Deductive and Algorithmic Methods for Formal Verification

Ph.D Research Proposal by: Yaniv Sa'ar

Supervisor: Prof. Amir Pnueli

DEPARTMENT OF COMPUTER SCIENCE AND APPLIED MATHEMATICS,
WEIZMANN INSTITUTE OF SCIENCE

October 30, 2007

1 Introduction

There are two methods that are widely used in order to formally prove a property of a given design implementation. The *deductive methods* (e.g. [MP91c]), schematically present proof rules consisting of a list of premises and a conclusion. Typically, the premises are assertions (state formulas) and the conclusion is temporal. For given auxiliary assertion, ranking function, and a proof for the premises, the user can conclude the validity of the property. On the other hand there are *model checking methods* (e.g. [KPR98], [KPRS02]). Model checking systematically explore all reachable states and possible computations, and check that none of them can falsify the property.

Model checking techniques have the advantage of being fully automatic and requiring no strong familiarity with the design being analyzed. Another important feature which makes model checking appealing, is the automatic generation of counter example in case of failure. On the down side, since all model checking algorithms are exponential in the size of the designs, their applicability is limited to designs whose size is not too big. The emergence of symbolic methods to implement model checking has dramatically improved the scale of designs being verified. However, we still have a long way to go in order to consider large industrial scale designs.

Deductive techniques employ symbolic representation, and thus do not suffer directly from model checking state explosion. Avoiding the size limitation makes deductive methods much more attractive for designs of a larger scale. However deductive methods too suffer from fundamental weaknesses. The main disadvantage of deductive methods is the fact that they demand user ingenuity to provide appropriate auxiliary assertions, and ranking functions (as shown in an example in [MP95]). Another disadvantage is that deductive methods do not produce a counter example. It follows that also Deductive methods still have much to do in order to improve user usability and accessibility.

Meanwhile, another innovative approach has recently won increasing attention. Instead of implementing a design, devising a specification, and struggling with either of the described above techniques to prove its correctness, *synthesis* offer a new tempting alternative to verification. Given

a property, synthesis promises to produce an implementation of the design which will satisfy the property.

In spite of the double exponential lower bound established in [PR89] for synthesis, there have been few attempts over the years to overcome this hopelessly high complexity. These were usually based on consideration of a restricted, simpler, or partial fragments of LTL. Thus, it has been shown that the synthesis problem can be solved in polynomial time (e.g. [AMPS98], [AT04]). Our recent results in [PPS06], have shown that there exists a polynomial time solution (of complexity N^3) for many expressive specifications of hardware designs belonging to the class of *Generalized Reactivity*[1] formulas (GR[1]), i.e. formulas of the form

$$(\Box \diamondsuit p_1 \wedge \cdots \wedge \Box \diamondsuit p_m) \quad \rightarrow \quad (\Box \diamondsuit q_1 \wedge \cdots \wedge \Box \diamondsuit q_n) \tag{1}$$

In this thesis we propose to focus on the following topics:

- A new deductive rule for liveness under compassion (strong fairness), and its implementation in PVS. A paper summarizing our preliminary results on this topic have been submitted to the VMCAI'08 conference (and is added as an appendix to the proposal).
- An algorithm for the synthesis of GR[1] specifications ([PPS06]), followed by:
    1. Optimizing both the algorithm of construction, and the resulting automata.
    2. Devising a better reasoning with the strategy being implemented, leading to a better diagnostic tool for the user of this methodology.

The proposal is organized as follows. In Section 2 we introduce several preliminary notions. We first present the computational model of fair discrete system with its related notions of fairness. We then view the problem of realizability and synthesis through the perspective of games. In Section 3 we discuss the various approaches for proving liveness properties under compassion (strong fairness). We start by presenting the legacy deductive rule [MP91a] for proving response properties. At the second half of Section 3 we present a new rule which has been developed as part

of this thesis, while discussing its innovative direct approach to handle compassion requirements. Finally, in Section 4 we show how to solve synthesis games, and discus future improvements in order to scale the algorithm to deal with larger designs.

2 Preliminaries

As a general note, we encourage the reader to review the fragment of *Linear Temporal Logic* (LTL), which we adopt as a specification language across this proposal. ([MP91b], [MP91c], [MP95])

2.1 Computational Model

As a computational model for reactive systems we take the model of *fair discrete systems* (FDS) [KP00]. A system $\mathcal{D} : \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ consists of the following components:

- $V = \{u_1, \ldots, u_n\}$: A finite set of Boolean variables. We define a *state* s to be an interpretation of V. For a state s and a system variable $u_i \in V$, we denote by $s[u_i]$ the value assigned to u_i by the state s. Let Σ denote the set of all states over V. An *assertion* is a first-order formula over V. A state s satisfies an assertion φ denoted $s \models \varphi$, if $s[\varphi] = \mathbf{true}$. We say that s is a φ-*state* if $s \models \varphi$.
- Θ: The *initial condition*, an assertion characterizing the initial states.
- ρ: A *transition relation*, an assertion $\rho(V, V')$, relating a state $s \in \Sigma$ to its $\mathcal{D}$-successor state $s' \in \Sigma$. We require that every state $s \in \Sigma$ has at least one $\mathcal{D}$-successor. This is often ensured by including in ρ the *idling* disjunct $V = V'$ (also called the *stuttering* step).
- $\mathcal{J}$: A set of *justice* (*weak fairness*) *requirements*. Each justice requirement $J \in \mathcal{J}$ is an assertion, intended to guarantee that every computation contains infinitely many J-states.
- $\mathcal{C}$: A set of *compassion* (*strong fairness*) *requirements*. Each compassion requirement is a pair $\langle p, q \rangle \in \mathcal{C}$ of assertions, intended to guarantee that every computation containing infinitely many p-states also contains infinitely many q-states.

Let $\mathcal{D}$ be a FDS as above. We define a *computation* of $\mathcal{D}$ to be an infinite sequence of states σ: $s_0, s_1, s_2, ...$, satisfying the following requirements:

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j = 0, 1, ...$, state s_{j+1} is a $\mathcal{D}$-successor of s_j. I.e., $\langle s_j, s_{j+1} \rangle \models \rho(V, V')$ where, for each $v \in V$, we interpret v as $s_j[v]$ and v' as $s_{j+1}[v]$.
- *Justice:* For each $J \in \mathcal{J}$, σ contains infinitely many occurrences of J-states
- *Compassion:* For each $\langle p, q \rangle \in \mathcal{C}$, if σ contains infinitely many occurrences of p-states, it must also contain infinitely many occurrences of q-states.

We say that FDS $\mathcal{D}$ *implements* specification φ if every computation of $\mathcal{D}$ satisfies φ.

We say that a FDS is a *fairness-free* FDS, if it has no fairness requirements. A system with no justice requirements is called a *compassionate discrete system* (CDS). Any FDS can also be easily converted to an equivalent CDS by simply converting $J \in \mathcal{J}$ to the equivalent requirement $\langle 1, J \rangle \in \mathcal{C}$. A system with no compassion requirements is called a *just discrete system* (JDS). Any FDS can be converted to an equivalent JDS which is exponential in the size of compassion requirements ([Cho74], [KPP05]).

2.2 Realizability and Synthesis

Assume two sets of variables $\mathcal{X}$ and $\mathcal{Y}$. Intuitively $\mathcal{X}$ is the set of input variables controlled by the environment, and $\mathcal{Y}$ is the set of system variables. With no loss of generality, we assume that all variables are Boolean. Given a specification φ, *realizability* amounts to checking whether there exists an *open controller* that satisfies φ. Such a controller can be represented as an automaton which, at any step, inputs values of the $\mathcal{X}$ variables and outputs values for the $\mathcal{Y}$ variables. *Synthesis* offer the construction of such a controller.

Realizability for LTL specifications is 2EXPTIME-complete [PR90]. We are interested in a subset of LTL for which we can solve realizability and synthesis in polynomial time. The speci-

fications we consider are of the form $\varphi = \varphi_e \rightarrow \varphi_s$. We require that φ_α for $\alpha \in \{e, s\}$ can be rewritten as a conjunction of the following parts.

- φ_i^α - a Boolean formula which characterizes the initial states of the implementation.
- φ_t^α - a formula of the form $\bigwedge_{i \in I} \Box B_i$ where each B_i is a Boolean combination of variables from $\mathcal{X} \cup \mathcal{Y}$ and expressions of the form $\bigcirc v$ where $v \in \mathcal{X}$ if $\alpha = e$, and $v \in \mathcal{X} \cup \mathcal{Y}$ otherwise.
- φ_g^α - a formula of the form $\bigwedge_{i \in I} \Box \Diamond B_i$ where each B_i is a Boolean formula.

It turns out that most of the specifications written in practice can be rewritten or converted to this format.

2.2.1 Game Structures [GTW02]

We reduce the realizability problem of a LTL formula to the decision of winner in games. We consider two-player game played between a system and an environment. The goal of the system is to satisfy the specification regardless of the actions of the environment.

A *game structure* (GS) $G : \langle V, \mathcal{X}, \mathcal{Y}, \Theta_e, \Theta_s, \rho_e, \rho_s, \varphi \rangle$ consists of the following component.

- $V = \{u_1, \ldots, u_n\}$: A finite set of typed *state variables* over finite domains. We define a *state* s to be an interpretation of V . For a state s and a variable $u_i \in V$, we denote by $s[u_i]$ the value assigned to u_i by the state s. Let Σ denote the set of all states over V. An *assertion* is a Boolean formula over V. A state s satisfies an assertion φ denoted $s \models \varphi$, if $s[\varphi] = \mathbf{true}$. We say that s is a φ-state if $s \models \varphi$.
- $\mathcal{X} \subseteq V$: A set of *input variables*. These are variables controlled by the environment. Let X denote the possible valuations to variables in $\mathcal{X}$.
- $\mathcal{Y} = V \setminus \mathcal{X}$: A set of *output variables*. These are variables controlled by the system. Let Y denote the possible valuations for the variables in $\mathcal{Y}$.

- Θ_e: The environment *initial condition*. An assertion over $\mathcal{X}$ characterizing the requirements of the environment for the initial states of G. A state s is called an *environment-initial state* if $s \models \Theta_e$.
- Θ_s: The system *initial condition*. An assertion over V characterizing the requirements of the system for the initial states of G. A state s is called *initial* if it satisfies Θ_s.
- $\rho_e(\mathcal{X}, \mathcal{Y}, \mathcal{X}')$: The *transition relation* of the environment. An assertion, relating a state $s \in \Sigma$ to a possible input value $\vec{x}' \in X$, by referring to the unprimed copies of $\mathcal{X}$ and $\mathcal{Y}$, and the primed copies of $\mathcal{X}$.
- $\rho_s(\mathcal{X}, \mathcal{Y}, \mathcal{X}', \mathcal{Y}')$: The *transition relation* of the system. This is an assertion, relating a state $s \in \Sigma$ and an input value $\vec{x}' \in X$ to an output value $\vec{y}' \in Y$, by referring to the primed and unprimed copies of V.
- φ: The winning condition, given by an LTL formula.

For two states s and s' of G, s' is a *successor* of s if $(s, s') \models \rho_e \wedge \rho_s$. A *play* σ of G is a maximal sequence of states $\sigma : s_0, s_1, \ldots$ satisfying *initiality* namely $s_0 \models \Theta$, and *consecution* namely, for each $j \geq 0$, s_{j+1} is a successor of s_j. Let G be a GS and σ be a play of G.

A play σ is *winning for the system* if it is infinite and satisfies φ. Otherwise, σ is *winning for the environment*. A *strategy* for the system is a partial function $f : \Sigma^+ \times X \mapsto Y$ such that if $\sigma = s_0, \ldots s_n$ then for every $\vec{x}' \in X$ such that $\rho_e(s_n, \vec{x}') = 1$ we have $\rho_s(s_n, \vec{x}', f(\sigma \cdot \vec{x}')) = 1$. Let f be a strategy for the system, and $s_0 \in \Sigma$. A play $s_0, s_1, \ldots$ is said to be *compliant* with strategy f if for all $i \geq 0$ we have $f(s_0, \ldots, s_i, s_{i+1}[\mathcal{X}]) = s_{i+1}[\mathcal{Y}]$, where $s_{i+1}[\mathcal{X}]$ and $s_{i+1}[\mathcal{Y}]$ are the restrictions of s_{i+1} to variable sets $\mathcal{X}$ and $\mathcal{Y}$, respectively. Strategy f is *winning* for the system from state $s \in \Sigma_G$ if all s-plays (plays departing from s) which are compliant with f are winning for the system. We denote by W_s the set of states from which there exists a winning strategy for the system. A *strategy* for player environment, *winning strategy*, and the *winning set* W_e are defined dually. A GS G is said to be *winning* for the system if every environment-initial state s_0 can be modified to an initial state $\widetilde{s}_0$ such that $\widetilde{s}_0 \in W_s$ and $\widetilde{s}_0[\mathcal{X}] = s_0[\mathcal{X}]$.

Given an LTL specification $\varphi_e \rightarrow \varphi_s$ as explained above and sets of input and output variables $\mathcal{X}$ and $\mathcal{Y}$ we construct a GS as follows. Let $\varphi_\alpha = \varphi_i^\alpha \wedge \varphi_t^\alpha \wedge \varphi_g^\alpha$ for $\alpha \in \{e, s\}$. Then, for Θ_e and Θ_s, we take φ_i^e and φ_i^s, respectively. Let $\varphi_t^\alpha = \bigwedge_{i \in I} \Box B_i$, then $\rho_\alpha = \bigwedge_{i \in I} \tau(B_i)$, where the translation τ replaces each instance of $\bigcirc v$ by v'. Finally, we set $\varphi = \varphi_g^e \rightarrow \varphi_g^s$. We *solve* the game, attempting to decide whether the game is winning for the environment or the system. If the environment is winning the specification is *unrealizable*. If the system is winning, we *synthesize* a winning strategy which is a *working implementation*, out of the system strategy.

3 Deductive Rule for Response under Compassion

The deductive theory of temporal verification presents several rules for proving typical temporal properties ([MP91a]). Each rule consists of a list of premises $\varphi_1, \ldots, \varphi_n$ and a temporal conclusion ψ. If premises $\varphi_1, \ldots, \varphi_n$ are successfully proved, the user can conclude the validity of the temporal property ψ (i.e. $\varphi_1 \wedge \ldots \wedge \varphi_n \rightarrow \psi$). Typically, $\varphi_1, \ldots, \varphi_n$ are assertions (i.e. non-temporal formulas), while the conclusion ψ is temporal. An exception to the typical case is rule F-WELL, where one of the premises is a response property by itself.

3.1 The Legacy Recursive Rule

We first present rule F-WELL, which is derived from the proof rule presented in [MP91a]. It is modified in order to represent the transition from the computational model of *fair transition systems* to that of *fair discrete systems*. The rule is presented in Fig. 1.

The FDS $(\mathcal{D} \backslash \{(p_i, q_i)\})$ is obtained by removing from $\mathcal{D}$ the compassion requirement (p_i, q_i). Thus, $(\mathcal{D} \backslash \{(p_i, q_i)\})$ has one compassion requirement less than $\mathcal{D}$. As seen, if some of the helpful requirements are compassionate, a recursive call is made (C4.). The recursive call establishes a similar temporal property over a system with fewer compassion requirements.

Rule F-WELL

For a well-founded domain $\mathcal{A} : (W, \succ)$,
assertions $p, q, \varphi_1, \ldots, \varphi_n$,,
fairness requirements $F_1, \ldots, F_n \in \mathcal{J} \cup \mathcal{C}$,,
and ranking functions $\Delta_1, \ldots, \Delta_n$ where each $\Delta_i : \Sigma \mapsto \mathcal{A}$

W1. $p \implies q \vee \bigvee_{j=1}^{n} \varphi_j$

For each $i = 1, \ldots, n$,

W2. $\varphi_i \wedge \rho \implies q' \vee (\varphi_i' \wedge \Delta_i = \Delta_i') \vee \left(\bigvee_{j=1}^{n} (\varphi_j' \wedge \Delta_i \succ \Delta_j')\right)$

W3. If $F_i = (p_i, q_i) \in \mathcal{C}$ then

C3. $\varphi_i \implies \neg q_i$

C4. $(\mathcal{D} \backslash \{(p_i, q_i)\}) \models (\varphi_i \implies \diamondsuit(p_i \vee \neg\varphi_i))$

Otherwise ($F_i = J_i \in \mathcal{J}$),

J3. $\varphi_i \implies \neg J_i$

$\mathcal{D} \models (p \implies \diamondsuit q)$

Figure 1: Deductive rule F-WELL

3.2 Flat Rule for Accessibility

In Fig. 2 we present proof rule FAIR-RESPONSE which establishes the response property $p \implies \diamondsuit q$ for a CDS $\mathcal{D}$. This is the first deductive rule for response property under compassion, in which all the premises are non-temporal and only the conclusion is temporal.

The new rule has been implemented in the theorem prover PVS [OSRSC99] as a part of the PVS-based temporal prover TLPVS [AP03]. To do so, we had to prove the soundness of the FAIR-RESPONSE rule within PVS. We had also apply the rule to prove the response property to few examples.

Not much work had been done in order to directly deal with compassion requirements. Usually, the compassion has been reduce away to a compassion free structure (i.e. by reducing the FDS to a JDS). This approach augment the structure with auxiliary variables, exponential in the size of compassion requirements ([Cho74]). The new rule enables the verification of a response property under compassion, without the additional price of these extra variables.

Rule FAIR-RESPONSE

For a well-founded domain $\mathcal{A} : (W, \succ)$,
assertions $p, q, \varphi_1, \ldots, \varphi_n$,
compassion requirements $(p_1, q_1), \ldots, (p_n, q_n) \in \mathcal{C}$,
and ranking functions $\Delta_1 , \ldots , \Delta_n$ where each $\Delta_i : \Sigma \mapsto W$

R1. $p \implies q \vee \bigvee_{j=1}^{n}(p_j \wedge \varphi_j)$

For each $i = 1, \ldots, n$,

R2. $p_i \wedge \varphi_i \wedge \rho \implies q' \vee \bigvee_{j=1}^{n}(p'_j \wedge \varphi'_j)$

R3. $\varphi_i \wedge \rho \implies q' \vee (\varphi'_i \wedge \Delta_i = \Delta'_i) \vee \bigvee_{j=1}^{n}\left(p'_j \wedge \varphi'_j \wedge \Delta_i \succ \Delta'_j\right)$

R4. $\varphi_i \implies \neg q_i$

$p \implies \diamondsuit q$

Figure 2: Deductive rule FAIR-RESPONSE

4 Design Synthesis

A challenging problem, which was first identified as Church's problem ([Chu63]), and recently received new interest ([PPS06]), is the problem of automatic synthesis of programs and designs from logical specifications. Several previous methods have been proposed for its solution e.g. [BL69], [Rab72], and has been considered again in the context of LTL specification in [CE81], [MW84], [PR89]. The two prevalent approaches to solving the synthesis problem were by reducing it to the emptiness problem of tree automata, and viewing it as the solution of a two-person game. In [PPS06] we have shown that there exists a polynomial time solution (N^3) for many expressive specifications of hardware designs from the class of GR[1].

4.1 Solving GR[1] Games ([PPS06])

For the usual construction and interpretation of μ-calculus formulas, we refer the reader to [Koz83], [EL86], [KPP05], [PPS06]. Let G: $\langle V, \mathcal{X}, \mathcal{Y}, \Theta_e, \Theta_s, \rho_e, \rho_s, \varphi \rangle$ be a GS, where φ is of the form:

$$\varphi = \bigwedge_{i=1}^{m} \square \diamondsuit J_i^1 \rightarrow \bigwedge_{j=1}^{n} \square \diamondsuit J_j^2$$

Here J_i^1 and J_j^2 are sets of Boolean formulas. [KPP05] refers to these as generalized Streett[1] games and provide the following μ-calculus formula to solve them. Let $j \oplus 1 = (j \ mod \ n) + 1$.

$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{i=1}^{m} \nu X (J_1^2 \wedge \bigcirc Z_2 \ \vee \ \bigcirc Y \ \vee \ \neg J_i^1 \wedge \bigcirc X) \right) \\ \mu Y \left(\bigvee_{i=1}^{m} \nu X (J_2^2 \wedge \bigcirc Z_3 \ \vee \ \bigcirc Y \ \vee \ \neg J_i^1 \wedge \bigcirc X) \right) \\ \vdots \\ \vdots \\ \mu Y \left(\bigvee_{i=1}^{m} \nu X (J_n^2 \wedge \bigcirc Z_1 \ \vee \ \bigcirc Y \ \vee \ \neg J_i^1 \wedge \bigcirc X) \right) \end{bmatrix} \tag{2}$$

Intuitively, for $j \in [1..n]$ and $i \in [1..m]$ the greatest fixpoint $\nu X (J_j^2 \wedge \bigcirc Z_{j\oplus 1} \vee \bigcirc Y \vee \neg J_i^1 \wedge \bigcirc X)$ characterizes the set of states from which the system can force the play either to stay indefinitely in $\neg J_i^1$ states (thus violating the left hand side of the implication) or in a finite number of steps reach a state in the set $J_j^2 \wedge \bigcirc Z_{j\oplus 1} \ \vee \ \bigcirc Y$. The two outer fixpoints make sure that the system wins from the set $J_j^2 \wedge \bigcirc Z_{j\oplus 1} \ \vee \ \bigcirc Y$. The least fixpoint μY makes sure that the unconstrained phase of a play represented by the disjunct $\bigcirc Y$ is finite and ends in a $J_j^2 \wedge \bigcirc Z_{j\oplus 1}$ state. Finally, the greatest fixpoint νZ_j is responsible for ensuring that, after visiting J_j^2, we can loop and visit $J_{j\oplus 1}^2$ and so on. By the cyclic dependence of the outermost greatest fixpoint, either all the sets in J_j^2 are visited or getting stuck in some inner greatest fixpoint, where some J_i^1 is visited only finitely many times.

[PPS06] continues to show that with the intermediate values in the computation of the fixpoints, we can construct a FDS that implements φ, thus solving the synthesis problem.

4.2 Picking and Minimizing the Synthesized Design

Formal verification methods are widely known for their extensive space complexity. Enumerated algorithm, which evaluate state by state, are considered to be hard by an order of a magnitude comparing to symbolic algorithms, which evaluate sets of states. The weakness of [PPS06] stems from its enumerated approach to the construction, and removal of redundancies. Although [PPS06] discusses symbolic minimization of the resulting FDS, it does not consider a symbolic algorithm

for the entire construction. It is yet an open question whether it is possible to directly construct a symbolic representation (the resulting FDS), out of the GR[1] game, without the need for an intermediate enumerated phase. In order to qualify synthesis as a front-end formal method solution, we still await the extra step which will allow us to consider at least medium scale designs. We believe that by avoiding all enumerated phases, [PPS06] can be enhanced to scale up the size of designs being considered.

Another important issue which is not fully addressed by [PPS06] is the choice of strategy to be implemented. [PPS06] collects the set of strategies, and when constructing the result, an arbitrary (single) strategy is picked to implement the design. The strategy is picked in a greedy step by step manner, without considering the resulting symbolic automata size, the readability of the resulting design, or any other factor. Another important question yet to be answered is whether we can wisely pick a strategy out of the set of strategies, which will optimize some of these factors.

References

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.

[AP03] T. Arons and A. Pnueli. Tlpvs: A pvs-based ltl verification system. In *Verification–Theory and Practice: Proceedings of an International Symposium in Honor of Zohar Manna's 64th Birthday*, Lect. Notes in Comp. Sci., pages 84–98. Springer-Verlag, 2003.

[AT04] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

[BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, volume 131 of *Lect. Notes in Comp. Sci.*, pages 52–71. Springer-Verlag, 1981.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Comp. Systems Sci.*, 8:117–141, 1974.

[Chu63] A. Church. Logic, arithmetic and automata. In *Proc. 1962 Int. Congr. Math.*, pages 23–25, Upsala, 1963.

[EL86] E. A. Emerson and C. L. Lei. Efficient model-checking in fragments of the propositional modal μ-calculus. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 267–278, 1986.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata logics, and infinite games: a guide to current research*, volume 2500 of *Lect. Notes in Comp. Sci.* Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP00] Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Inf. and Comp.*, 163:203–243, 2000.

[KPP05] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. *Inf. and Comp.*, 200(1):36–61, 2005.

[KPR98] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In K.G. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Int. Colloq. Aut. Lang. Prog.*, volume 1443 of *Lect. Notes in Comp. Sci.*, pages 1–16. Springer-Verlag, 1998.

[KPRS02] Y. Kesten, A. Pnueli, L. Raviv, and E. Shahar. Ltl model checking with strong fairness. *Formal Methods in System Design*, 2002. Accepted.

[MP91a] Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.

[MP91b] Z. Manna and A. Pnueli. On the faithfulness of formal models. In *Mathematical Foundation of Computer Science*, volume 520 of *Lect. Notes in Comp. Sci.*, pages 28–42. Springer-Verlag, 1991.

[MP91c] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Prog. Lang. Sys.*, 6:68–93, 1984.

[OSRSC99] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380, 2006.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. Princ. of Prog. Lang.*, pages 179–190, 1989.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st IEEE Symp. Found. of Comp. Sci.*, pages 746–757, 1990.

[Rab72] M.O. Rabin. *Automata on Infinite Objects and Churc's Problem*, volume 13 of *Regional Conference Series in Mathematics*. Amer. Math. Soc., 1972.