Decision Procedures for CTL*

Oliver Friedmann[1] Markus Latte[1]

[1] Dept. of Computer Science, Ludwig-Maximilians-University, Munich, Germany

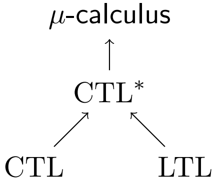
CLoDeM

Edinburgh, 15 July 2010

Introduction to CTL*

Origin: Emerson and Halpern '86

- supersedes the branching-time logic CTL and the linear-time logic LTL



- applied to specify and verify reactive and agent-based systems
- also applied to program synthesis
- however: decision procedures difficult to obtain
- worst case runtime: doubly exponential
  - lower bound: Vardi and Stockmeyer '85
  - upper bound: Emerson and Sistla '84; Emerson and Jutla '00

Table of contents

Syntax of CTL*

Negation normal form

$$\psi ::= q \mid \neg q \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathtt{X}\psi \mid \psi\mathtt{U}\psi \mid \psi\mathtt{R}\psi \mid \mathtt{E}\psi \mid \mathtt{A}\psi$$

where $q \in \mathcal{P}$ are propositional constants

Syntax of CTL*

Negation normal form

$$\psi ::= q \mid \neg q \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathtt{X}\psi \mid \xcancel{\psi\mathtt{U}\psi} \mid \xcancel{\psi\mathtt{R}\psi} \mid \mathtt{E}\psi \mid \mathtt{A}\psi$$

where $q \in \mathcal{P}$ are propositional constants

This talk: replace fixpoints $\psi\mathtt{U}\psi$, $\psi\mathtt{R}\psi$ by $\mathtt{F}\psi$, $\mathtt{G}\psi$.

Syntax of CTL*

Negation normal form

$$\psi ::= q \mid \neg q \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathtt{X}\psi \mid \mathtt{F}\psi \mid \mathtt{G}\psi \mid \mathtt{E}\psi \mid \mathtt{A}\psi$$

where $q \in \mathcal{P}$ are propositional constants

This talk: replace fixpoints $\psi\mathtt{U}\psi$, $\psi\mathtt{R}\psi$ by $\mathtt{F}\psi$, $\mathtt{G}\psi$.

Interpretation

Transition systems

TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ with

- $(\mathcal{S}, \rightarrow)$ directed, total graph
- $\lambda : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ labeling function

Interpretation

Transition systems

TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ with

- $(\mathcal{S}, \rightarrow)$ directed, total graph
- $\lambda : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ labeling function

Path π: sequence $(s_i)_{i \in \mathbb{N}} = s_0, s_1, \ldots$ of states respecting edges

Interpretation

Transition systems

TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ with

- $(\mathcal{S}, \rightarrow)$ directed, total graph
- $\lambda : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ labeling function

Path π: sequence $(s_i)_{i \in \mathbb{N}} = s_0, s_1, \ldots$ of states respecting edges

Notations: $\pi^i = s_i, s_{i+1}, \ldots$

Semantics

Semantics of Formulas

- $\mathcal{T}, \pi \models q$ iff $q \in \lambda(\pi(0))$
- $\mathcal{T}, \pi \models \neg q$ iff $q \notin \lambda(\pi(0))$
- $\mathcal{T}, \pi \models \psi_1 \wedge \psi_2$ iff $\mathcal{T}, \pi \models \psi_1$ and $\mathcal{T}, \pi \models \psi_2$
- $\mathcal{T}, \pi \models \psi_1 \vee \psi_2$ iff $\mathcal{T}, \pi \models \psi_1$ or $\mathcal{T}, \pi \models \psi_2$
- $\mathcal{T}, \pi \models \mathtt{X}\psi$ iff $\mathcal{T}, \pi^1 \models \psi$
- $\mathcal{T}, \pi \models \mathtt{F}\psi$ iff $\mathcal{T}, \pi^i \models \psi$ for some $i \in \mathbb{N}$
- $\mathcal{T}, \pi \models \mathtt{G}\psi$ iff $\mathcal{T}, \pi^i \models \psi$ for all $i \in \mathbb{N}$
- $\mathcal{T}, \pi \models \mathtt{E}\psi$ iff $\mathcal{T}, \widetilde{\pi} \models \psi$ for some $\widetilde{\pi}$ with $\pi(0) = \widetilde{\pi}(0)$
- $\mathcal{T}, \pi \models \mathtt{A}\psi$ iff $\mathcal{T}, \widetilde{\pi} \models \psi$ for all $\widetilde{\pi}$ with $\pi(0) = \widetilde{\pi}(0)$

State and Path Formulas

State and Path Formulas

A formula is a state formula iff X, F and G only occur under an E or an A. Otherwise the formula is a path formula.

Property

For any state formula φ, any paths π and π' in some TS $\mathcal{T}$ we have:

$$\mathcal{T}, \pi \models \varphi \text{ iff } \mathcal{T}, \pi' \models \varphi$$

provided that $\pi(0) = \pi'(0)$.

Notation:
$\mathcal{T}, s \models \varphi$ abbreviates $\mathcal{T}, \pi \models \varphi$ for a path π starting with s.

Satisfiability Problem

Satisfiability

Given a CTL^* state formula ϑ, decide whether there is a TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ and a state $s^* \in \mathcal{S}$ s.t.

$$\mathcal{T}, s^* \models \varphi$$

Satisfiability Problem

Satisfiability

Given a CTL^* state formula ϑ, decide whether there is a TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ and a state $s^* \in \mathcal{S}$ s.t.

$$\mathcal{T}, s^* \models \varphi$$

as opposed to the model checking problem

Model Checking

Given a CTL^* state formula φ *and* TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ *and* a state $s^* \in \mathcal{S}$, decide whether

$$\mathcal{T}, s^* \models \varphi$$

Satisfiability Problem

Satisfiability

Given a CTL^* state formula ϑ, decide whether there is a TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ and a state $s^* \in \mathcal{S}$ s.t.

$$\mathcal{T}, s^* \models \varphi$$

as opposed to the model checking problem

Model Checking

Given a CTL^* state formula φ *and* TS $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ *and* a state $s^* \in \mathcal{S}$, decide whether

$$\mathcal{T}, s^* \models \varphi$$

note: there is no strong relationship between satisfiability and model checking decision procedures (in general)!

Running Example

Consider the formula

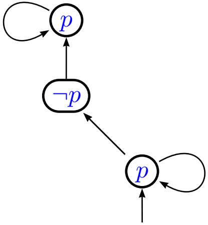
$$\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p$$

Running Example

Consider the formula

$$\texttt{AFG}p \wedge \texttt{EGEF}\neg p$$

The following TS is a model of it.



Overview

Emerson-Jutla Method ('84)

- emptiness test of a tree automaton accepting all models
- drawbacks: no implementation, unintuitive proof structure, constant branching degree

Reynolds' Tableaux ('09)

- exhaustive tableau-search restricted by small model property
- drawbacks: fairly slow in practice, no intrinsic detection of unfulfilled eventualities

Our System

- existence of infinite tableaux with global conditions
- drawbacks: requires automata determinisation for checking global conditions

Table of contents

Emerson et. al. – Overview

Given a CTL^*-formula ϑ,

- normalise ϑ to a normal form ψ,

$$\psi ::= \mathtt{E}\lambda \mid \mathtt{A}\lambda \mid \mathtt{AGE}\lambda \mid \psi \wedge \psi \mid \psi \vee \psi \mid p \mid \neg p$$

  where λ is a LTL-formula,

- construct a tree automaton which recognises tree-models of ψ, and
- test automaton for emptiness.

Emerson et. al. – Normalisation

Given a CTL^*-formula ϑ,

1. transform ϑ into negation form.
2. replace a subformula $\mathtt{Q}\lambda$, $\mathtt{Q} \in \{\mathtt{E}, \mathtt{A}\}$, by a fresh variable, say p.
3. attach $\wedge$"$\mathtt{AG}(p \leftrightarrow \mathtt{Q}\lambda)$" to ϑ.

$$\text{“}\mathtt{AG}(q \leftrightarrow \mathtt{E}\lambda)\text{”} \equiv \mathtt{AGE}(q \rightarrow \lambda) \wedge \mathtt{AG}(\neg q \rightarrow \neg \lambda)$$

$$\text{“}\mathtt{AG}(q \leftrightarrow \mathtt{A}\lambda)\text{”} \equiv \mathtt{AG}(q \rightarrow \lambda) \wedge \mathtt{AGE}(\neg q \rightarrow \neg \lambda)$$

4. iterate $2.$–$3.$ as long as possible.

Emerson et. al. – Tree Automaton

Let $\mathcal{B}_\lambda$ be a non-det. Büchi automaton for λ, (exp. size)
and $\mathcal{D}_\lambda$ be a det. parity or Rabin automaton for λ. (2-exp. size)

A tree automata for φ

$\mathtt{E}\lambda$: Simulate $\mathcal{B}_\lambda$ on a guessed path.

$\mathtt{A}\lambda$: Simulate $\mathcal{D}_\lambda$ on all paths.

Note: implicit quantifier in $\mathcal{B}_\lambda$ does not commute with the path quantifier.

$\mathtt{AGE}\lambda$: start a simulation of $\mathcal{B}_\lambda$ everywhere.

φ: follow the Boolean connectives.

Note: The connectives apply to the root only.

Emerson et. al. – Running Example

1. Normalize $\texttt{AFG}p \wedge \texttt{EGEF}\neg p$:
   $\texttt{AFG}p \wedge \texttt{EG}q \wedge \texttt{AGE}(q \implies \texttt{F}\neg p) \wedge \texttt{AG}(\texttt{F}\neg p \implies q)$
2. Build non-det. Büchi automata $B_{\texttt{G}q}$ and $B_{q \implies \texttt{F}\neg p}$
3. Build det. Rabin automata $D_{\texttt{FG}p}$ and $D_{\texttt{G}(\texttt{F}\neg p \implies q)}$
4. Turn all four automata into determinstic tree automata
5. Use a crossproduct construction to get a tree automaton for the initial formula
6. Apply an emptiness test

Emerson et. al. – Conclusion

Corollary

The decision procedure by Emerson, Sistla and Jutla is in 2EXPTIME.

However, Emerson noted that ...

> *"... [o]ne drawback to the use of automata is that, due to the delicate combinatorial constructions involved, there is usually no clear relationship between the structure of the automaton and the candidate formula."*

(E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 996–1072. Elsevier and MIT Press, New York, USA, 1990.)

Emerson et. al. – Conclusion

Corollary

The decision procedure by Emerson, Sistla and Jutla is in 2EXPTIME.

However, Emerson noted that . . .

> *". . . [o]ne drawback to the use of automata is that, due to the delicate combinatorial constructions involved, there is usually no clear relationship between the structure of the automaton and the candidate formula."*

(E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 996–1072. Elsevier and MIT Press, New York, USA, 1990.)

another drawback: fixed branching degree of final tree automaton

Table of contents

Reynolds' Tableaux

Structure

- finite tableaux with back-loops
- nodes labelled with colours: a set of hues
- hues – Hintikka-style sets – correspond to fullpaths in the intended model
- edges in a tableau correspond to proceeding in time by one step
- successors of a node depend on the contained intended fullpaths

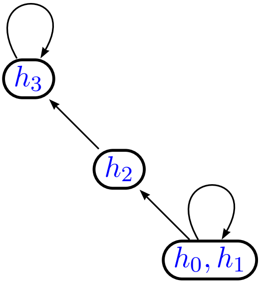
Reynolds' Tableaux (cont.)

Correctness

- local conditions: node correctness and successor correctness
- global conditions: eventualities in hue threads have to be fulfilled

Theorem: Reynolds' tableau system is sound and complete.

Reynolds' Tableau for $\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p$



Relevant Hues

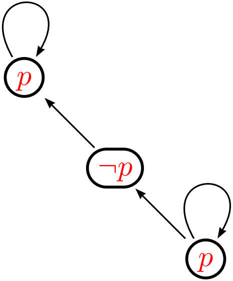
h_0 : $\{\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{EF}\neg p, \mathtt{G}p, p, \mathtt{EGEF}\neg p, \mathtt{GEF}\neg p\}$

h_1 : $\{\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{EF}\neg p, \mathtt{F}\neg p, p, \mathtt{EGEF}\neg p, \mathtt{FAG}p\}$

h_2 : $\{\mathtt{EGF}\neg p \vee \mathtt{AFAG}p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{EF}\neg p, \mathtt{F}\neg p, \neg p, \mathtt{AFAG}p, \mathtt{FAG}p\}$

h_3 : $\{\mathtt{EGF}\neg p \vee \mathtt{AFAG}p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{G}p, p, \mathtt{AFAG}p, \mathtt{FAG}p, \mathtt{AG}p\}$

Reynolds' Tableau for $\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p$



Relevant Hues

h_0 : $\{\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{EF}\neg p, \mathtt{G}p, p, \mathtt{EGEF}\neg p, \mathtt{GEF}\neg p\}$

h_1 : $\{\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{EF}\neg p, \mathtt{F}\neg p, p, \mathtt{EGEF}\neg p, \mathtt{FAG}p\}$

h_2 : $\{\mathtt{EGF}\neg p \vee \mathtt{AFAG}p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{EF}\neg p, \mathtt{F}\neg p, \neg p, \mathtt{AFAG}p, \mathtt{FAG}p\}$

h_3 : $\{\mathtt{EGF}\neg p \vee \mathtt{AFAG}p, \mathtt{AFG}p, \mathtt{FG}p, \mathtt{G}p, p, \mathtt{AFAG}p, \mathtt{FAG}p, \mathtt{AG}p\}$

Tableau Search

Algorithmic Method

- tableau-building
- loop checking
- backtracking

Tableau Search

Algorithmic Method

- tableau-building
- loop checking
- backtracking

Good Loops

- witness the fact that every eventually in the hue thread is satisfied after a finite number of steps
- checked by a model-checking style algorithm

Tableau Search (cont.)

Bad Loops

- occurring repetition but looping back results in unfulfilled eventualities
- solution: extend the branch instead of looping back
- problem: when do we stop to extend unfulfilled branches?

Tableau Search (cont.)

Bad Loops

- occurring repetition but looping back results in unfulfilled eventualities
- solution: extend the branch instead of looping back
- problem: when do we stop to extend unfulfilled branches?

When to stop?

- currently: use small model property to restrict the length of the branches
- but: small model property yields doubly exponential bound

Performance in practice

based on Reynolds' prototype implementation

- comparably slow as unprofitable branches are solely detected by hitting the length restriction
- running example: longer than one day; our system requires less than a second

Table of contents

A Tableau for CTL*

A tableau for ϑ is a tree which imitates a potential model of ϑ.

A Tableau for CTL*

A tableau for ϑ is a tree which imitates a potential model of ϑ.

A pre-tableau for a formula ϑ is an infinite tree s.th.

- it is finitely branching,
- each node is labelled with a goal (as a set),

$$\mathtt{A}\Sigma_1, \ldots, \mathtt{A}\Sigma_n, \mathtt{E}\Pi_1, \ldots, \mathtt{E}\Pi_m, \Lambda$$

set of literals in ϑ (Λ)

blocks ($\mathtt{A}\Sigma_1, \ldots, \mathtt{A}\Sigma_n, \mathtt{E}\Pi_1, \ldots, \mathtt{E}\Pi_m$)

Example: $\{\mathtt{A}\{\neg p \vee q\},\ \mathtt{E}\{\mathtt{X}p, \mathtt{F}q\},\ \neg p,\ \neg q\}$.

Sloppy writing: $\mathtt{A}(\neg p \vee q)$ or $\mathtt{E}(\mathtt{X}p, \Pi)$, e.g.

A Tableau for CTL*

A tableau for ϑ is a tree which imitates a potential model of ϑ.

A pre-tableau for a formula ϑ is an infinite tree s.th.

- it is finitely branching,
- each node is labelled with a goal (as a set),

$$\mathtt{A}\,\Sigma_1,\ \ldots,\ \mathtt{A}\,\Sigma_n,\ \mathtt{E}\,\Pi_1,\ \ldots,\ \mathtt{E}\,\Pi_m,\ \Lambda$$

$\Pi_1, \ldots, \Pi_m$: set of subformulas of ϑ (as a conjunction)

$\Sigma_1, \ldots, \Sigma_n$: set of subformulas of ϑ (as a disjunction)

Example: $\{\mathtt{A}\{\neg p \lor q\},\ \mathtt{E}\{\mathtt{X}p, \mathtt{F}q\},\ \neg p,\ \neg q\}$.
Sloppy writing: $\mathtt{A}(\neg p \lor q)$ or $\mathtt{E}(\mathtt{X}p, \Pi)$, e.g.

A Tableau for CTL*

A tableau for ϑ is a tree which imitates a potential model of ϑ.

A pre-tableau for a formula ϑ is an infinite tree s.th.

- it is finitely branching,
- each node is labelled with a goal (as a set),

$$\underbrace{\mathtt{A}\Sigma_1,\ \dots,\ \mathtt{A}\Sigma_n,\ \mathtt{E}\Pi_1,\ \dots,\ \mathtt{E}\Pi_m,\ \Lambda}$$

$$\bigwedge_{i=1}^{n} \mathtt{A}(\bigvee \Sigma_i) \ \wedge\ \bigwedge_{i=1}^{m} \mathtt{E}(\bigwedge \Pi_i) \ \wedge\ \bigwedge \Lambda$$

Example: $\{\mathtt{A}\{\neg p \vee q\},\ \mathtt{E}\{\mathtt{X}p, \mathtt{F}q\},\ \neg p,\ \neg q\}$.
Sloppy writing: $\mathtt{A}(\neg p \vee q)$ or $\mathtt{E}(\mathtt{X}p, \Pi)$, e.g.

A Tableau for CTL*

A tableau for ϑ is a tree which imitates a potential model of ϑ.

A pre-tableau for a formula ϑ is an infinite tree s.th.

- it is finitely branching,
- each node is labelled with a goal (as a set),

  $\mathtt{A}\Sigma_1, \ldots, \mathtt{A}\Sigma_n, \mathtt{E}\Pi_1, \ldots, \mathtt{E}\Pi_m, \Lambda$
- nodes are locally consistent, i.e.
  - does not contain a literal together with its negation, and
  - does not contain $\mathtt{A}\emptyset$.
- root is labelled with $\mathtt{E}\{\vartheta\}$,
- nodes follow the following rules . . .

Exemplary Rules

$$(\mathtt{E}\vee)\ \frac{\mathtt{E}(\varphi,\Pi),\Phi \quad | \quad \mathtt{E}(\psi,\Pi),\Phi}{\mathtt{E}(\varphi\vee\psi,\Pi),\Phi} \qquad (\mathtt{E}\wedge)\ \frac{\mathtt{E}(\varphi,\psi,\Pi),\Phi}{\mathtt{E}(\varphi\wedge\psi,\Pi),\Phi}$$

$$(\mathtt{EF})\ \frac{\mathtt{E}(\psi,\Pi),\Phi \quad | \quad \mathtt{E}(\mathtt{X}(\mathtt{F}\psi),\Pi),\Phi}{\mathtt{E}(\mathtt{F}\psi,\Pi),\Phi} \qquad (\mathtt{AF})\ \frac{\mathtt{A}(\psi,\mathtt{X}(\mathtt{F}\psi),\Sigma),\Phi}{\mathtt{A}(\mathtt{F}\psi,\Sigma),\Phi}$$

$$(\mathtt{X}_1)\ \frac{\mathtt{E}\Pi_1,\mathtt{A}\Sigma_1,\ldots,\mathtt{A}\Sigma_m,\Phi \quad \ldots \quad \mathtt{E}\Pi_n,\mathtt{A}\Sigma_1,\ldots,\mathtt{A}\Sigma_m,\Phi}{\mathtt{EX}\Pi_1,\ldots,\mathtt{EX}\Pi_n,\mathtt{AX}\Sigma_1,\ldots,\mathtt{AX}\Sigma_m,\Lambda,\Phi}$$

Traces and Threads

Traces

- A trace is an infinite sequence of connected blocks.
- A trace is an A- resp. E- trace iff the block quantifier eventually remains A resp. E.

Thread

- A thread is an infinite sequence of connected formulas.
- A thread is an F- resp. G-thread iff there is some ψ s.t. the thread finally alternates between $\mathtt{F}\psi$ or $\mathtt{XF}\psi$ (resp. $\mathtt{G}\ldots$).

Tableau

Pre-tableaux are insufficient – an *informal* dicussion

- In the intended model
  - every formula on a F-thread is false, and
  - every formula on a G-thread is true.
- Blocks in an E-trace is understood as a conjunction.
  - Avoid F-threads.
- Blocks in an A-trace is understood as a disjunction.
  - Assure a G-thread.

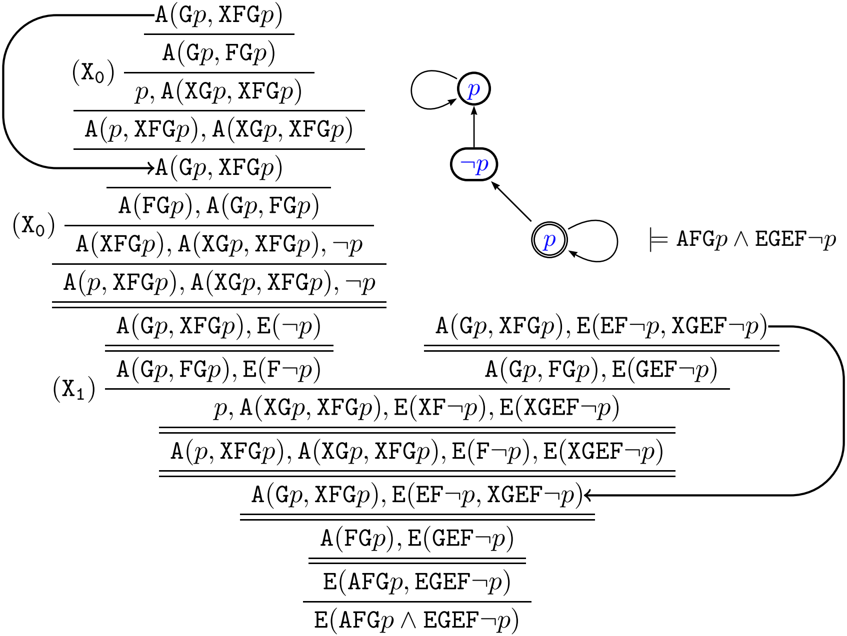
Definiton

A tableau for ϑ is a pre-tableau for ϑ iff on every branch we have

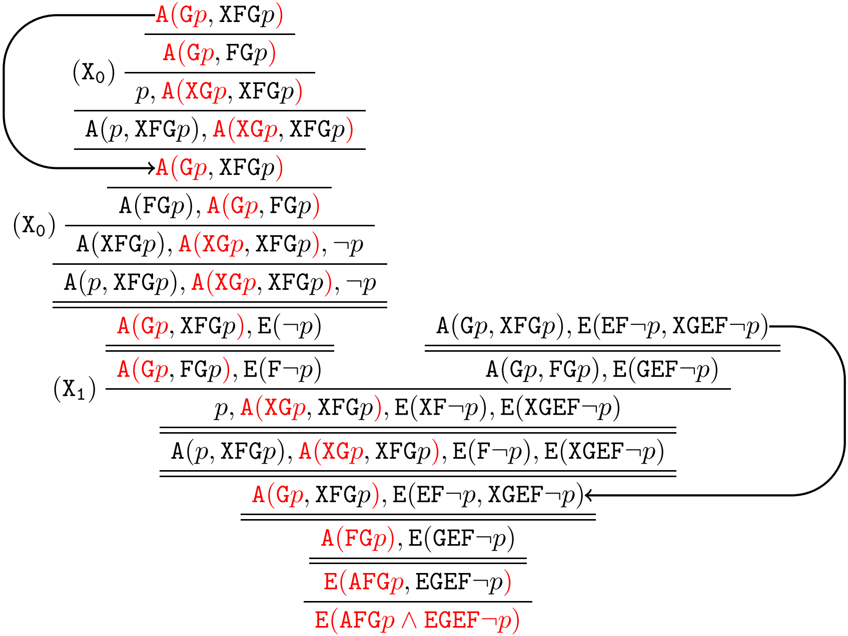
- every E-trace does not contain an F-thread, and
- every A-trace contains a G-thread.

Such traces and branches are called good.

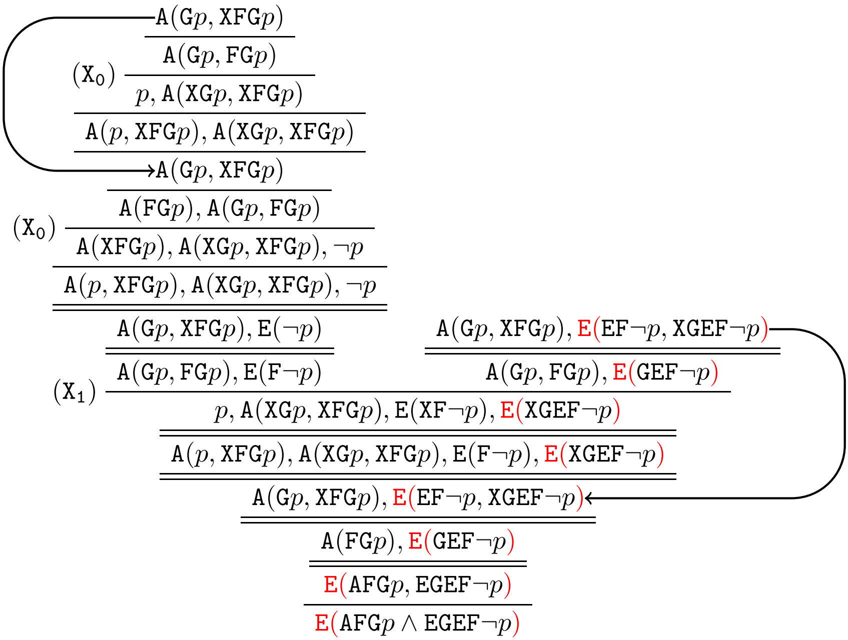
Successful Tableau for $\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p$



Successful Tableau for $\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p$



Successful Tableau for $\mathtt{AFG}p \wedge \mathtt{EGEF}\neg p$



Decision Procedure

Given a CTL^*-formula ϑ, decide whether ϑ is satisfiable.

Decision Procedure

there is a tableau for ϑ

Given a CTL^*-formula ϑ, decide whether ~~ϑ is satisfiable~~.

Decision Procedure

there is a tableau for ϑ

Given a CTL^*-formula ϑ, decide whether ~~ϑ is satisfiable~~.

Idea: treat a tableau as a parity game.

Reduction to Parity Games

The tableaux as a game

- Nodes are the goals for ϑ.
- Proponent (player 0) chooses a rule application if neither $(\mathtt{X}_0)$ nor $(\mathtt{X}_1)$ is applicable.
- Opponent (player 1) chooses a rule application and a premise if $(\mathtt{X}_0)$ or $(\mathtt{X}_1)$ is applicable.

Reduction to Parity Games

The tableaux as a game

- Nodes are the goals for ϑ.
- Proponent (player 0) chooses a rule application if neither $(\mathtt{X}_0)$ nor $(\mathtt{X}_1)$ is applicable.
- Opponent (player 1) chooses a rule application and a premise if $(\mathtt{X}_0)$ or $(\mathtt{X}_1)$ is applicable.

Problem

This game defines a pre-tableaux but not a tableaux.

Observation

The property separating pre-tableau and tableaux is ω-regular.

Table of contents

Implementation – Our vs. Reynold

Note: Reynold's implementation is a proof-of-concept in `Java` but compiled with `gcj`.

- Formula $(\mathtt{AG}(p \rightarrow \mathtt{EX}r) \wedge \mathtt{AG}(r \rightarrow \mathtt{EX}p)) \rightarrow (p \rightarrow \mathtt{EG}(\mathtt{F}p \wedge \mathtt{F}r))$

	formula	negated formula
Reynold	> 10h	> 10h
Ours	0s	15s

- Formula $\mathtt{AG}\big((p \wedge \mathtt{X}\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \mathtt{X}p \wedge q \wedge \neg r) \vee (\neg p \wedge \mathtt{X}p \wedge \neg q \wedge r)\big) \wedge \mathtt{E}(\mathtt{F}q \wedge \mathtt{F}r)$

	formula	negated formula
Reynold	17s	> 10h
Ours	0s	0s

Concluding Comparison

Aspect / Method	Emerson et. al.	Reynolds	ours
Concept	tree-automata	tableau	tableau
Worst-case complexity	2EXPTIME	2EXPTIME	2EXPTIME
Implementation available	no	not public	yes
Model construction	yes	yes	yes
Finite representation by	rabin	small. mod. p.	parity
Out-degree	fix., lin. bounded	var., lin. bounded	var., lin. bounded
Req. small model property	no	yes	no
Derives small model prop.	yes	no	yes
Needs Büchi determ.	yes	no	yes