Model Checking and Games

Part III - Temporal Logic

Rüdiger Ehlers, Clausthal University of Technology

September 2019

Temporal Logic

Introduction

$$\phi \models \psi$$

Introduction

$$\phi \models \psi$$

Kripke
Structures

Introduction

$$\phi \models \psi$$

Kripke Structures

Assert statements
Next: Temporal logic

Basic idea

Temporal logic

With temporal logic, we can express properties of the *behavior* of a reactive system rather than its *state*

Basic idea

Temporal logic

With temporal logic, we can express properties of the *behavior* of a reactive system rather than its *state*

Examples

- Every *green light* of a traffic light should eventually be followed by a *yellow light*.
- For every direction d, the traffic light implementation should permit a future execution such that there is a green light for direction d.

Agenda

Temporal logic

- Temporal logics provide a way to *formalize* such requirements
- The formalization allows us to use the requirements as an input to a *model checking* process.

Logics considered in the following

- Linear temporal logic (LTL)
- Computation tree logic (CTL)

Linear Temporal Logic (LTL)

Linear temporal logic

Basic properties

- Originally introduced by Pnueli (1977)
- It is a logic on *infinite words* over the alphabet 2^{AP} for some set of atomic propositions AP $\rightarrow$ can be used to reason about traces of a system

Use in verification

We can use LTL to express properties that we want to hold along *all* traces of a system. Examples:

- Every *green light* of a traffic light should eventually be followed by a *yellow light*.
- ~~For every direction *d*, the traffic light implementation should permit a future execution such that there is a green light for direction *d*.~~

Ultimately periodic words

Idea

- For finite prefixes of words, we cannot always say for sure if an LTL formula is satisfied or not.
- Infinite words cannot be stored in memory.
- Are there some *finitary* representations of *some* infinite words?

Ultimately periodic words

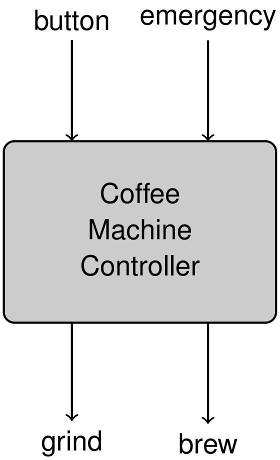
Idea

- For finite prefixes of words, we cannot always say for sure if an LTL formula is satisfied or not.
- Infinite words cannot be stored in memory.
- Are there some *finitary* representations of *some* infinite words?

Ultimately periodic words

Ultimately periodic words over an alphabet Σ are of the form uv^{ω} for some finite words $u \in \Sigma^*$ and $v \in \Sigma^*$.

Some LTL properties – examples



Atomic propositions

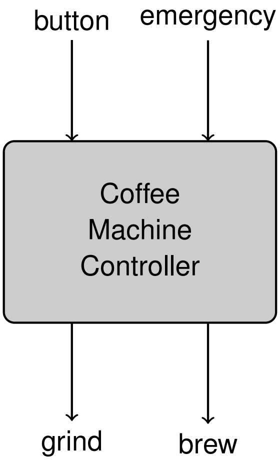
- $\mathrm{AP}_I = \{bu, e\}$
- $\mathrm{AP}_O = \{g, br\}$

Example property (1)

When the machine starts, it does not grind.

$$\neg g$$

Some LTL properties – examples



Atomic propositions

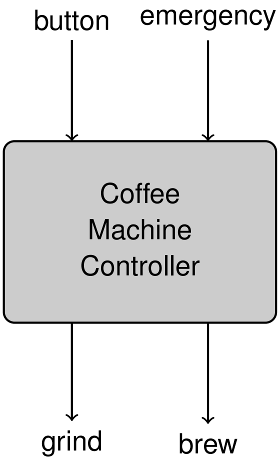
- $\mathrm{AP}_I = \{bu, e\}$
- $\mathrm{AP}_O = \{g, br\}$

Example property (2)

When the machine starts, it does not grind for the first three time steps.

$$\neg g \wedge \mathbf{X}\neg g \wedge \mathbf{XX}\neg g$$

Some LTL properties – examples



Atomic propositions

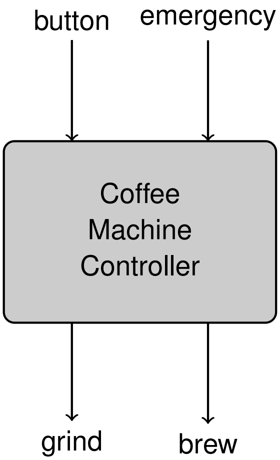
- $\mathrm{AP}_I = \{bu, e\}$
- $\mathrm{AP}_O = \{g, br\}$

Example property (3)

Whenever the button is pressed, grinding happens in the next step:

$$\mathbf{G}(bu \rightarrow \mathbf{X}g)$$

Some LTL properties – examples



Atomic propositions

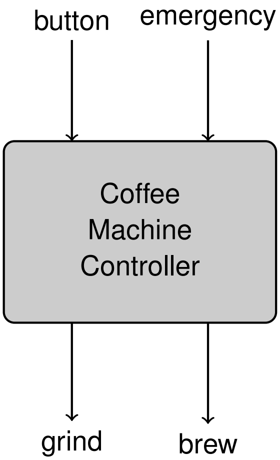
- $\mathsf{AP}_I = \{bu, e\}$
- $\mathsf{AP}_O = \{g, br\}$

Example property (4)

Whenever the machine grinds, it does so until it brews

$$\mathbf{G}(g \rightarrow (g\,\mathcal{U}\,b))$$

Some LTL properties – examples



Atomic propositions

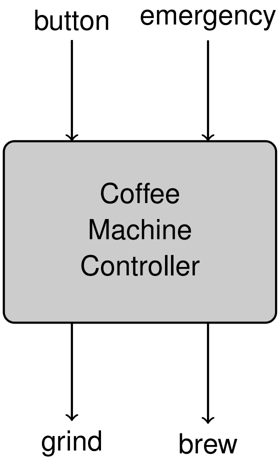
- $\mathrm{AP}_I = \{bu, e\}$
- $\mathrm{AP}_O = \{g, br\}$

Example property (5)

After the emergency button is pressed, the machine does not grind nor brew any more:

$$\mathbf{G}(e \rightarrow \mathbf{G}\neg g) \land \mathbf{G}(e \rightarrow \mathbf{G}\neg b)$$

Some LTL properties – examples



Atomic propositions

- $\mathrm{AP}_I = \{bu, e\}$
- $\mathrm{AP}_O = \{g, br\}$

Example property (6)

The grinding unit does not start running forever:

$$\mathbf{GF}\neg g$$

Encoding infinitely often

Question

Why does $\mathbf{GF}\psi$ encode that ψ should hold *infinitely often* along a trace?

Answer - An inductive argument

Encoding infinitely often

Question

Why does $\mathbf{GF}\psi$ encode that ψ should hold *infinitely often* along a trace?

Answer - An inductive argument

Whenever there is a position at which ψ holds, there is also a *later* one on which ψ also holds, as $\mathbf{F}\psi$ holds globally.

Encoding infinitely often

Question

Why does $\mathbf{GF}\psi$ encode that ψ should hold *infinitely often* along a trace?

Answer - An inductive argument

Whenever there is a position at which ψ holds, there is also a *later* one on which ψ also holds, as $\mathbf{F}\psi$ holds globally. Hence, every finite sequences of indices $i_1, \ldots, i_n$ such that $w^{i_j} \models \psi$ for all $1 \leq j \leq n$ can be extended while maintaining this property.

Encoding infinitely often

Question

Why does $\mathbf{GF}\psi$ encode that ψ should hold *infinitely often* along a trace?

Answer - An inductive argument

Whenever there is a position at which ψ holds, there is also a *later* one on which ψ also holds, as $\mathbf{F}\psi$ holds globally. Hence, every finite sequences of indices $i_1, \ldots, i_n$ such that $w^{i_j} \models \psi$ for all $1 \leq j \leq n$ can be extended while maintaining this property. Thus, all such sequences must be infinite!

Encoding infinitely often

Question

Why does $\mathbf{GF}\psi$ encode that ψ should hold *infinitely often* along a trace?

Answer - An inductive argument

Whenever there is a position at which ψ holds, there is also a *later* one on which ψ also holds, as $\mathbf{F}\psi$ holds globally. Hence, every finite sequences of indices $i_1, \ldots, i_n$ such that $w^{i_j} \models \psi$ for all $1 \leq j \leq n$ can be extended while maintaining this property. Thus, all such sequences must be infinite!

The fact that there exists such a sequence is trivial (ϵ).

Complementing “infinitely often”

Used lemma

$$\mathbf{G}\psi \quad \equiv \quad \neg\mathbf{F}\neg\psi$$

Idea

We want to encode that some subformula ψ holds only *finitely often*. So we complement $\mathbf{GF}\psi$:

$$\begin{aligned} & \neg\mathbf{GF}\psi \\ \equiv\ & \mathbf{F}\neg\mathbf{F}\psi \\ \equiv\ & \mathbf{FG}\neg\psi \end{aligned}$$

This encodes that at some point in the future, ψ never holds (again).

Did we define all important components?

$$\phi \models \psi$$

Did we define all important components?

$$\phi \models \psi$$

Kripke Structures

Did we define all important components?

$$\phi \models \psi$$

Kripke Structures

Linear temporal logic

Did we define all important components?

$$\phi \models \psi$$

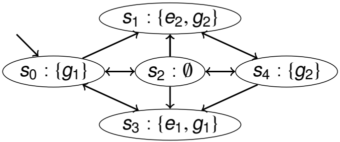
Kripke Structures

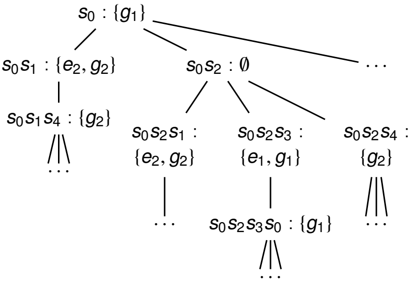
Linear temporal logic

Observation

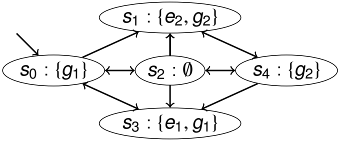
That does not fit! LTL is defined over words, not transition systems!

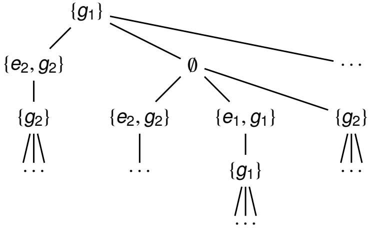
Unrolling a transition system to a tree (assuming a single initial state)



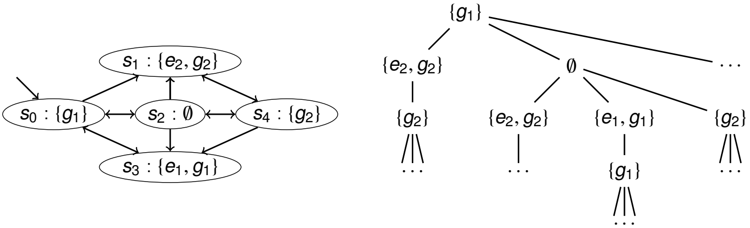


Unrolling a transition system to a tree (assuming a single initial state)





Unrolling a transition system to a tree (assuming a single initial state)



LTL interpretation

An LTL specification is checked along every *branch* in the computation tree!

LTL model checking with `spin`

Using `spin`, we can check if every trace of a model satisfies a specification.

Downside of the current definitions

Observation

The current definition for the satisfaction of a temporal logic formula by a Kripke structure treats all traces individually. This disallows expressing system properties such as:

> *Along every trace, at every point in the trace there is some future evolution of the system on which the coffee machine controller never grinds again.*

Downside of the current definitions

Observation

The current definition for the satisfaction of a temporal logic formula by a Kripke structure treats all traces individually. This disallows expressing system properties such as:

> *Along every trace, at every point in the trace there is some future evolution of the system on which the coffee machine controller never grinds again.*

How to fix this?

We can use a logic that reasons about the whole computation tree rather than only over its traces.

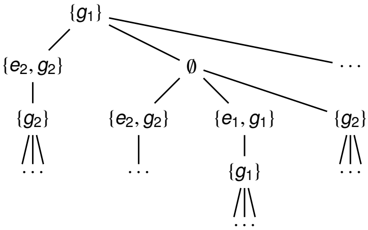
Computation Tree Logic (CTL)

Computation tree logic

Basic properties

- Originally introduced by Clarke and Emerson (1981)
- Is a logic over *trees with infinite branches*
- Extends propositional logic

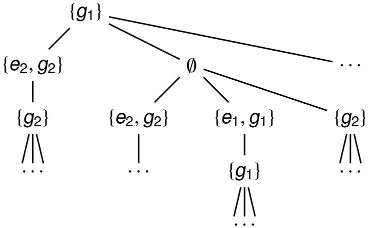
Examples for CTL



Property

From every node in every every branch, there is the possibility for the system to remain in states labeled by $\{e_1, g_1\}$ forever.

Examples for CTL



Property

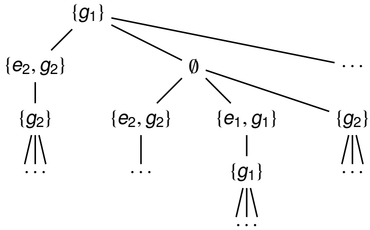
From every node in every every branch, there is the possibility for the system to remain in states labeled by $\{e_1, g_1\}$ forever.

In CTL

$$\mathbf{AG}(\mathbf{EG}(e_1 \wedge g_1 \wedge \neg e_2 \wedge \neg g_2))$$

(Note that the property makes little sense.)

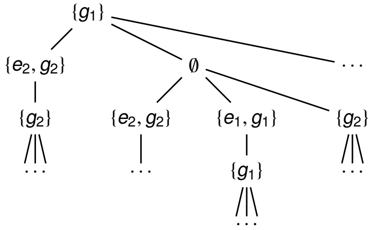
Examples for CTL



Property

There exists a sequence of states such that eventually, the set of states labeled with emergency(1) cannot be left again.

Examples for CTL



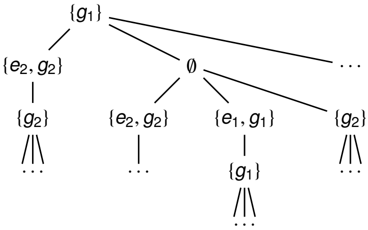
Property

There exists a sequence of states such that eventually, the set of states labeled with emergency(1) cannot be left again.

In CTL

$$\mathbf{EF}(\mathbf{AG}e_1)$$

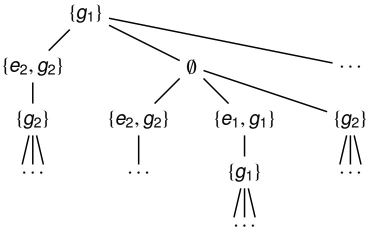
Examples for CTL



Property

There exists a trace on which g_1 is never given, but from every state on the trace, it is always given at most two steps after emergency override e_1 happens.

Examples for CTL



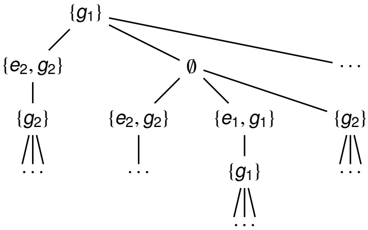
Property

There exists a trace on which g_1 is never given, but from every state on the trace, it is always given at most two steps after emergency override e_1 happens.

In CTL

$$\mathbf{EG}(\neg g_1 \wedge \mathbf{AX}(e_1 \rightarrow (g_1 \vee \mathbf{AX}g_1)))$$

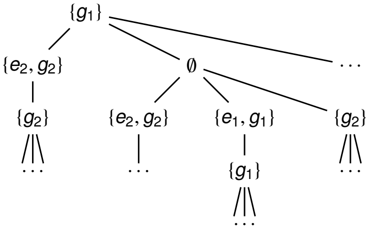
Examples for CTL



Property

There exists a trace on which when g_1 is true for the first time, g_1 can stay true together with e_2 immediately afterwards.

Examples for CTL



Property

There exists a trace on which when g_1 is true for the first time, g_1 can stay true together with e_2 immediately afterwards.

In CTL

$$\mathbf{E}(\neg g_1 \,\mathcal{U}\, (g_1 \wedge \mathbf{EX}(g_1 \wedge e_2)))$$

Computation tree logic

Preview

We postpone the in-depth discussion of CTL to a later part of the course.

Summary / List of Concepts

- Basics of temporal logic
- Linear Temporal Logic (Syntax and Semantics)
- Computation Tree logic (Syntax and Semantics)
- Basic LTL model checking with `spin`

References I

Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. doi: 10.1007/BFb0025774. URL `https://doi.org/10.1007/BFb0025774`.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57, 1977. doi: 10.1109/SFCS.1977.32. URL `https://doi.org/10.1109/SFCS.1977.32`.