Synthesis of Designs from Property Specifications

Amir Pnueli

New York University and Weizmann Institute of Sciences (Emeritus)

ISCAS, Beijing, December, 2008

Joint work with

Nir Piterman, Yaniv Sa'ar,

Research Supported in part by SRC grant 2004-TJ-1256 and the European Union project Prosyd.

Dedication

I wish to dedicate this lecture to

Prof. TANG Zhisong (Chih-sung TANG)

of ISCAS. A great scholar and a personal friend to whom we have had a great respect and admiration both as a scientist and as a wise and inspiring human being.

Motivation

Why verify, if we can automatically synthesize a program which is correct by construction?

A Brief History of System Synthesis

In 1965 Church formulated the following Church problem: Given a circuit interface specification (identification of input and output variables) and a behavioral specification, e.g. an LTL formula.

- Determine if there exists an automaton (sequential circuit) which realizes the specification.
- If the specification is realizable, construct an implementing circuit

Originally, the specification was given in the sequence calculus which is an explicit-time temporal logic.

Example of a Specification

Behavioral Specification:

$$\square\,(\bigcirc x = (y \oplus \bigcirc y))$$

Is this specification realizable?

The essence of synthesis is the conversion

From relations to Functions.

From Relations to Functions

Consider a computational program:

- The relation $x = y^2$ is a specification for the program computing the function $y = \sqrt{x}$.
- The relation $x \models y$ is a specification for the program that finds a satisfying assignment to the CNF boolean formula x.

Checking is easier than computing.

Solutions to Church's Problem

In 1969, M. Rabin provided a first solution to Church's problem. Solution was based on automata on Infinite Trees. All the concepts involving ω-automata were invented for this work.

At the same year, Büchi and Landweber provided another solution, based on infinite games.

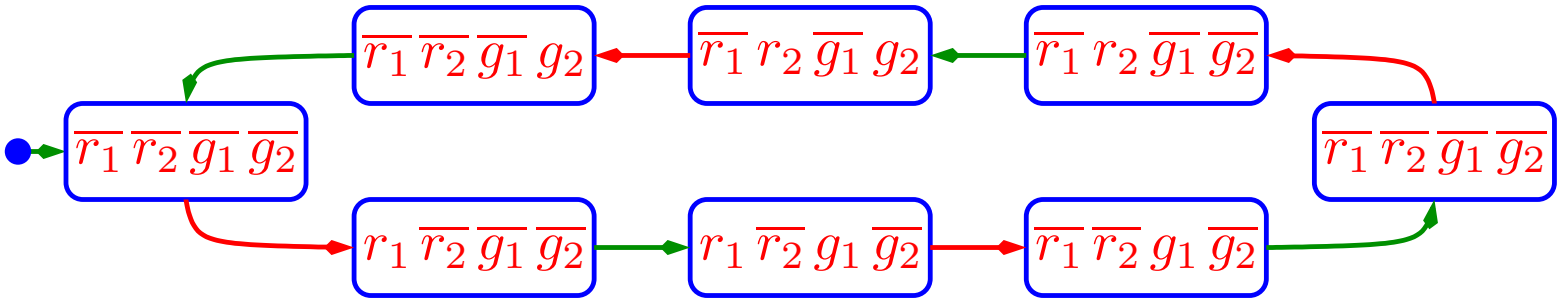
These two techniques (Trees and Games) are still the main techniques for performing synthesis.

Synthesis of Reactive Modules from Temporal Specifications

Around 1981 Wolper and Emerson, each in his preferred brand of temporal logic (linear and branching, respectively), considered the problem of synthesis of reactive systems from temporal specifications.

Their (common) conclusion was that specification φ is realizable iff it is satisfiable, and that an implementing program can be extracted from a satisfying model in the tableau.

A typical solution they would obtain for the arbiter problem is:



Such solutions are acceptable only in circumstances when the environment fully cooperate with the system.

Next Step: Realizability $\sqsubset$ Satisfiability

There are two different reasons why a specification may fail to be realizable.

Inconsistency

$$\Diamond g \quad \wedge \quad \Box \neg g$$

Unrealizability For a system

Realizing the specification

$$g \quad \longleftrightarrow \quad \Diamond r$$

requires clairvoyance.

The essence of reactive systems synthesis is the conversion from relations to causal functions.

A Synthesized Module Should Maintain Specification Against Adversarial Environment

In 1988, Rosner claimed that realizability should guarantee the specification against all possible (including adversarial) environments.

In a related work he has shown [1989] that the synthesis process has worst case complexity which is doubly exponential. The first exponent comes from the translation of φ into a non-deterministic Büchi automaton. The second exponent is due to the determinization of the automaton.

This result doomed synthesis to be considered highly intractable, and discouraged further research on the subject for a long time.

Simple Cases of Lower Complexity

In 1989, Ramadge and Wonham introduced the notion of controller synthesis and showed that for a specification of the form $\Box p$, the controller can be synthesized in linear time.

In 1995, Asarin, Maler, P, and Sifakis, extended controller synthesis to timed systems, and showed that for specifications of the form $\Box p$ and $\Diamond q$, the problem can be solved by symbolic methods in linear time.

Property-Based System Design

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

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

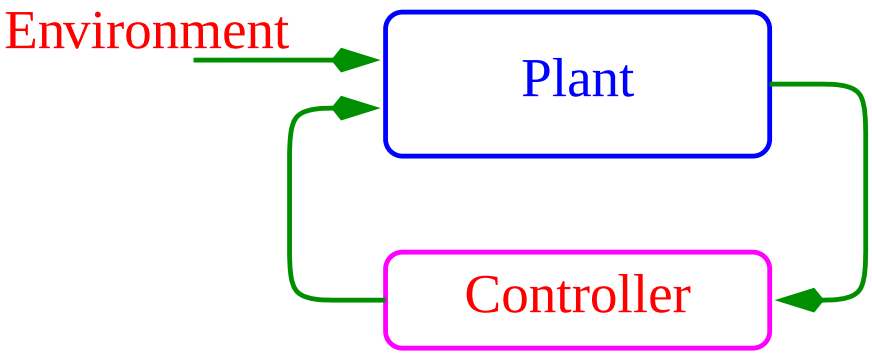
This has been investigated in the hardware-oriented European project PROSYD.

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.

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 uncontrollable.

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.

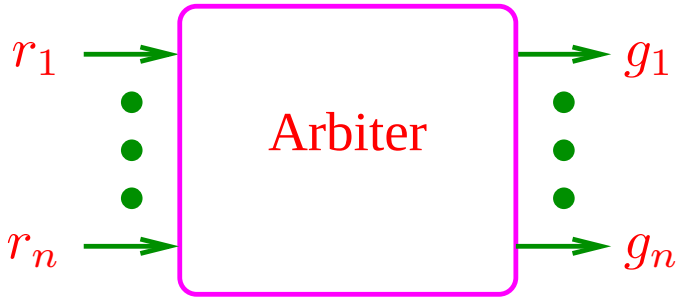
Required: Find a strategy for the controllable actions which will maintain a temporal specification against all possible adversary moves. Derive a program from this strategy. View as a two-persons game.

Apply to Programs

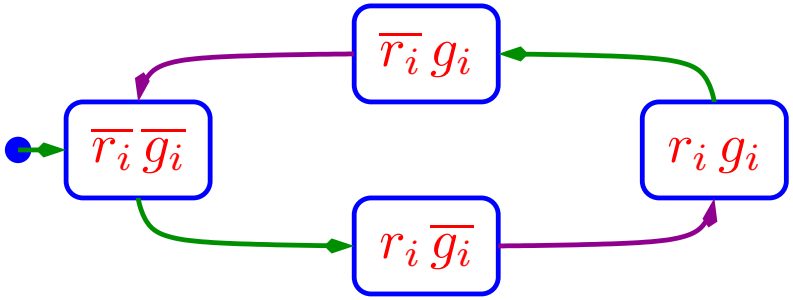
Let us apply the controller synthesis paradigm to synthesis of programs (or designs, in general).

Example Design: Arbiter

Consider a specification for an arbiter.



The protocol for each client:

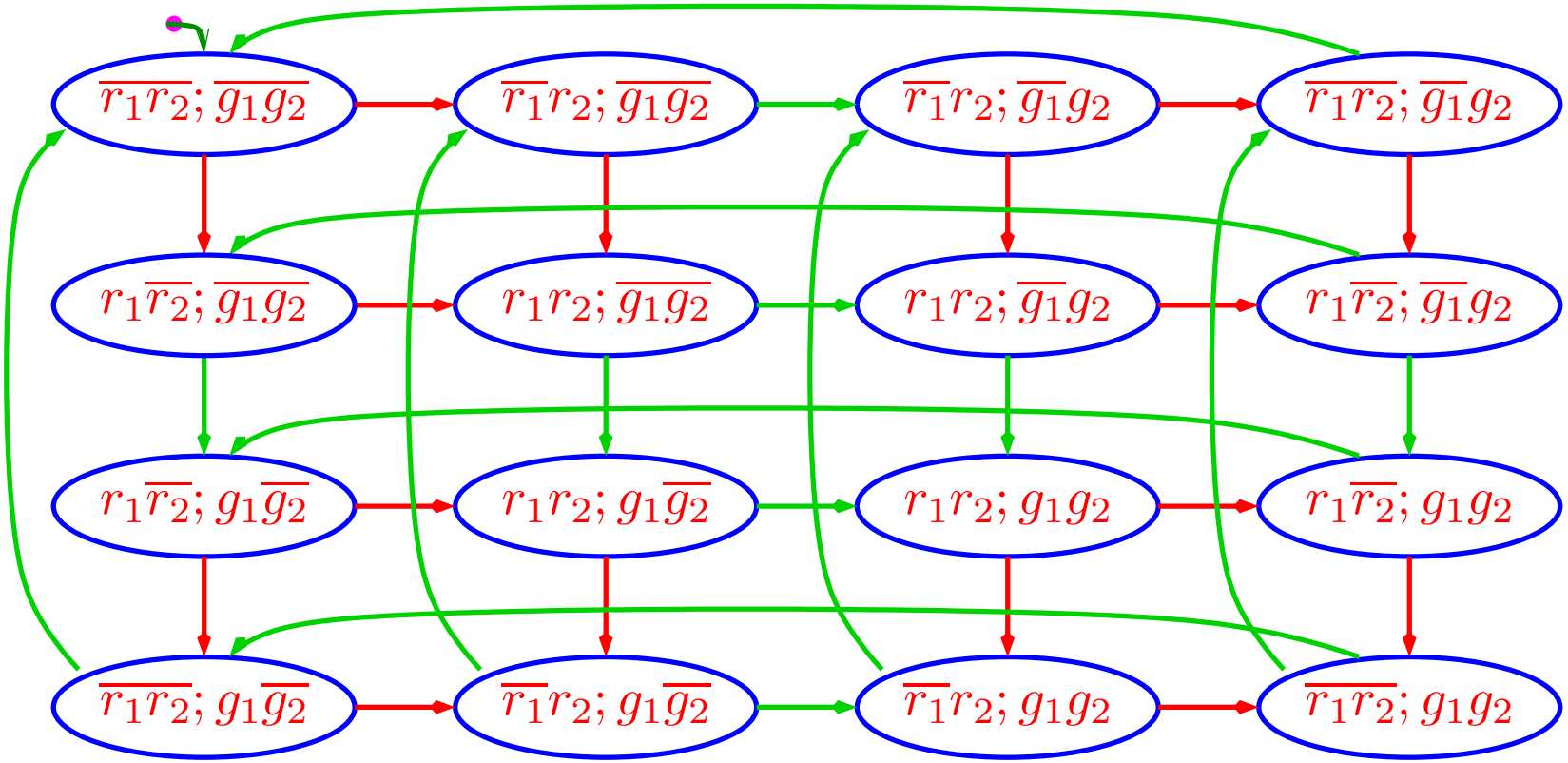


Required to satisfy

$$\bigwedge_{i \neq j} \Box\, \neg(g_i \wedge g_j) \qquad \wedge \qquad \bigwedge_{i} \Box\, \Diamond\, (g_i = r_i)$$

Start by Controller Synthesis

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



Due to idling, every node is connected to itself by both green and red transitions, which for simplicity we do not explicitly draw. A complete move consists of a red edge followed by a green edge. Also given is an LTL specification (winning condition):

$$\varphi: \quad \Box\,\neg(g_1 \wedge g_2) \;\wedge\; \Box\Diamond\,(g_1 = r_1) \;\wedge\; \Box\Diamond\,(g_2 = r_2)$$

For Simplicity

We added to the specification the requirement that, at every complete move, at most one of the four variables r_1, r_2, g_1, g_2 may change its value.

Controller Synthesis Via Game Playing

A game is given by $\mathcal{G} : \langle V = \vec{x} \cup \vec{y}, \Theta_1, \Theta_2, \rho_1, \rho_2, \varphi \rangle$, where

- $V = \vec{x} \cup \vec{y}$ are the state variables, with $\vec{x}$ being the environment's (player 1) variables, and $\vec{y}$ 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.
- $\Theta_1(\vec{x})$ — the initial condition for player 1 (Environment). An assertion characterizing the environment's initial states.
- $\Theta_2(\vec{x}, \vec{y})$ — the initial condition for player 2 (system).
- $\rho_1(\vec{x}, \vec{y}, \vec{x}')$ — Transition relation for player 1 (Environment).
- $\rho_2(\vec{x}, \vec{y}, \vec{x}', \vec{y}')$ — 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 $\rho_1(s_1[V], s_2[\vec{x}])$ and $\rho_2(s_1[V], s_2[V])$ are true.

We denote by $D_{\vec{x}}$ and $D_{\vec{y}}$ the domains of variables $\vec{x}$ and $\vec{y}$, respectively.

Plays and Strategies

Let $\mathcal{G} : \langle V, \Theta_1, \Theta_2, \rho_1, \rho_2, \varphi \rangle$ be a game. A play of $\mathcal{G}$ is an infinite sequence of states

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

satisfying the requirement:

- 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 $\sigma_1 : \Sigma^+ \mapsto D_{\vec{x}}$, which determines the next set of values for $\vec{x}$ 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 $j \geq 0$, $s_{j+1}[\vec{x}] = \sigma_1(s_0, \ldots, s_j)$.

Strategy σ_1 is winning for player 1 from state s if all s-originated plays (i.e., plays $\pi : s = s_0, s_1, \ldots$) 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 $\sigma_2 : \Sigma^+ \times D_{\vec{x}} \mapsto D_{\vec{y}}$.

From Winning Games to Programs

A game $\mathcal{G}$ is said to be winning for player 2 if, every $\vec{x}$-interpretation ξ satisfying $\Theta_1(\xi) = 1$ can be matched by a $\vec{y}$-interpretation η satisfying $\Theta_2(\xi, \eta) = 1$, such that the state $\langle \vec{x} : \xi,\ \vec{y} : \eta \rangle$ is winning for 2.

Otherwise, i.e., there exists an interpretation $\vec{x} : \xi$ satisfying $\Theta_1(\xi)$ such that, for all interpretations $\vec{y} : \eta$ satisfying $\Theta_2(\xi, \eta)$, the state $\langle \vec{x} : \xi,\ \vec{y} : \eta \rangle$ is winning for 1, the game is winning for player 1.

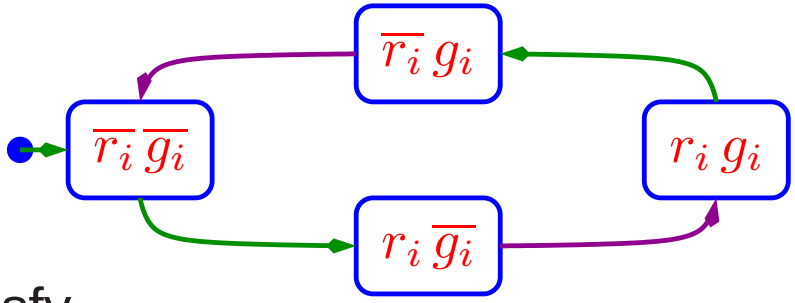
We solve the game, attempting to decide whether the game is winning for player 1 or 2. If it is winning for player 1 the specification is unrealizable. If it is winning for player 2, we can extract a winning strategy which is a working implementation.

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 Game for the Sample Specification

For the sample specification in which the client-server protocol is:



and was required to satisfy

$$\bigwedge_{i \neq j} \Box\, \neg(g_i \wedge g_j) \qquad \wedge \qquad \bigwedge_{i} \Box\, \Diamond\, (g_i = r_i)$$

We take the following game structure:

$$\begin{aligned}
\vec{x} \cup \vec{y}: &\quad \{r_i \mid i = 1, \ldots, n\} \cup \{g_i \mid i = 1, \ldots, n\} \\
\Theta_1: &\quad \textstyle\bigwedge_i \overline{r_i} \qquad\qquad \Theta_2: \quad \bigwedge_i \overline{g_i} \\
\rho_1: &\quad \textstyle\bigwedge_i \left((r_i \neq g_i) \rightarrow (r_i' = r_i)\right) \\
\rho_2: &\quad \textstyle\bigwedge_i \left((r_i = g_i) \rightarrow (g_i' = g_i)\right) \\
\varphi: &\quad \left(\textstyle\bigwedge_{i \neq j} \neg(g_i \wedge g_j)\right) \wedge \bigwedge_i \Box\Diamond\, (g_i = r_i)
\end{aligned}$$

Note that while some parts of the specification are placed in the components ($\Theta_1, \Theta_2, \rho_1, \rho_2$), exclusion and liveness are relegated to the winning condition.

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 **EX**p satisfied by all states which have a p-state as a successor.

For synthesis, we use the controlled predecessor operator $\boxdot p$. Its semantics can be defined by

$$\boxdot p: \quad \forall \vec{x}' : \rho_1(V, \vec{x}') \rightarrow \exists \vec{y}' : \rho_2(V, V') \wedge p(V')$$

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

In our graphical notation, $s \models \boxdot p$ iff s has at least one green p-successor, and all red successors different from s satisfy p.

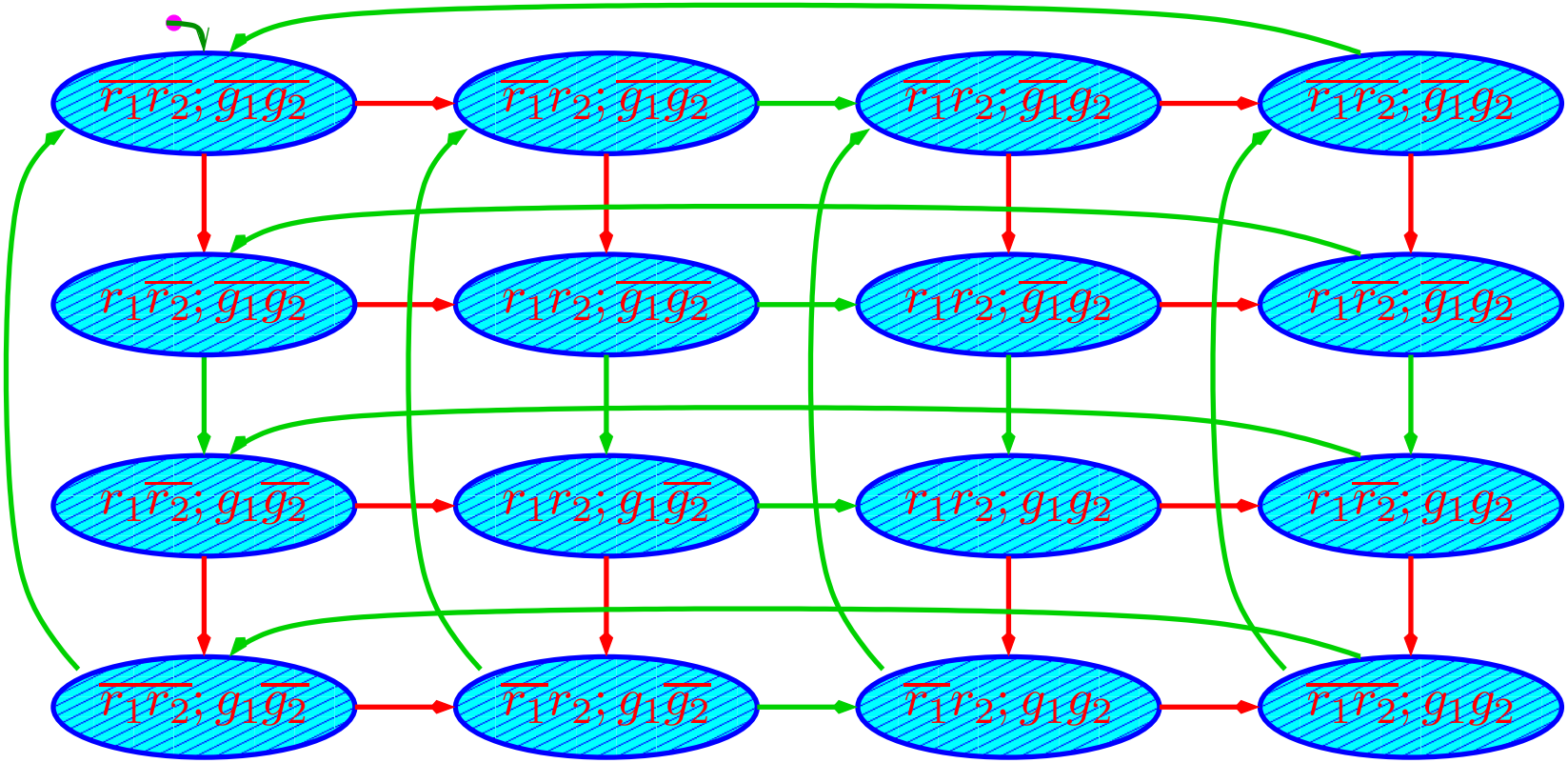
That is, for all possible moves of the red player, the green player can force a visit to a p-state.

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

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

$$\nu Y.\, p \;\wedge\; \bigcirc Y \quad = \quad 1 \wedge p \wedge \bigcirc p \wedge \bigcirc\bigcirc p \wedge \cdots$$

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



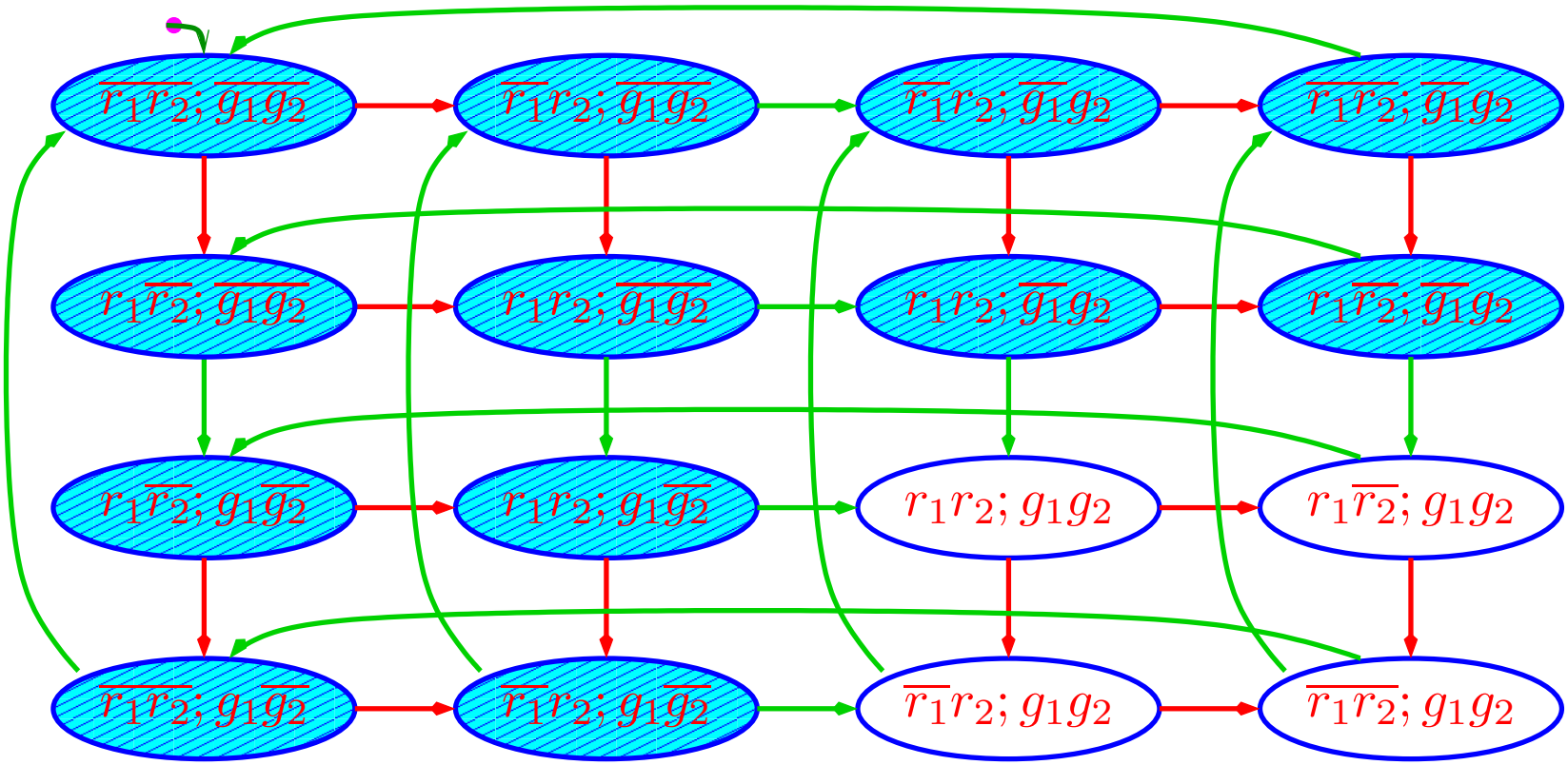
Iteration 0, $\quad Y_0 : 1$

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

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

$$\nu Y.\, p \;\wedge\; \Box Y \quad = \quad 1 \wedge p \wedge \Box p \wedge \Box\Box p \wedge \cdots$$

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



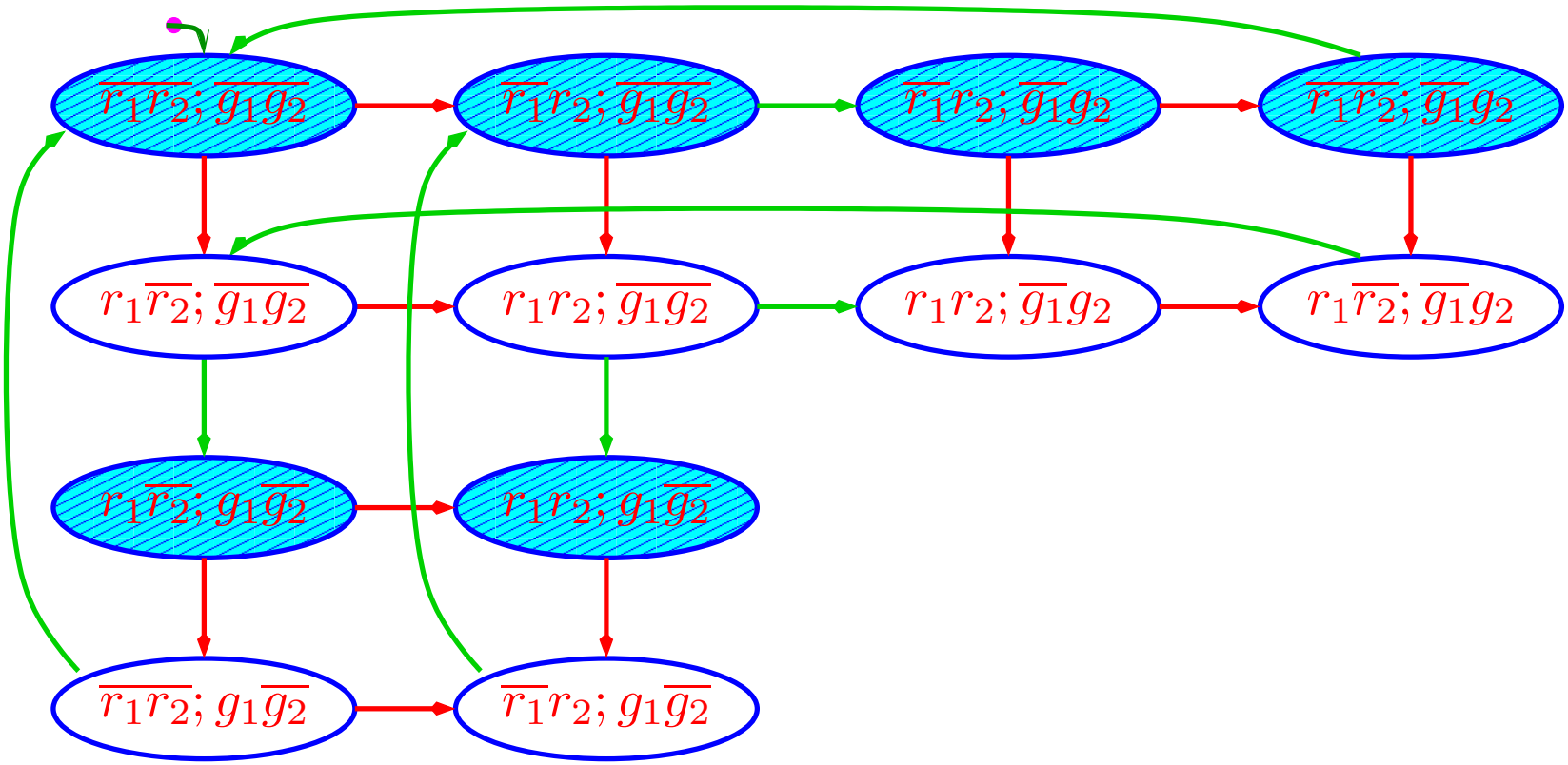
Iteration 1, $\quad Y_1 : \neg(g_1 \wedge g_2) \wedge \Box 1$

Solving $\Diamond q$ Games, Iteration 1

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

$$\mu Y.\, q \ \vee \ \Diamond Y \quad = \quad q \vee \Diamond q \vee \Diamond\Diamond q \vee \cdots$$

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



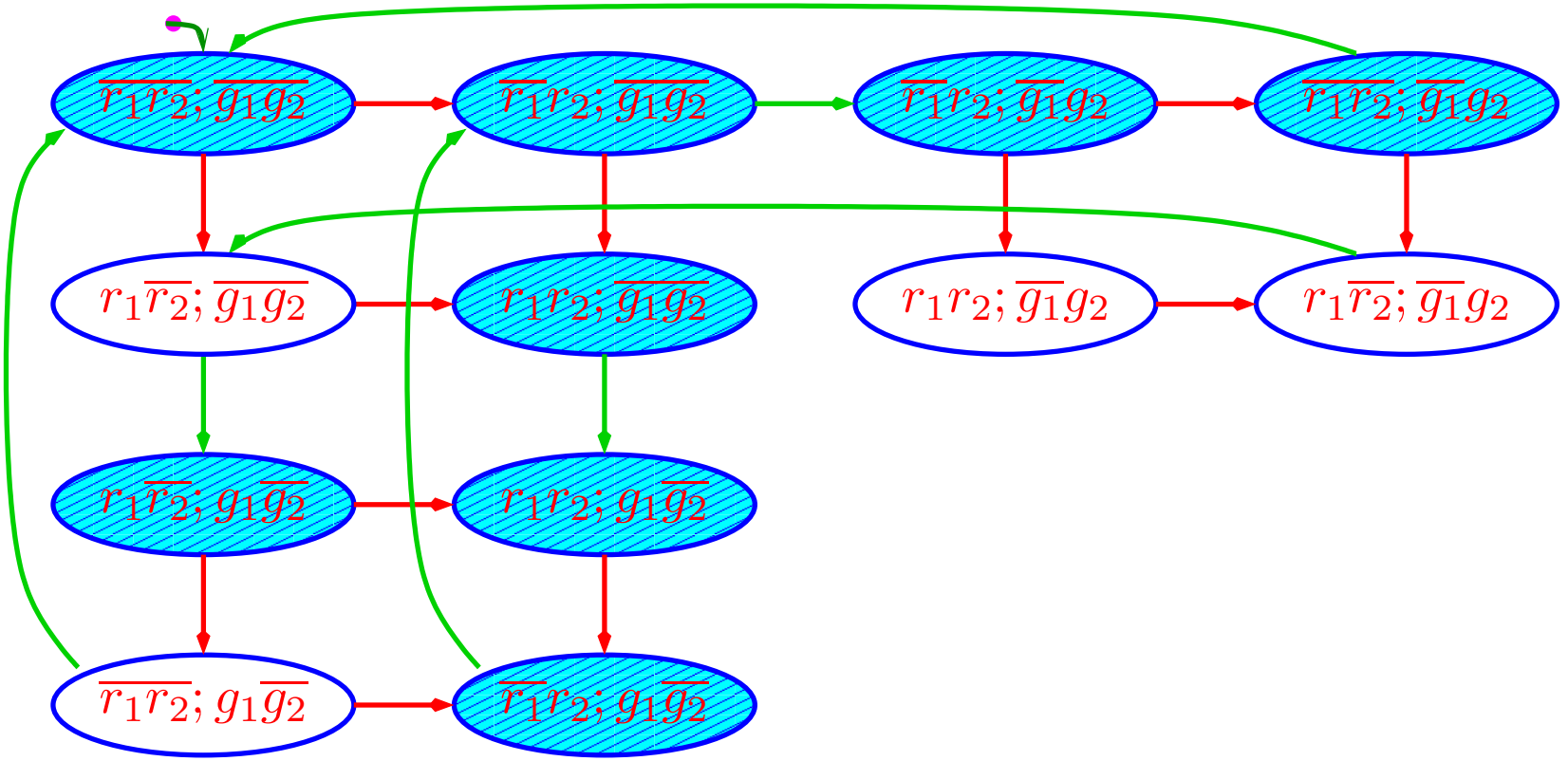
Iteration 1, $\quad Y_1 : (g_1 = r_1)$

Solving $\Diamond q$ Games, Iteration 2

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

$$\mu Y.\, q \vee \Box Y \quad = \quad q \vee \Box q \vee \Box\Box q \vee \cdots$$

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



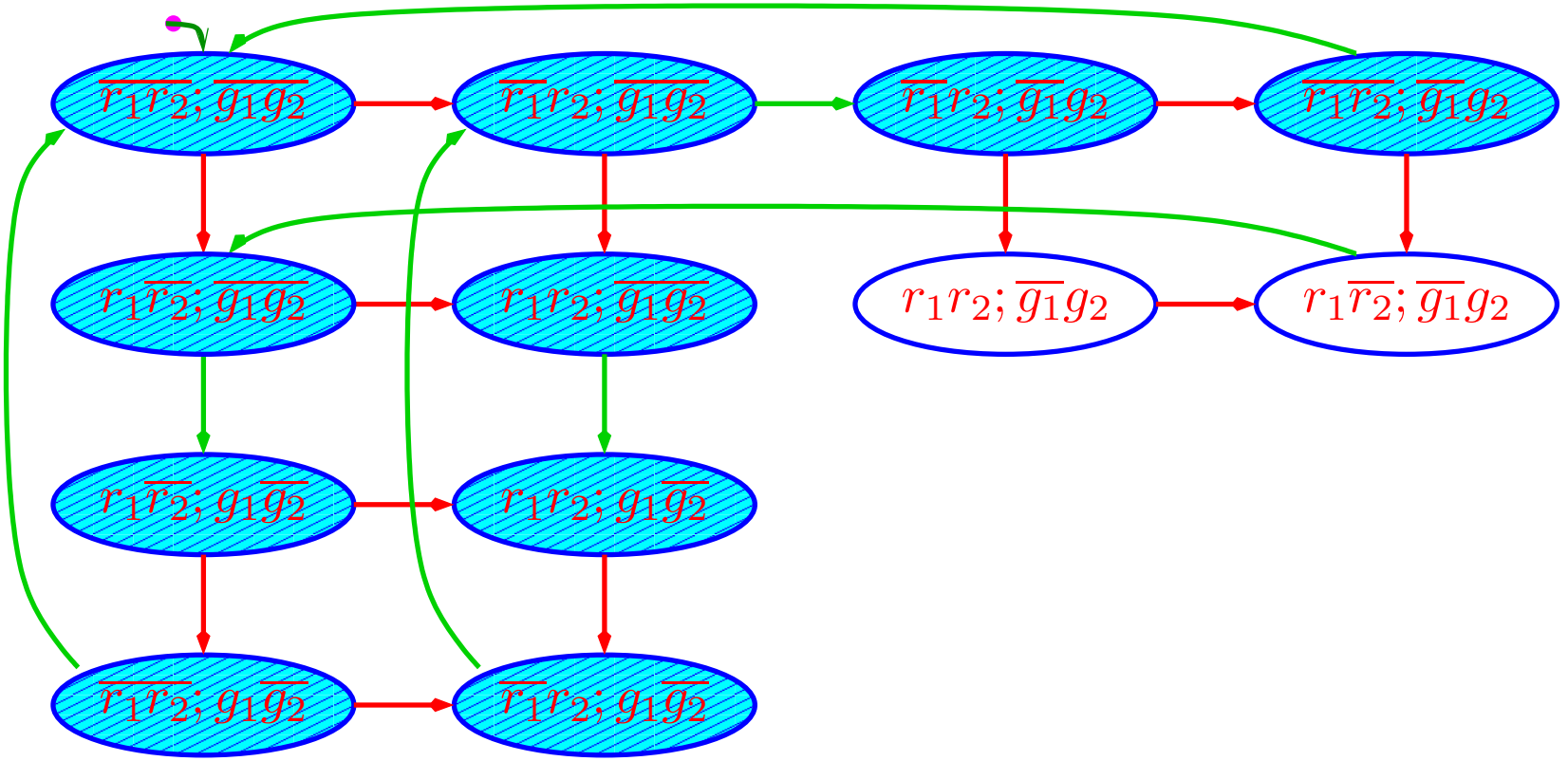
Iteration 2, $\quad Y_2 : Y_1 \vee \Box Y_1$

Solving $\Diamond q$ Games, Iteration 3

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

$$\mu Y.\, q \ \vee \ \Box Y \quad = \quad q \vee \Box q \vee \Box\Box q \vee \cdots$$

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



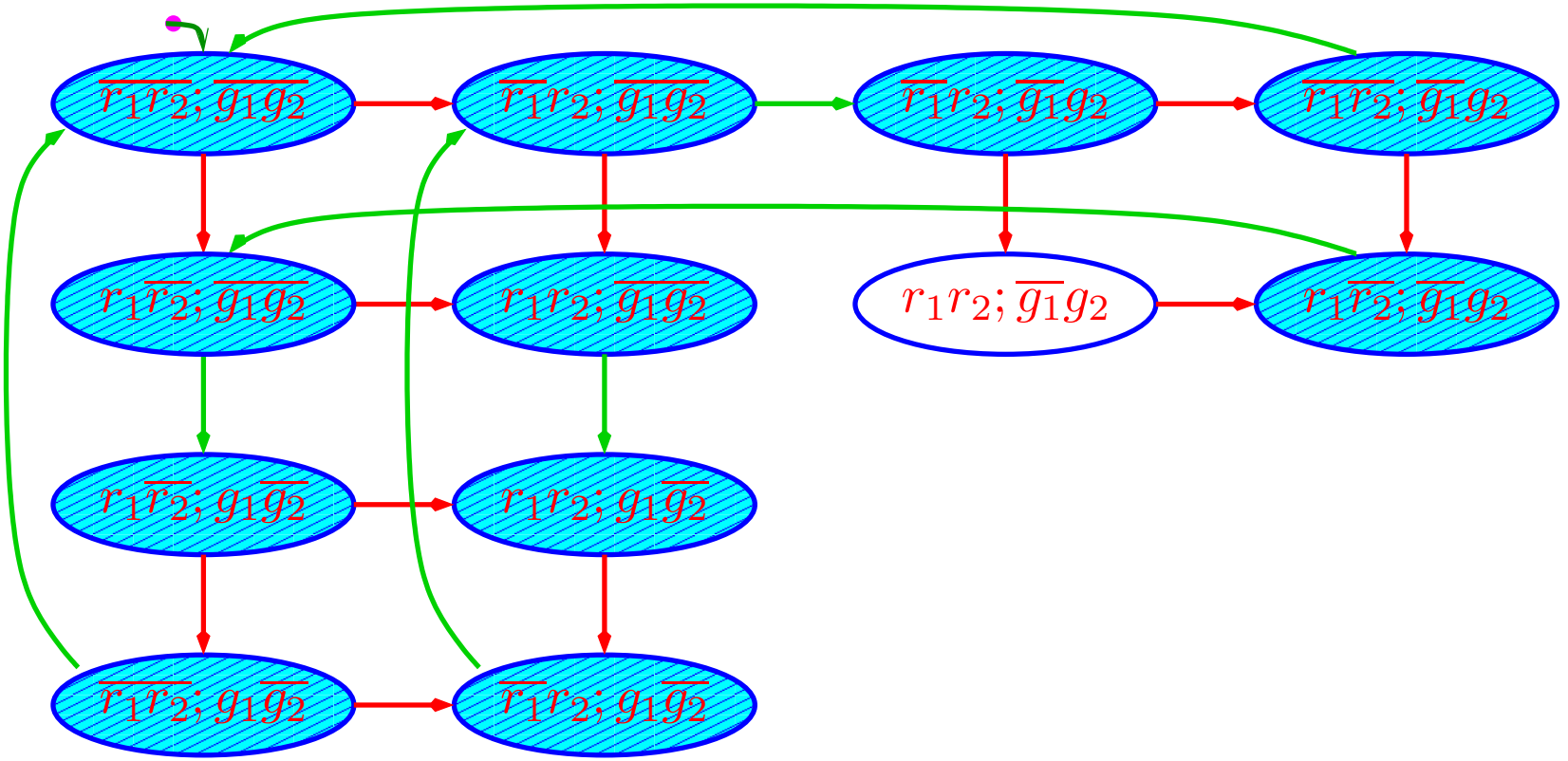
Iteration 3, $\quad Y_3 : Y_2 \vee \Box Y_2$

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

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

$$\mu Y.\, q \ \vee \ \diamondsuit Y \quad = \quad q \vee \diamondsuit q \vee \diamondsuit\diamondsuit q \vee \cdots$$

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



Iteration 4, $\quad Y_4 : Y_3 \vee \diamondsuit Y_3$

Solving $\Box\Diamond q$ Games

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

$$\nu Z \mu Y. q \wedge \Diamond Z \vee \Diamond Y$$

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

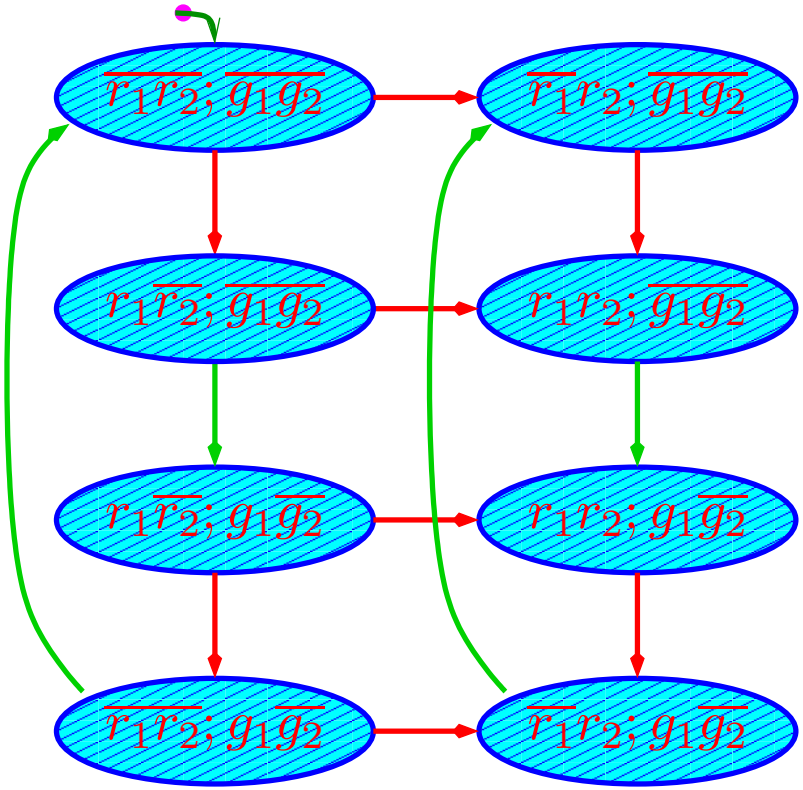
$$Z \quad = \quad \mu Y. (q \wedge \Diamond Z) \vee \Diamond Y$$

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

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

Solving $\Box\Diamond(g_1 = r_1)$ 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

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

Generalized Response (Büchi)

Solving the game for $\Box\Diamond q_1 \wedge \cdots \wedge \Box\Diamond q_n$.

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

Iteratively:

For $(i \in 1..n)$ **do** $[Z[i] := 1]$
Fix $(Z[1])$
$\Big[$ **For** $(i \in 1..n)$ **do**
$\Big[\; Y := 0$
$\;$ **Fix** (Y)
$\;\;\big[Y := (q[i] \wedge \bigcirc Z[i \oplus_n 1]) \vee \bigcirc Y\big]$
$\;Z[i] := Y \;\Big]\Big]$
Return $Z[1]$

Specification is Unrealizable

Applying the above algorithm to the specification

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

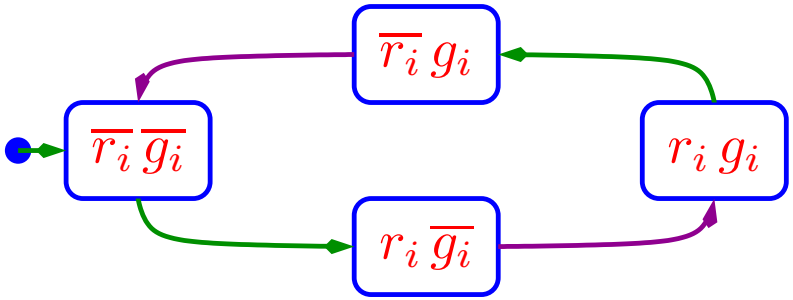
we find that it fails. Conclusion:

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.

A Realizable Specification

Consider a specification consisting of the protocol:



and the temporal specification

$$\bigwedge_{i \neq j} \Box\, \neg(g_i \wedge g_j) \quad \wedge \quad \left(\bigwedge_i \Box\Diamond \neg(r_i \wedge g_i) \quad \rightarrow \quad \bigwedge_i \Box\Diamond (g_i = r_i) \right)$$

We take the following game components:

$$\begin{array}{rl} \vec{x} \cup \vec{y}: & \{r_i \mid i = 1, \ldots, n\} \cup \{g_i \mid i = 1, \ldots, n\} \\ \Theta_1: & \bigwedge_i \overline{r_i} \qquad\qquad \Theta_2: \quad \bigwedge_i \overline{g_i} \\ \rho_1: & \bigwedge_i \left((r_i \neq g_i) \rightarrow (r_i' = r_i)\right) \\ \rho_2: & \bigwedge_{i \neq j} \neg(g_i' \ \wedge \ g_j') \quad \wedge \quad \bigwedge_i \left((r_i = g_i) \rightarrow (g_i' = g_i)\right) \\ \varphi: & \bigwedge_i \Box\Diamond \neg(r_i \wedge g_i) \quad \rightarrow \quad \bigwedge_i \Box\Diamond (g_i = r_i) \end{array}$$

Solving in Polynomial Time a Doubly Exponential Problem

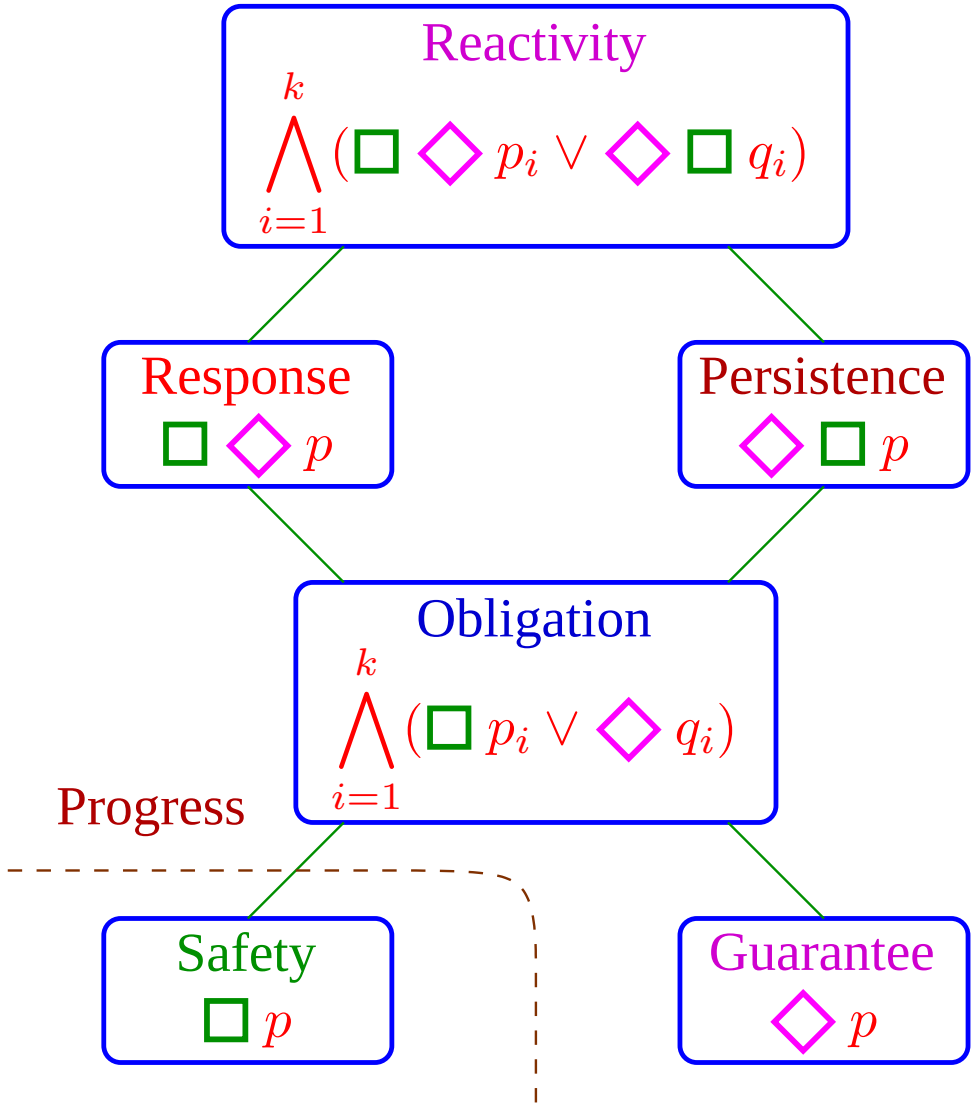
Recall that in [1989] Roni Rosner provided a general solution to the Synthesis problem. He showed that any approach that starts with the standard translation from LTL to Büchi automata, has a doubly exponential lower bound.

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

$$(\Box\Diamond p_1 \wedge \Box\Diamond p_2 \wedge \cdots \wedge \Box\Diamond p_m) \rightarrow \Box\Diamond q_1 \wedge \Box\Diamond q_2 \wedge \cdots \wedge \Box\Diamond q_n$$

This class of properties is bigger than the properties specifiable by deterministic Büchi automata. 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 React[1] game can be computed by

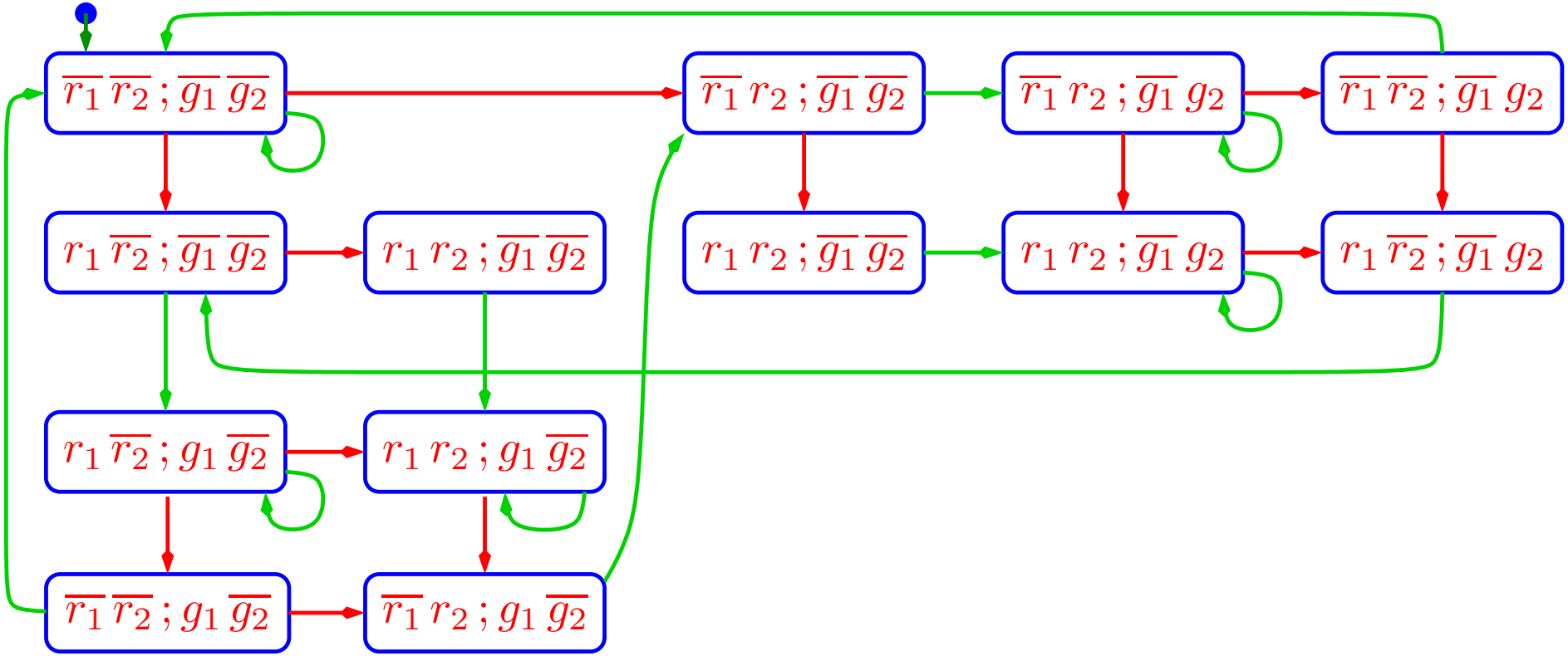
$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{j=1}^{m} \nu X (q_1 \wedge \boxdot Z_2 \vee \boxdot Y \vee \neg p_j \wedge \boxdot X) \right) \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X (q_2 \wedge \boxdot Z_3 \vee \boxdot Y \vee \neg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

where

$$\boxdot \varphi : \quad \forall \vec{x}' : \rho_1(V, \vec{x}') \rightarrow \exists \vec{y}' : \rho_2(V, V') \wedge \varphi(V')$$

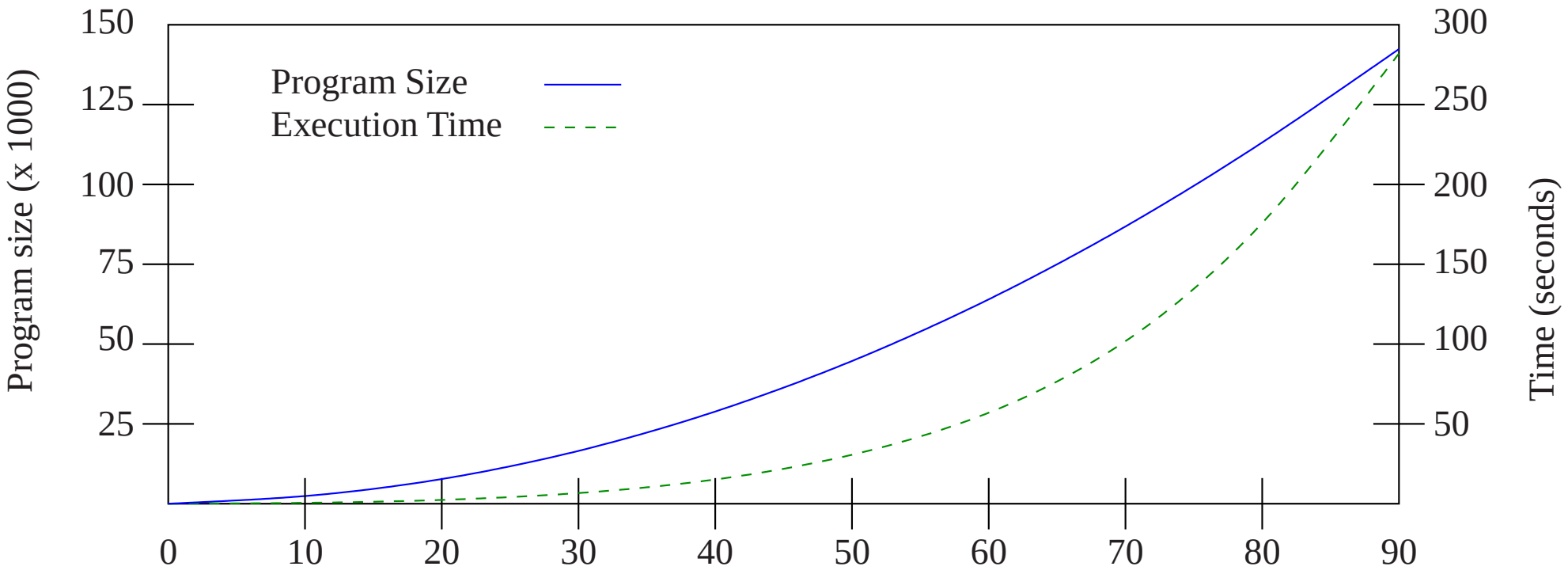
Results of Synthesis

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 methodology of property-based system design (Prosyd) is an option worth considering. It is greatly helped by improved algorithms for synthesis.

What about Distributed Systems?

One of the pessimistic results of Rosner's thesis [PR90] is that the synthesis problem for distributed systems is undecidable.

The problem comes from the fact that each process in the system has only partial knowledge of the full system state.

It can be algorithmically solved for very restricted types of architectures.

What will Happen if You all Agree to Buy the Synthesis Approach to System Development?

This opens the gate to a horde of new questions:

- What is the recommended style of Programming in a Specification Language? The controversy between
  - Model based Abstract State Machines, e.g., TLA, vs.
  - Property based.

Beware: Programming in Logic may Result in Very Inefficient Implementations

Recall the difference between logic programming and Prolog.

A naive specification of a sorting algorithm will consist of

$$\mathit{sorted}(y[1..N]) \ \wedge \ \mathit{permute}(x[1..N], y[1..N])$$

and may synthesize the code that enumerates all possible permutations and check each of them for being sorted.

On the other hand, a different representation of the specification which is logically equivalent to the above may produce *Quicksort* as a synthesized implementation.