Logics in Computer Science

Fabio Mogavero

Università degli Studi di Napoli Federico II

13th Italian Conference on Theoretical Computer Science
Varese, Italy, September 19-21, 2012

Idea behind the thesis

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*.

Application

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

Areas of research

Topic areas

Our research fall in the two, so called, areas of:

- *logics for computations*;
- *logics for strategies*.

The thesis contains four different works, two for each area of research!

In this talk

Logics for computations

Graded Computation Tree Logic

Logics for strategies

Reasoning About Strategies

System verification, design, and synthesis

The framework

Let **S** represent a system and **P** a desired behavior.

There are three very important problems!

- Verification: Is **S** correct w.r.t. **P**?
- Design: Is **P** a correct specification?
- Synthesis: Build an **S** satisfying **P**.

Formal methods

Formalization

To answer to these questions, formal methods are used.

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

Decision procedures

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

Verification

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”.

Design and synthesis

Satisfiability: useful to verify...

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

Satisfiability witness: useful to synthesize the model of a correct system.

Part I

Logics for Computations

Kripke structures

KS

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

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

Classic temporal logics

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 path 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 of paths satisfying LTL-like properties.
  - Useful for software specification.

Linear-time Temporal Logic [Pnueli, ’79]

LTL syntax

$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\,\varphi \mid \varphi\, U\, \varphi \mid \varphi\, R\, \varphi.$

Informal semantics

- $X\,\varphi$: φ holds in the next state;
- $\varphi_1\, U\, \varphi_2$: φ_1 holds until φ_2 holds;
- $\varphi_1\, R\, \varphi_2$: φ_2 holds forever or until φ_1 holds;
- $F\,\varphi$: φ holds eventually;
- $G\,\varphi$: φ holds forever.

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

CTL^* syntax

1. $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi$,
2. $\psi ::= \text{LTL}(\varphi)$.

Example

- $\mathsf{EG}\,\mathsf{EF}\,\psi$: there is a computation from which, in each moment of its time, it starts another computation satisfying eventually ψ;
- $\mathsf{AFG}\,\psi$: all computations eventually reach a point in their time from which ψ holds forever.

Computational complexities

	Model Checking	Satisfiability
LTL	PSPACE-COMPLETE	PSPACE-COMPLETE
CTL	PTIME-COMPLETE	EXPTIME-COMPLETE
CTL*	PSPACE-COMPLETE	2EXPTIME-COMPLETE
μCALCULUS	UPTIME $\cap$ COUPTIME	EXPTIME-COMPLETE

Table: Computational complexity of model-checking and satisfiability problems.

Graded Computation Tree Logic

Facts

The ultimate temporal logic

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

Several extension of μCALCULUS have been considered.

One among all: GμCALCULUS, i.e., the μCALCULUS extended with graded modalities ("there are at least n successors such that...").

Expressiveness vs simplicity

μCALCULUS: 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?

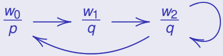
- XML query languages.
- Cyclomatic complexity.
- Redundancy in a system.
- Counting error counterexamples.

Our contribution

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

There is a technical challenge involved with such an extension:

- the counting concept have to relapse both on states and paths;
- it is easy to have structures with an infinite number of paths satisfying a given property, so the counting concept becomes easily useless.



$w_0 \longrightarrow w_1 \longrightarrow w_2 \longrightarrow w_0 / w_2 \longrightarrow w_1 / w_0 / w_2 \longrightarrow \cdots$

To solve the problem we use the concepts of minimality and conservativeness.

Syntax of GCTL*

GCTL* is a syntactic variant of CTL* in which classic path quantifiers are replaced by their *graded* version, the existential $\mathsf{E}^{\geq g}$ and the universal $\mathsf{A}^{<g}$.

Definition

GCTL* *state* (φ) and *path* (ψ) *formulas* are built inductively in the following way, where p is an atomic proposition:

1. $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{E}^{\geq g}\psi \mid \mathsf{A}^{<g}\psi$,
2. $\psi ::= \mathrm{LTL}(\varphi)$.

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

Fundamental notions

Domain of quantification

The domain on which quantifiers range is not simply the set of paths, but that of equivalence classes on paths w.r.t. an appropriate equivalence relation.

An useful equivalence relation

Two paths are prefix 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).

Semantics of GCTL*

Graded quantifications

- $E^{\geq g}\psi$: *"there exist at least g equivalence classes of paths satisfying ψ"*.
- $A^{<g}\psi$: *"all but less than g equivalence classes of paths satisfy ψ"*.

CTL* sublogic

- Boolean and temporal operators have classic semantics.

Model-theoretic properties

Positive model-theoretic properties

- Invariance under unwinding;
- Tree and finite model property.

Negative model-theoretic property

Non-invariance under bisimulation.

Expressiveness

Theorem

- GCTL* *is strictly more expressive than* CTL*.
- GCTL *is strictly more expressive than* CTL.

Succinctness

A simple property

In a tree, there are just g grandchildren of the root labeled with p, while all other nodes are not.

GCTL formalization

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

Observation

Each GμCALCULUS formulas equivalent to φ has to have size at least polynomial in g.

A natural question

A question

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

The answer

No! It remains EXPTIME-COMPLETE.

Satisfiability via tree automata

General procedure: $\exists \mathcal{T} \,.\, \mathcal{T} \models \varphi \Longleftrightarrow \mathrm{L}(\mathcal{A}_\varphi) \neq \emptyset$

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

Challenge: The automaton $\mathcal{A}_\varphi$ has to deal with the degree of φ.

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

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; two-player game between system and environment [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 all players of a game have optimal strategies (each player has a strategy such that it is rational for him to adhere to it assuming that all the other players also do so).

Concurrent game structures (I)

CGS

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

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

Concurrent game structures (II)

St is the arena of the game in which the agents operate.

Ac consists of local actions of all the agents.

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

Alternating-time Temporal Logic [Alur et al., '02]

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

Example

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

- Strategies are treated only implicitly.
- Quantifier alternation fixed to 1.

Strategy Logic [Chatterjee et al., '07]

Example

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

- Restricted framework of two-players turn-based games.
- Non-elementary model checking without matching lower-bound.
- Open satisfiability problem.

Reasoning About Strategies

Facts

Logics for strategies

In the literature there are no logics for the concurrent multi-player framework in which strategies are treated as explicit first-order objects.

Motivations

Extending the framework

We are looking for a logic in which we can talk about the strategic behavior of agents in generic multi-player concurrent games.

Application

It can be used as a specification language for the formal verification and synthesis of modular and interactive systems.

Our contribution

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 also study a chain of more tractable syntactic fragments which result to be strictly more expressive than ATL*.

Syntax of SL

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).

Definition

SL *formulas* are built inductively in the following way, where x is a variable and a an agent.

$$\varphi ::= \text{LTL}(\varphi) \mid \langle\langle x \rangle\rangle \varphi \mid [[x]] \varphi \mid (a, x) \varphi.$$

Fundamental notions

A *strategy* is a function mapping each *history* of the game to an *action*.

When all the agents have associated strategies, they identify a unique path named play of the game.

Semantics of SL

Strategy quantifications

- $\langle\langle x \rangle\rangle \varphi$: *"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".*

LTL sublogic

- Boolean and temporal operators have classic semantics.

Failure is not an option

Example (No failure property)

"In a system $\mathcal{S}$ built on three processes, α, β, and γ, the first two have to cooperate in order to ensure that $\mathcal{S}$ never enters a failure state".

Three different formalization in SL.

- $\langle\langle x\rangle\rangle\langle\langle y\rangle\rangle[[z]](\alpha, x)(\beta, y)(\gamma, z)(\mathsf{G}\,\neg \mathit{fail})$: α and β have two strategies, x and y, respectively, that, independently of what γ decides, ensure that a failure state is never reached.
- $\langle\langle x\rangle\rangle[[z]]\langle\langle y\rangle\rangle(\alpha, x)(\beta, y)(\gamma, z)(\mathsf{G}\,\neg \mathit{fail})$: β can chose his strategy y dependently of that one chosen by γ.
- $\langle\langle x\rangle\rangle[[z]](\alpha, x)(\beta, x)(\gamma, z)(\mathsf{G}\,\neg \mathit{fail})$: α and β have a common strategy x to ensure the required property.

Equilibria

Example (Nash equilibrium)

If each player has chosen a strategy and no player can benefit by changing his strategy while the other players keep theirs unchanged, then the current strategy profile constitute a Nash equilibrium [Nash, '50].

Example (k-resilient equilibrium)

Up to k players can deviate without improving their payoff [Aumann, '59].

Example (k-immune equilibrium)

The payoff of a non-deviating player does not degrade if up to k players deviate [Abraham et al., '06].

ATL* model-theoretic properties

Positive model-theoretic properties

- Invariance under bisimulation.
- Invariance under decision-unwinding.
- Bounded decision-tree model property.

SL model-theoretic properties

Negative model-theoretic properties

- Non-invariance under bisimulation.
- Non-invariance under decision-unwinding.
- Unbounded model property.

Positive model-theoretic properties

- Invariance under local isomorphism.
- Invariance under state-unwinding.
- State-tree model property.

Expressiveness

Theorem

SL *is strictly more expressive than* ATL*.

Explanation

- Unbounded quantifier alternation.
- More than one temporal goal at a time.
- Agents can be forced to share the same strategy.

Model checking (I)

Upper bound

- We reduce the solution of the model-checking problem to the checking of the non-emptiness of a suitable tree-automaton.
- The automaton construction is an evolution and merging of the translations from QPTL into nondeterministic Büchi automata [Sistla et al., '87] and from LTL into alternating Büchi automata [Vardi, '94].

Lower bound

- We reduce the satisfiability problem of QPTL to the model-checking problem of SL on a fixed one-agent game.
- The reduction is inspired by the hardness proof of ATL* with strategy contexts [Costa et al., '10].

Model checking (II)

Theorem

SL *model checking problem has a* NONELEMENTARYTIME-COMPLETE *formula complexity and* PTIME *data complexity.*

Satisfiability

Lower bound

- We reduce the recurrent domino problem to the satisfiability of SL.
- The reduction is strongly-based on the fact that SL does not have the bounded model property.

Theorem

SL *has a highly undecidable* (Σ_1^1-HARD) *satisfiability problem.*

An overview

	Model checking	Satisfiability
SL	NONELEMENTARY-COMPLETE	Σ_1^1-HARD
SL[NG] SL[BG] SL[CG] SL[1G]	NONELEMENTARY-COMPLETE ? 2EXPTIME-COMPLETE 2EXPTIME-COMPLETE	Σ_1^1-HARD Σ_1^1-HARD ? 2EXPTIME-COMPLETE
ATL*	2EXPTIME-COMPLETE	2EXPTIME-COMPLETE

Thank you very much for your kind attention!

I hope my talk was enough logic!