Specification models and their analysis

Kai Lampka

November 18, 2009

Institut für Technische Informatik und Kommunikationsnetze

Agenda

1. Graph Theory: Some Definitions
2. Introduction to Petri Nets
3. Introduction to Computation Tree Logic and related model checking techniques ◁
4. Introduction to Binary Decision Diagrams

Part I

Introduction to Computation Tree Logic

- In (formal) logic one studies how to combine propositional formulae consisting of atomic propositions, manipulate the formulae, and ultimately draw correct conclusions, i. e., decide if a (complex) formula (= combination of statements) is correct or not.
- This requires a decidable theory and a set of "mechanical" methods for showing that a complex formula is true or not.

Question: What does this mean in the context of systems engineering?

⟶ Example 1.1: Introduction to propositional logic

We extend the notion of Labelled Transition Systems as follows:

Definition 1.1: Kripke structure

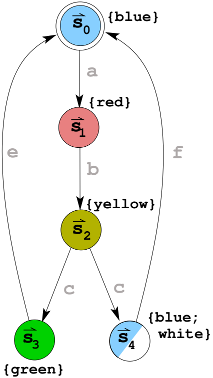
A Kripke structure $\mathcal{K}$ is a six-tuple $\mathcal{K} := (\mathbb{S}, \mathbb{S}_0, \mathcal{A}ct, \mathbb{E}, \mathcal{AP}, \mathcal{L})$, where

1. $\mathbb{S} := \{\vec{s}_1, \ldots, \vec{s}_n\}$ is an ordered (indexed) set of states with
2. $\mathbb{S}_0$ is the set of initial states.
3. $\mathcal{A}ct$ is the discrete set of transition labels,
4. $\mathbb{E} \subseteq \mathbb{S} \times \mathcal{A}ct \times \mathbb{S}$ is an ordered (indexed) set of labelled state-to-state transitions.
5. $\mathcal{AP}$ is a set of atomic propositions, e. g. $\{green, blue, yellow, black\}$ and
6. $\mathcal{L} : \mathbb{S} \mapsto 2^{\mathcal{AP}}$ as state labelling function.

⟶ Example 1.2: "Weather" Kripke structure

Introduction to CTL Model checking

- Analogously to propositional logic one wants to reveal if a formal statement about a system's behavior is correct or not.
- Whereas in propositional logics this is easy, –one simply needs to evaluate a formulae w. r. t. an assignment μ–, the reasoning about Kripke structures is much more demanding.
- However, at first we need to clarify how a Kripke structure defines a system behavior.



- System is given as Kripke structure, hence future behavior is defined by sequences states.
- for any pair of states within such a sequence, denoted as path, the resp. states within the Kripke structure are connected by an edge:

$$\pi_{\vec{s}_0} := \vec{s}_0, \vec{s}_1, \vec{s}_2, \vec{s}_4, \vec{s}_0, \vec{s}_1 \text{ (finite path fragment)}$$

- in fact we are intrested in the sequence of atomic propositions attached to each state ($\mathcal{L}(\vec{s}_i)$), but for simplicity we stick to the state identifiers $\vec{s}_i$

As we see from this:

- temporal logics which are the logics on transition systems are *time abstract*, i. e., they allow to reason about *ordering* of states. They *do not* allow to reason about state residence times!
- The modelling and reasoning about real-time systems is denoted timed verification.
- Hence one reasons over system behaviors which are defined by paths in the Kripke structure.
- This allows one to make statements over a single path (= linear time view), or over sets of paths (= branching time view).
- CTL follows the branching time view, hence it allows to make statements about set of paths, like $\exists$ a path s.t. , $\forall$ paths it holds: ...

Introduction to CTL Model Checking: Branching time view

- To reason about the properties of a system (model) in a branching time view one must expand all possible behaviors, starting from some dedicated state.
- For simplicity we are considering in the following only Kripke structures
  - with a single initial state ($\vec{s}_0$), s.t. we only need to worry about paths starting in state $\vec{s}_0$.
  - which are non-terminal (= non-deadlocks),

$\longrightarrow$ Question 1.1: What do we get if we unroll all paths of a Kripke structure, transition by transition starting at the initial state

The computation tree (CT) of a Kripke structure $\mathcal{K} := (\mathbb{S}, \mathbb{S}_0, \mathcal{A}ct, \mathbb{E}, \mathcal{AP}, \mathcal{L})$ can be constructed as follows:

- each node of the CT carries a state label contained in $\mathbb{S}$;
- the root of the CT is labelled with the state label $\vec{s}_0$;
- each child of a CT-node c is labelled with a state-label $\vec{s}$ and it is a successor of $\vec{s}$ resp. state in $\mathcal{K}$.
- The set of children nodes of a CT-node c can than be defined as follows:

$$\texttt{child}(c) := \bigcup_{\forall l \in \mathcal{A}ct : (\vec{s}, l, \vec{t}) \in \mathbb{E}} \vec{t}$$

- Since each node of the CT carries a state label $\vec{s}$, it can be annotated with the set of atomic propositions which are actually fulfilled by the resp. state $\vec{s}$, i. e., with $\mathcal{L}(\vec{s})$.

CTL Model Checking: Defining CTL

CTL has the following ingredients:

1. atomic propositions, where a state $\vec{s}$ satisfies a atomic proposition $a \in \mathcal{AP}$ if it carries the respective label ($\mathcal{L}(\vec{s}) = a$)
2. standard logic operators $\wedge, \neg$ and their derivatives, e. g. $\rightarrow$, which allow to construct more complex state formulae;

   $\longrightarrow$ Example 1.3: $a \rightarrow \neg(c \vee b)$
3. quantifiers $\exists$ and $\forall$ applied to path formulae, i. e., sequences of state properties to be fulfilled w. r. t. some starting state $\vec{s}_0$.

   $\longrightarrow$ Example 1.4: $\exists\Psi, \forall\Psi$
4. temporal operators $\bigcirc$ (= next) and U (= until) which we apply to state formulae and which gives us path formulae;

   $\longrightarrow$ Example 1.5: $\Psi := \bigcirc\, b$ $\Psi' := a\, \mathrm{U}\, b$

Definition 1.2: Computation Tree Logic

- CTL formula consists of sub-formulae which are either **path** formulae (Ψ) or **state** formulae (ϕ). With $a \in \mathcal{AP}$ as set of atomic propositions we give the following definitions:
  - A CTL **state formula** ϕ is defined as

    $$\phi := \mathit{true} \,|\, a \in \mathcal{AP} \,|\, \phi' \wedge \phi'' \,|\, \neg\phi' \,|\, \exists\Psi \,|\, \forall\Psi$$

    with ϕ, ϕ', ϕ'' as CTL state formulae and Ψ as CTL path formula.
  - A CTL **path formula** Ψ is defined as

    $$\Psi := \bigcirc\,\phi \,|\, \phi\,\mathsf{U}\,\phi'$$

    where the ϕ's are CTL **state formulae**.

Consider the following CTL formulae with $\{coin, wash\} =: \mathcal{AP}$

- $\exists \bigcirc coin$
- $\forall(true \, \mathsf{U} \, wash)$
- $\exists(coin \wedge \forall \bigcirc wash)$
- $\exists \bigcirc (coin \wedge \forall \bigcirc wash)$

1. Which of the above formulae are syntactically correct?
2. How does a non-trivial fulfilling CT look like?

As for propositional logics we define a **satisfaction relation** $\models$ for CTL-formulae:

Definition 1.3: Semantics of CTL

1. For a Kripke structure $\mathcal{K}$ and a state $\vec{s}$ we define the following:
   - $\vec{s} \models a \Leftrightarrow a \in \mathcal{L}(\vec{s})$
   - $\vec{s} \models \neg\phi \Leftrightarrow \vec{s} \not\models \phi$
   - $\vec{s} \models \phi \wedge \phi' \Leftrightarrow \vec{s} \models \phi \wedge \vec{s} \models \phi'$
   - $\vec{s} \models \exists\Psi \Leftrightarrow \pi_{\vec{s}} \models \Psi$ for **some** path $\pi_{\vec{s}}$ in $\mathcal{K}$
   - $\vec{s} \models \forall\Psi \Leftrightarrow \pi_{\vec{s}} \models \Psi$ for **all** paths $\pi_{\vec{s}}$ in $\mathcal{K}$
2. For a path $\pi_{\vec{s}}$ in $\mathcal{K}$ we define:
   - $\pi_{\vec{s}} \models \bigcirc\phi \Leftrightarrow \pi_{\vec{s}}[1] \models \phi$
   - $\pi_{\vec{s}} \models \phi \,\mathsf{U}\, \phi'$

     $\Leftrightarrow \exists j \geq 0 : \pi_{\vec{s}}[j] \models \phi' \wedge \forall(k : 0 \leq k < j) : \pi_{\vec{s}}[k] \models \phi,$

     where $\pi_{\vec{s}}[x]$ refers to the x'th state of path $\pi_{\vec{s}}$.

- However complex CTL-formulae might also contain non-standard operators, e. g. $a \rightarrow \neg(c \vee b)$.
- For reducing the number of cases to be covered ($true, a \in \mathcal{AP}, \wedge, \neg, \forall\bigcirc, \exists\bigcirc, \forall\,\mathsf{U}, \exists\,\mathsf{U}$), as well as for simplifying their treatment each CTL-formula is converted into a **normal form**
- In the following we will make use of the so called existential normal form (ENF) which solely employs the operators $\neg, \wedge, \exists\bigcirc, \exists\,\mathsf{U}$ and $\exists\Box$ where $\Box$ is the always operator.

Definition 1.4: The always operator ($\Box$)

- potentially always: $\exists\Box\phi := \neg\forall(true\,\mathsf{U}\,\neg\phi)$

  there is (at least one) path π s.t. ϕ holds in each state of π.
- invariantly: $\forall\Box\phi := \neg\exists(true\,\mathsf{U}\,\neg\phi)$

  for all paths Π and hence all states ϕ holds

⟶ Example 1.6: Example for the $\Box$-operator

Definition 1.5: Existential normal form

A CTL-formula is in existential normal form (ENF) if it is of the following type:

$$\phi := \mathit{true} \mid a \in \mathcal{AP} \mid \phi \wedge \phi \mid \neg\phi \mid \exists \bigcirc \phi \mid \exists(\phi \,\mathsf{U}\, \phi) \mid \exists\Box\phi$$

For converting a CTL formula in ENF one needs to replace the universal by the existential quantifier. This is possible by exploiting the following dualities:

- $\forall \bigcirc \phi = \neg\exists \bigcirc \neg\phi$
- $\forall(\phi' \,\mathsf{U}\, \phi'') = \neg\exists\left[\neg\phi'' \,\mathsf{U}\, (\neg\phi' \wedge \neg\phi'')\right] \wedge \neg\exists\Box\neg\phi''$

Thus for deciding if a system $\mathcal{L}$ complies with a property a resp. model checking algorithm must only cover the above 7 ENF-base cases.

- For actually model checking a LTS $\mathcal{L}$ we need to extend the above defined satisfaction relation to transition systems (we also do not want to expand the CT explicitly).
- Let Ω be a CTL-formula and let $\mathcal{L}$ be a finite non-terminal LTS

$$\mathcal{L} \models \Omega \Leftrightarrow \vec{s}_0 \models \Omega$$

- This gives the outline of the CTL model checking procedure:
  1. Construct *Satisfy*(Ω) which is the set of states for which a given CTL-formula Ω holds and which we therefore define as follows:

$$Satisfy(\Omega) := \{\vec{s} \in \mathbb{S} \mid \vec{s} \models \Omega\}$$

  2. Check if the initial state of $\mathcal{L}$ is contained in this set, since

$$\mathcal{L} \models \Omega \Leftrightarrow \vec{s}_0 \in Satisfy(\Omega)$$

- How to compute the set *Satisfy* is of major concern now.

Preliminary: take CTL-formula and convert it into ENF and provide state labellings for LTS w. r. t. the atomic propositions of the CTL formula.

1. generate a parse tree for the CTL formula s.t. the leaves of the parse tree carry atomic propositions or the constant *true*
2. construct *Satisfy*(Ω) by processing the parse tree bottom-up, i. e., one computes the satisfaction sets of the leave nodes then for their parent nodes and so on and on ...
3. check if the initial state is contained in the satisfaction set *Satisfy*(Ω)

Definition 1.6: Parse Tree

Given a CTL-formula Ω we construct a parse tree s.t.

- a leaf of the parse tree carries an atomic proposition or the constant *true* as occurring in a sub-formulae of the CTL-formula to be parsed
- the inner nodes carry combined operators as employed for connecting different state formulae, i. e., $op \in \{\neg, \wedge, \vee, \forall\bigcirc, \exists\bigcirc, \forall\,\mathsf{U}, \exists\,\mathsf{U}\}$.

$\longrightarrow$ Example 1.7: Parse tree for $\exists\bigcirc a \wedge \exists(b\,\mathsf{U}\,[\neg\forall(\mathit{true}\,\mathsf{U}\,\neg c)]$

(I) What do we need to do for the **leaves** of the parse tree,
i. e., . how do we compute *Satisfy*(ϕ) for $\phi := true \,|\, a \in \mathcal{AP}$?

1. $\phi = true$
   this set contains all states, since all states are satisfying the constant *true* formula, i. e., we have

$$Satisfy(\phi) := Satisfy(true) := \mathbb{S}$$

2. $\phi \in \mathcal{AP}$
   we collect all states labelled with ϕ, i. e.,

$$Satisfy(\phi) := \{\vec{s} \in \mathbb{S} \mid \mathcal{L}(\vec{s}) = \phi\}$$

(II) What do we need to do for the **inner** nodes of the parse tree?

1. Simple case covering the computation of *Satisfy*(ϕ) for

$$\phi := \neg\varphi \,|\, \varphi' \wedge \varphi'' \,|\, \exists \bigcirc \varphi$$

- $\underline{\phi = \neg\varphi}$: *Satisfy*$(\phi)$ is the complement of *Satisfy*(φ) w. r. t. $\mathbb{S}$

$$\mathit{Satisfy}(\phi) := \mathbb{S} \setminus \mathit{Satisfy}(\varphi)$$

- $\underline{\phi = \varphi' \wedge \varphi''}$: *Satisfy*$(\phi)$ is the intersection of the satisfaction sets of φ' and φ'':

$$\mathit{Satisfy}(\phi) := \mathit{Satisfy}(\varphi') \cap \mathit{Satisfy}(\varphi'')$$

- $\underline{\phi = \exists \bigcirc \varphi}$: *Satisfy*$(\phi)$ are all those states which predecessors satisfy φ, i. e.,

$$\mathit{Satisfy}(\phi) := \{\vec{s} \in \mathbb{S} \mid \mathcal{P}ost(\vec{s}) \cap \mathit{Satisfy}(\varphi) \neq \emptyset\}$$

$\longrightarrow$ Example 1.8: *Satisfy*$(\exists \bigcirc \varphi)$

(II) Handling of **inner** nodes of the parse tree (continued).

2. Complex case requires fixed point computation for obtaining *Satisfy*(ϕ) in case

$$\phi := \varphi' \,\mathsf{U}\, \varphi'' \,|\exists\Box\varphi$$

- $\phi = \exists(\varphi' \,\mathsf{U}\, \varphi'')$:

$$\begin{aligned} Satisfy_0(\phi) &:= Satisfy(\varphi'') \\ Satisfy_{i+1}(\phi) &:= Satisfy_i(\phi) \cup \\ &\quad \{\vec{s} \in Satisfy(\varphi') \mid \mathcal{P}ost(\vec{s}) \cap Satisfy_i(\phi) \neq \emptyset\} \end{aligned}$$

- $\phi = \exists\Box\varphi$:

$$\begin{aligned} Satisfy_0(\phi) &:= Satisfy(\varphi) \\ Satisfy_{i+1}(\phi) &:= \{\vec{s} \in Satisfy(\varphi) \mid \mathcal{P}ost(\vec{s}) \cap Satisfy_i(\phi) \neq \emptyset\} \end{aligned}$$

⟶ Example 1.9: Model Checking of "weather" LTS

1. Witnesses and counter examples:
   - path demonstrating $\mathcal{L} \models \phi$ is denoted **witnesses**
   - path demonstrating $\mathcal{L} \not\models \phi$ is denoted **counter example**.
2. A last operator (eventually):

Definition 1.7: The eventually operator ($\Diamond$)

- potentially: $\exists\Diamond\phi := \exists(\mathit{true}\,\mathsf{U}\,\phi)$

  at least one path π goes at least through one state where ϕ holds.
- inevitable: $\forall\Diamond\phi := \forall(\mathit{true}\,\mathsf{U}\,\phi)$

  all paths go at least through one state there ϕ holds.
