

Logics in Computer Science

Fabio Mogavero

Università degli Studi di Napoli "Federico II"
<http://people.na.infn.it/mogavero>

Ph.D. Thesis Defense
Napoli, January 21, 2011

Our work

Main idea

We looked for logics extending some of the classic *temporal logics* used in computer science, with the aim to augment their *expressive power* without increasing the *complexity* of their *decision problems*.

Areas of research

Our research fall in the two, so called, areas of *Logics for Computations* and *Logics for Strategies*.

Application

Some of the introduced logics can be effectively used as *specification languages* for the *formal verification* and *synthesis* of systems.

Our presentation

We present two of our works, one for each area of research!

Logics for Computations:

- *Graded Computation Tree Logics.*

Logics for Strategies:

- *Reasoning About Strategies.*

System correctness (I)

Let **S** be a system and **P** a desired behavior (specification).

Two very important problems

- **Model Checking**: Is **S** correct w.r.t. **P**?
- **Satisfiability**: Is **P** a correct specification?

To answer to these questions, formal methods are used.

- **S** can be modeled by a **labeled transition graph** \mathcal{K} (Kripke structure).
- **P** can be expressed as a **temporal logic formula** φ .

Then,

- **Model Checking**: Is \mathcal{K} a model of φ ($\mathcal{K} \models \varphi$)?
- **Satisfiability**: Is there a \mathcal{K} such that $\mathcal{K} \models \varphi$?

System correctness (II)

Verification as debugging: failure of verification identifies bugs.

- Both specifications and programs formalize informal requirements.
- Verification contrasts two independent formalizations.
- Failure of verification reveals inconsistency between formalizations.

Model checking: uncommonly effective debugging tool.

- Systematic exploration of the design state space.
- Good at catching difficult “limit cases”.

Satisfiability: useful to verify...

- the realizability of a specification;
- the non-triviality of a specification.



Part I

Logics for Computations

Classical system model

Key Idea: Systems can be represented as Kripke structures!

A *Kripke structure* $\mathcal{K} = \langle AP, W, R, L, w_0 \rangle$ is a labeled transition graph used to model system behaviors in a monolithic way:

- 1 AP : atomic propositions;
- 2 W : worlds represent system states;
- 3 $w_0 \in W$: designated initial world;
- 4 $R \subseteq W \times W$: edges represent system transitions;
- 5 $L : W \rightarrow 2^{AP}$: labels represent state properties.

The paths $\pi \in \text{Pth}(\mathcal{K})$ in the structure \mathcal{K} represent system executions.

Classical system specifications

Key Idea: Temporal logic allows the description of the ordering of events!

Two main families of temporal logics:

- **Linear-Time Temporal Logics (LTL)**
 - Each moment in time has a unique possible future.
 - LTL expresses path properties based on the paths state labels.
 - Useful for hardware specification.
- **Branching-Time Temporal Logics (CTL, CTL*, and μ CALCULUS)**
 - Each moment in time may split into various possible future.
 - CTL* expresses state properties based on the existence or universality of paths exiting from that state and satisfying LTL-like properties.
 - Useful for software specification.

Linear-time temporal logic

LTL: Linear Temporal Logic [Pnueli, '79]

■ $\psi ::= p \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid X\psi \mid \psi U \psi \mid \psi R \psi.$

Example

- $X\phi$: ϕ holds in the next state;
- $F\phi$: ϕ holds eventually;
- $G\phi$: ϕ holds forever;
- $\phi_1 U \phi_2$: ϕ_1 holds until ϕ_2 holds;
- $\phi_1 R \phi_2$: ϕ_2 holds forever or until ϕ_1 holds.

Branching-time temporal logics

CTL: Computation Tree Logic [Clarke & Emerson, '81]

CTL*: Full Computation Tree Logic [Emerson & Halpern, '86]

1 $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid E\psi \mid A\psi,$

2 $\psi ::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid X\psi \mid \psi U \psi \mid \psi R \psi.$

Example

- **EGEF** ψ : there is a computation for which, in each moment of its time, there is another computation satisfying eventually ψ ;
- **AFG** ψ : all computations eventually reach a point in their time from which ψ holds forever.

Computational complexities

	M.C.	Sat.
LTL	PSPACE-COMPLETE	PSPACE-COMPLETE
CTL	PTIME-COMPLETE	EXPTIME-COMPLETE
CTL*	PSPACE-COMPLETE	2EXPTIME-COMPLETE
μ CALCULUS	UPTIME \cap COUPTIME	EXPTIME-COMPLETE

Table: Computational complexity of Model Checking and Satisfiability.

Graded Computation Tree Logics

Facts

The μ CALCULUS subsumes many logics, in particular, LTL, CTL, and CTL*.

Several extension of μ CALCULUS have been considered.

One among all: the **G μ CALCULUS**, i.e., the μ CALCULUS extended with **graded modalities** (“there are at least n successors such that...”).

μ CALCULUS: very expressive but too low-level (hard to understand and use).

LTL, CTL, and CTL*: less expressive but much more human-friendly.

Motivations

A natural question: how could logics that allow to reason about paths be affected by considering graded modalities?

Why graded modalities on paths?

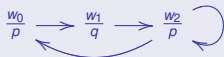
- XML query languages.
- Cyclomatic complexity.
- Redundancy in a system.
- Counting error counterexamples.

Our proposal

We investigate an extension of CTL with graded modalities (**GCTL**, for short).

There is a technical challenge involved with such an extension:

- the concept of graded have to relapse both on states and paths;
- it is easy to have structures with an infinite number of paths satisfying a given property (e.g., Fq), so the concept of grade becomes useless.



$$w_0 \rightarrow w_1 \rightarrow w_2 \rightarrow w_0/w_2 \rightarrow w_1/w_0/w_2 \rightarrow \dots$$

We solve this problem using the concepts of **minimality** and **conservativeness**.

Syntax

Definition

GCTL* *state* (φ) and *path* (ψ) *formulas* are built inductively as follows:

- 1 $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid E \geq^g \psi \mid A <^g \psi,$
- 2 $\psi ::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid X\psi \mid \psi U \psi \mid \psi R \psi \mid \tilde{X}\psi \mid \psi \tilde{U} \psi \mid \psi \tilde{R} \psi.$

The simpler class of GCTL formulas is obtained by forcing each temporal operator, occurring in a formula, to be coupled with a path quantifier.

Since our semantics is defined on both finite and infinite paths, we have also to consider the weak temporal operators \tilde{X} , \tilde{U} , and \tilde{R} .

Informal semantics

Informally, the graded quantifiers $E^{\geq g}\psi$ and $A^{<g}\psi$ can be read as

- $E^{\geq g}\psi$: “there exist at least g paths that satisfy ψ ”;
- $A^{<g}\psi$: “all but less than g paths satisfy ψ ”.

However, the domain on which the quantifiers range is not the set of finite and infinite paths, but that of equivalence classes on paths w.r.t. an apposite equivalence relation.

An useful equivalence relation: **the prefix equivalence!**

Two paths are equivalent iff ...

- 1 their common prefix satisfy the formula (**minimality**);
- 2 no matter how the prefix is extended in the structure, the resulting path satisfies the formula (**conservativeness**).

Model properties

GCTL*, as CTL*,

- is invariant under **unwinding**;
- has the **tree** and **finite model property**.

However,

- it is not invariant under **bisimulation**;
- it is **more expressive** than CTL*.

All the above results hold for GCTL too.

Succinctness

Consider the property “in a tree, there are just g grandchildren of the root labeled with p , while all other nodes are not”.

It is possible to express such a property with the following GCTL formula of size logarithmic in g : $\varphi = (E^{=g}F p) \wedge (\neg p \wedge AX(\neg p \wedge AX(p \wedge AXAG \neg p)))$.

However, each $G\mu$ CALCULUS formulas equivalent to φ has to have size at least polynomial in the degree g .

Satisfiability

Question

Do the additional expressiveness and succinctness of GCTL imply an increasing of the computational-complexity cost of deciding its satisfiability problem?

Answer

No! It remains **EXPTIME-COMPLETE**.

Satisfiability via tree automata

General procedure

- 1 Given a specification φ , construct a tree automaton \mathcal{A}_φ recognizing all the tree models of φ itself.
- 2 Check for \mathcal{A}_φ emptiness.

Challenge here: The automaton \mathcal{A}_φ has to deal with the degree of φ .

Decidability of satisfiability

Key Idea: Every original tree model is encoded into a binary tree having an extra labeling used to manage the splitting of the degrees among the paths.

Thus, we reduce the satisfiability problem to the emptiness of a parity tree automaton polynomial in the size of the formula.
Hence, we obtain an **EXPTIME** satisfiability procedure for GCTL.

Part II

Logics for Strategies

From monolithic to multi-agent systems

Historical development:

- **Model checking**: analyzes systems monolithically (system components plus environment) [Clarke & Emerson, Queille & Sifakis, '81].
- **Module checking**: separates out the environment from the system components, but still views the system monolithically [Kupferman & Vardi, '96].
- **Alternating temporal reasoning**: multi-agent systems (components individually considered), playing strategically [Alur et al., '02].
- **Strategic logic reasoning**: two-player turn-based games verified by considering strategies as first order objects [Chatterjee et al., '07].

Strategic reasoning

Example

Reactive synthesis: Synthesize an interactive system that satisfies a given specification, independently of the possible sequences of inputs.

Example

Nash equilibrium: Verify that the players of a game have optimal strategies (each player is willing to follow its optimal strategy, if also the other do that).

Concurrent game model (I)

A *concurrent game structure* is a tuple $\mathcal{G} = \langle AP, Ag, Ac, St, \lambda, \tau, s_0 \rangle$ where:

- 1 AP : set of *atomic propositions*;
- 2 Ag : set of *agents*;
- 3 Ac : set of *actions*;
- 4 St : set of *states*;
- 5 $s_0 \in St$: *designated initial state*;
- 6 $\lambda : St \rightarrow 2^{AP}$: *labeling function*;
- 7 $\tau : St \times Ac^{Ag} \rightarrow St$: *transition function* mapping a state and a *decision* (i.e., a function from Ag to Ac) to a new state.

Concurrent game model (II)

St is not the global state space of the system, but the state space of the environment (the game) in which the agents operate.

Ac consists of local actions of all the agents.

Ac^{Ag} represents the set of choices of an action for each agent.

Alternating-Time Temporal Logics [Alur et al., '02]

$\langle\langle A \rangle\rangle\psi$: There is a strategy for the agents in $A \subseteq \text{Ag}$ enforcing the property ψ , independently of what the agents not in A can do.

Example

$\langle\langle \{\alpha, \beta\} \rangle\rangle G \neg \text{fail}$: “Agents α and β cooperate to ensure that a system (having possibly more than two processes (agents)) never enters a fail state.”

Despite its powerful expressiveness, ATL* suffers from the strong limitation that strategies are treated only implicitly.

The quantifier alternation is fixed to 1!

For instance, in the previous example, you cannot say that α has an uniform strategy w.r.t. the other agents, while β can choose its in dependence of their.

Strategy Logic [Chatterjee et al., '07]

Example

$\exists x_1, x_2. \forall y. \psi(x_1, y) \wedge \neg \psi(x_2, y)$: There are two strategies for the first player, x_1 and x_2 , such that x_1 enforces ψ while x_2 enforces $\neg \psi$, independently of the strategy y of the second player.

Reasoning About Strategies

Facts

- The strategy logic investigated by Chatterjee et al. is defined only for the weak framework of two-players and turn-based games.
- The model checking procedure is non-elementary, without matching lower-bound.
- The satisfiability problem is not studied at all.

Motivations

We need a logic in which we can talk about the **strategic behavior** of agents in **multi-player concurrent games**.

Such a logic can be used as a specification language for the verification and synthesis of **modular and interactive systems**.

Our proposal

We introduce a new **Strategy Logic** (SL), as a more general framework (both in its syntax and semantics), for explicit reasoning about strategies in *multi-player concurrent games*.

We investigate and solve both the problems of

- **model-checking** of a fragment (decidable, **2EXPTIME-COMPLETE**),
- **satisfiability** (undecidable, **Σ_1^1 -HARD**).

Syntax

Definition

SL *formulas* are built inductively in the following way, where $p \in AP$ is an atomic proposition, $x \in Var$ a variable, and $a \in Ag$ an agent.

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid \langle\langle x \rangle\rangle\varphi \mid [\![x]\!] \varphi \mid (a, x)\varphi.$$

SL syntactically extends LTL by means of *strategy quantifiers*, the existential $\langle\langle x \rangle\rangle$ and the universal $[\![x]\!]$, and *agent binding* (a, x) .

Informal semantics (I)

$\text{Trk}(\mathcal{G}) \subseteq \text{St}^+$ denotes the set of all *tracks*, i.e., finite paths, of the CGS \mathcal{G} .

A *strategy* for \mathcal{G} is a partial function $f : \text{Trk}(\mathcal{G}) \rightarrow \text{Ac}$ mapping each track in its domain to an action.

When all the agents have associated strategies, they individuate a unique path in the underlying CGS \mathcal{G} , called the *play* of the game.

Informal semantics (II)

Strategy quantification

- $\langle\langle x \rangle\rangle\varphi$: “there exists a strategy x for which φ is true”;
- $[[x]]\varphi$: “for all strategies x , it holds that φ is true”.

Agent binding

- $(a, x)\varphi$: “ φ holds, when the agent a uses the strategy x ”.

Ordering on strategies

We can force the existence of a **strict partial order without upper bound** over strategies of the agent α , by using the following formula:

$$\varphi^{ord} \triangleq \varphi^{unb} \wedge \varphi^{trn}:$$

$$1 \quad \varphi^{unb} \triangleq [[x_1]] \langle\langle x_2 \rangle\rangle x_1 < x_2;$$

$$2 \quad \varphi^{trn} \triangleq [[x_1]][[x_2]][[x_3]] (x_1 < x_2 \wedge x_2 < x_3) \rightarrow x_1 < x_3;$$

$$\text{where } x_1 < x_2 \triangleq \langle\langle y \rangle\rangle (\beta, y)((\alpha, x_1)(Xp) \wedge (\alpha, x_2)(X\neg p)).$$

Model checking

Question

Does the additional expressiveness of SL imply an increasing of the computational-complexity cost of deciding its model-checking problem?

Answer

No (for a fragment)! It remains **2EXPTIME-COMPLETE**.

Model checking via tree automata

General procedure

- 1 Given a model \mathcal{G} , construct a tree automaton $\mathcal{A}_{\mathcal{G}}$ recognizing the tree-unwinding of \mathcal{G} itself.
- 2 Given a specification φ , construct a tree automaton \mathcal{A}_{φ} recognizing all the tree models of φ itself.
- 3 Construct the product-automaton of $\mathcal{A}_{\mathcal{G}}$ and \mathcal{A}_{φ} and check for its emptiness.

Challenge here: The automaton \mathcal{A}_{φ} has to deal with the strategy quantifications of the formula φ .

Decidability of model checking

Key Idea (I): Every strategy quantification can be reduced to an action quantification, for each node of the tree-unwinding of the model.

Key Idea (II): Every action quantification can be handled locally on each node of the tree, by using the transition function of a tree automaton.

Thus, we reduce the model-checking problem to the emptiness of an exponential (in the size of the formula) parity tree automaton.
Hence, we obtain a **2EXPTIME** model-checking procedure for an SL fragment.

Satisfiability

Question

Does the additional expressiveness of SL imply an increasing of the computational-complexity cost of deciding its satisfiability problem?

Answer

Unfortunately, yes! It becomes **undecidable**.

Recurrent domino problem

Definition

An *recurrent domino system* is a tuple $\mathcal{D} = \langle \mathbf{D}, H, V, t^* \rangle$ consisting of a finite non-empty set \mathbf{D} of *domino types*, two *horizontal* and *vertical matching relations* $H, V \subseteq \mathbf{D} \times \mathbf{D}$, and a *distinguished* tile type $t^* \in \mathbf{D}$.

The recurrent domino problem asks for an *admissible tiling* of $\mathbb{N} \times \mathbb{N}$, i.e., a function $\partial : \mathbb{N} \times \mathbb{N} \rightarrow \mathbf{D}$ that labels the plane consistently with H and V in such a way that the tile type t^* is repeated infinitely often on the first row of the grid.

The solution of the problem is known to be *undecidable* [Harel, '84]. In particular, it is Σ_1^1 -*HARD*.

Undecidability of satisfiability

Key Idea (I): By using the ordering sentence, we force the existence of two infinite chains of strategies for two players α and β .

Key Idea (II): The two chains represent two perpendicular sides of the grid $\mathbb{N} \times \mathbb{N}$, whose point are in bijection with the pairs of strategies for α and β .

We use the previous ideas to construct a reduction of the recurrent domino problem to the satisfiability of a particular SL formula.

Hence, we obtain that the satisfiability problem is Σ_1^1 -HARD.

Thank you very much for your attention!