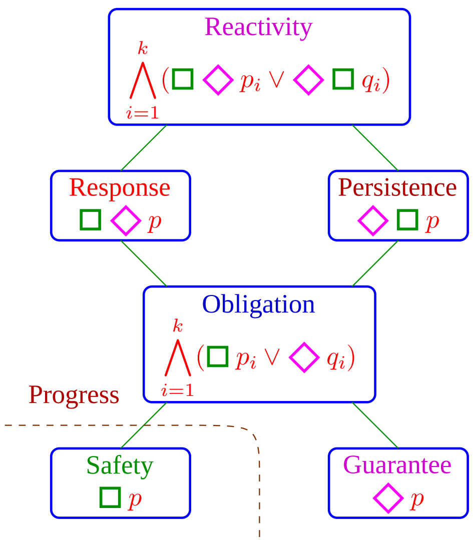
In [1989] Roni Rosner provided a general solution to the Synthesis problem. He showed that any approach that starts with the standard translation from LTL to Büchiautomata, has a doubly exponential lower bound.

One of the messages resulting from the work reported here is

> Do not be too hasty to translate LTL into automata. Try first to locate the formula within the temporal hierarchy.

For each class of formulas, synthesis can be performed in polynomial time.

Hierarchy of the Temporal Properties



where p, p_i, q, q_i are past formulas.

Solving Games for Generalized Reactivity[1] (Streett[1])

Following [KPP03], we present an n^3 algorithm for solving games whose winning condition is given by the (generalized) Reactivity[1] condition

$$(\square \Diamond p_1 \wedge \square \Diamond p_2 \wedge \cdots \wedge \square \Diamond p_m) \;\to\; \square \Diamond q_1 \wedge \square \Diamond q_2 \wedge \cdots \wedge \square \Diamond q_n$$

This class of properties is bigger than the properties specifiable by deterministic Büchiautomata. It covers a great majority of the properties we have seen so far.

For example, it covers the realizable version of the specification for the Arbiter design.

The Solution

The winning states in a Streett[1] game can be computed by

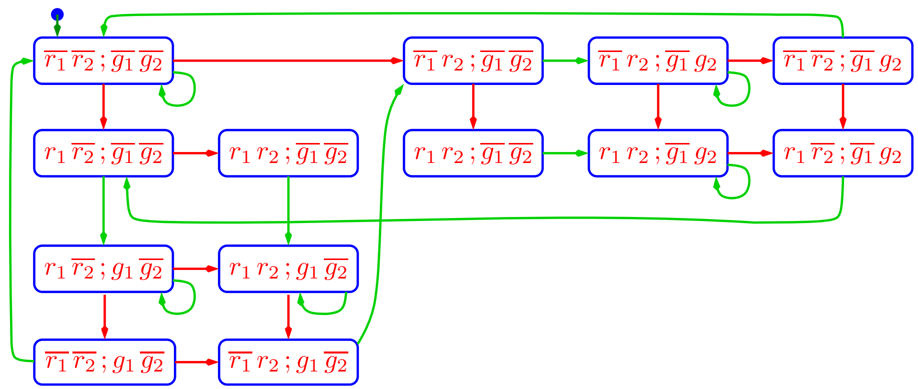
$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{j=1}^{m} \nu X (q_1 \wedge \boxed{\diamond} Z_2 \ \vee \ \boxed{\diamond} Y \ \vee \ \neg p_j \wedge \boxed{\diamond} X) \right) \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X (q_2 \wedge \boxed{\diamond} Z_3 \ \vee \ \boxed{\diamond} Y \ \vee \ \neg p_j \wedge \boxed{\diamond} X) \right) \\ \vdots \\ \vdots \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X (q_n \wedge \boxed{\diamond} Z_1 \ \vee \ \boxed{\diamond} Y \ \vee \ \neg p_j \wedge \boxed{\diamond} X) \right) \end{bmatrix}$$

where

$$\boxed{\diamond}\varphi : \quad \forall X' : \rho_1(V, X') \rightarrow \exists Y' : \rho_2(V, V') \wedge \varphi(V')$$

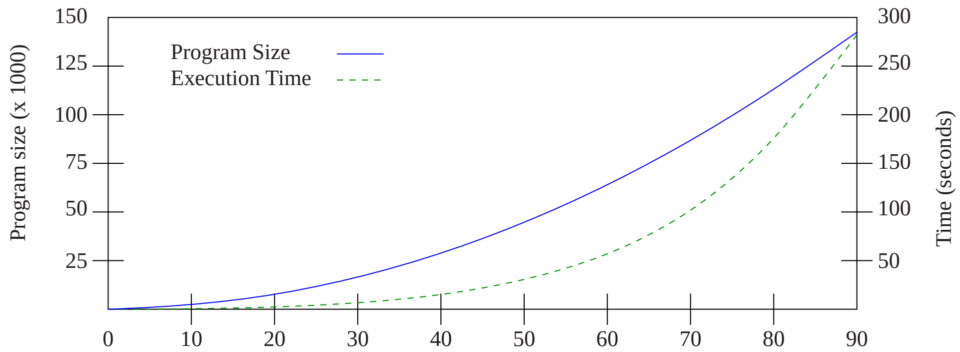
Results of Synthesis

The design realizing the specification can be extracted as the winning strategy for Player 2. Applying this to the Arbiter specification, we obtain the following design:



There exists a symbolic algorithm for extracting the implementing design/winning strategy.

Execution Times and Programs Size for Arbiter(N)

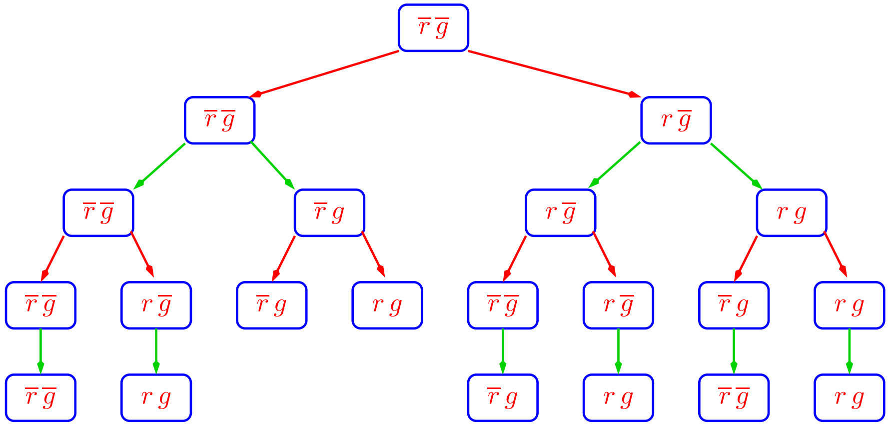


Conclusions

- It is possible to perform design synthesis for restricted fragments of LTL in acceptable time.
- The tractable fragment (React(1)) covers most of the properties that appear in standard specifications.
- It is worthwhile to invest an effort in locating the formula within the temporal hierarchy. Solving a game in React(k) has complexity $N^{(2k+1)}$.

The Semantics of Game Analysis

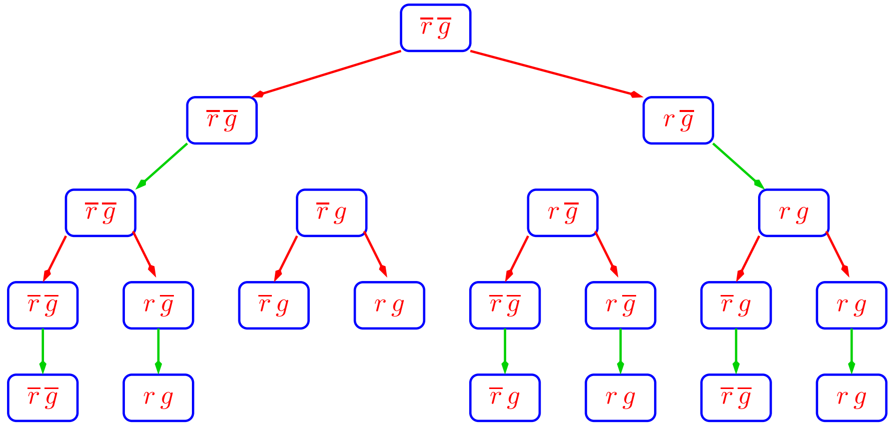
We can always consruct the game tree



Strategies

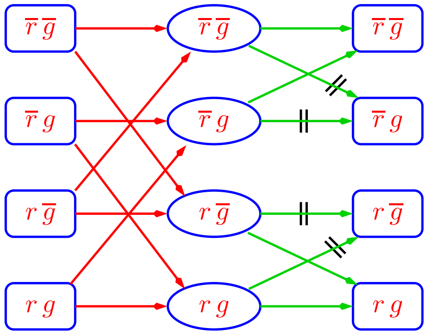
All strategies can be represented as pruning of the tree at the controllable levels. For example, a strategy for the specification

$$(r \Rightarrow \bigcirc g) \;\land\; (\overline{r} \Rightarrow \bigcirc \overline{g})$$



Folding the Tree Into a Finite Graph

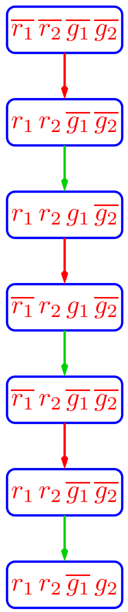
In order to be algorithmically tractable, we need to perform the pruning process over the finite graph which generated the game tree. In many cases, this is possible (and leads to memory-less strategy).



Folding not Immediate

There are cases in which the pruning must depend on the path leading to the current state.

Folding is still possible but may need a longer period.



Controller (Design) Extraction

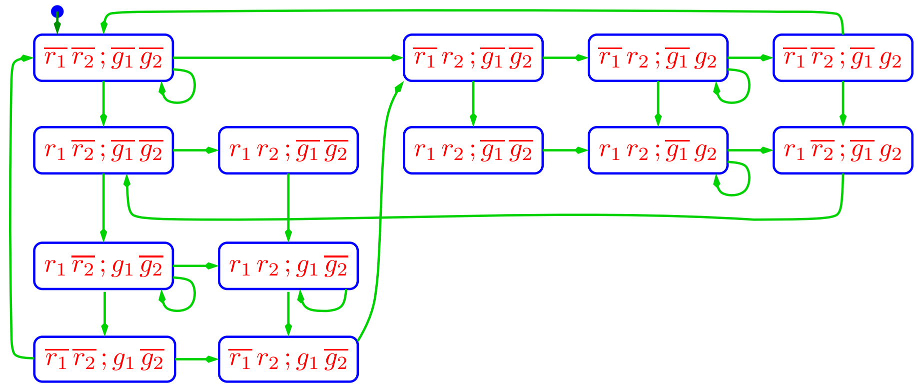
It remains to show how to extract a winning strategy for the case that a game is winning for player 2.

Let $\mathcal{G} : \langle V = X \cup Y, \Theta, \rho_1, \rho_2, \varphi \rangle$ be a given game. A controller for $\mathcal{G}$ is an FDS $\mathcal{G}_c : \langle V_c, \Theta_c, \rho_c, \emptyset, \emptyset \rangle$, such that:

- $V_c \supseteq V$. That is, V_c extends the set of variables of $\mathcal{G}$.
- $\Theta_c \Downarrow_V = \Theta$. That is, when projecting the initial states of $\mathcal{G}_c$ on the variables of $\mathcal{G}$, we obtain precisely the initial states of $\mathcal{G}$.
- $\rho_c \rightarrow \rho$, where $\rho = \rho_1 \wedge \rho_2$. That is, if s_2 is a ρ_c successor of s_1, then s_2 is also a ρ_c successor of s_1.
- Player-1 Completenss — $\rho_c \Downarrow_{V,X'} = \rho_1$. That is, a state $s_1 \in \Sigma_c$ has a ρ_1-successor s_2 iff s_1 has a ρ_c-successor which agrees with s_2 on the valuation of X.
- Every infinite run of $\mathcal{G}_c$ satisfies the winning condition φ.

Example: Extracted Controller for Arbiter

Following is the controller extracted for the Arbiter example:



Interpreting a Controller as a Program

A program (equivalently, a circuit) implementing the extracted controller follows the states that are contained in $\mathcal{G}_c$. It has a program counter which ranges over the states of $\mathcal{G}_c$.

Assume that control is currently at state S of $\mathcal{G}_c$. Let the next values of the input variables be $X = \xi$. Choose a state S' which is a ρ_c-successor of S, and such that $S'[X] = \xi$. By the requirement of Player-1 Completenss, there always exists such a successor.

The actions of the program is to output the values η such that $S'[Y] = \eta$, and to move to state S'.

Computing a Controller for the Winning Condition $\square\, p$

The winning states in a game with a winning condition $\square\, p$ are given by:

$$\textit{win} \quad = \quad \nu Z.\, p \ \wedge\ \boxed{\diamond} Z$$

The full contoller extraction algorithm can be given by the following program:

$Z := 1$
Fix (Z)
  $[\ Z := p \wedge \boxed{\diamond} Z\ ]$
if $(\Theta \wedge \neg Z) \neq 0$ **then**
  Print "Specification is unrealizable"
else
  $\begin{bmatrix} \Theta_c & := & \Theta \\ \rho_c & := & Z \wedge \rho \wedge Z' \end{bmatrix}$

Claim 10. *If s is a winning state of a $(\square\, p)$-game, then $s \models p$, and player 2 can force the game to move from s to a succesor which is also a winning state.*

Computing A Controller for the Winning Condition $\Diamond\, q$

The winning states in a game with a winning condition $\Diamond\, p$ are given by:

$$\textit{win} \quad = \quad \mu Y.\, q \ \lor \ \boxed{\Diamond} Y$$

The full contoller extraction algorithm can be given by the following program:

$Y := q; \quad r := 0; \quad U[0] := q$
Fix (Y)
  $[\ Y := q \lor \boxed{\Diamond} Y; \quad r := r + 1; \quad U[r] := Y \ ]$
if $(\Theta \land \neg Y) \neq 0$ **then**
  Print "Specification is unrealizable"
else
  $\Theta_c := \Theta$
  $\rho_c := 0; \quad \textit{prev} := U[0]$
  for $i \in 1 \ldots r$ **do**
    $[\ \rho_c := \rho_c \ \lor \ (U[i] \land \neg \textit{prev}) \land \rho \land \textit{prev}'; \quad \textit{prev} := \textit{prev} \lor U[i] \ ]$

Claim 11. *Every winning state s in a $(\Diamond\, q)$-game is associated with a natural rank $r(s) \geq 0$, such that if $r(s) = 0$ then $s \models q$, and if $r(s) > 0$, then player 2 can force the game to move from s to a winning succesor with a lower rank.*

Computing A Controller for the Winning Condition $\Box \Diamond\, q$

The winning states in a game with a winning condition $\Box \Diamond\, p$ are given by:

$$\textit{win} \quad = \quad \nu Z \mu Y.\, (q \land \boxed{\Diamond} Z) \ \lor \ \boxed{\Diamond} Y$$

The full contoller extraction algorithm can be given by the following program:

$Z := 1$
Fix (Z)
  $Y := q \land \boxed{\Diamond} Z; \quad r := 0; \quad U[0] := Y$
  Fix (Y)
    $[\ Y := q \lor \boxed{\Diamond} Y; \quad r := r + 1; \quad U[r] := Y \ ]$
if $(\Theta \land \neg Z) \neq 0$ **then**
  Print "Specification is unrealizable"
else
  $\Theta_c := \Theta$
  $\rho_c := U[0] \land \rho \land Z'; \quad \textit{prev} := U[0]$
  for $i \in 1 \ldots r$ **do**
    $[\ \rho_c := \rho_c \ \lor \ (U[i] \land \neg \textit{prev}) \land \rho \land \textit{prev}'; \quad \textit{prev} := \textit{prev} \lor U[i] \ ]$

Claim 12. *Every winning state s in a $(\Box \Diamond\, q)$-game is associated with a natural rank $r(s)$, such that player 2 can force the game to move from s to a winning succesor s' where either $r(s) = 0$ and $s \models q$, or $r(s) > r(s')$.*