Computational Tree Logic and Model Checking
A simple introduction

F. Raimondi
I am Franco Raimondi, f.raimondi@cs.ucl.ac.uk

Slides, coursework, coursework solutions can be found online: http://www.cs.ucl.ac.uk/staff/f.raimondi/ (choose “Teaching”).


Structure of the lectures:

- Lectures on CTL and NuSMV (2 or 3 hours, 1.20)
- Lab class on NuSMV (1 or 2 hours, 4.06)
- Coursework (will be given next Friday, deadline Friday 2 Feb, electronic submission by email)

Thanks to Dr. Alessio Lomuscio (now at Imperial College) for various material in these slides.
Reference material for CTL


- pp. 221 – 230 (Section 3.6.1): Model checking algorithms.

Reference material for NuSMV

- Lecture notes based on NuSMV tutorial available online.
Introduction

“There is a great advantage in being able to verify the correctness of computer systems, whether they are hardware, software, or a combination. This is most obvious in the case of safety-critical systems, but also applies to those that are commercially critical (such as mass-produced chips), mission critical (think of NASA rovers), etc.”

In this lecture: verification using model checking.

Model checking is an automatic, model-based, property verification approach.
Model checking: definition

Given a model $M$ and a formula $\varphi$, model checking is the problem of verifying whether or not $\varphi$ is true in $M$ (written $M \models \varphi$: don’t worry, it will be much clearer by the end of these slides).
Model checking: definition

Given a model $M$ and a formula $\varphi$, model checking is the problem of verifying whether or not $\varphi$ is true in $M$ (written $M \models \varphi$: don’t worry, it will be much clearer by the end of these slides).
Model checking: definition

Given a model $M$ and a formula $\varphi$, model checking is the problem of verifying whether or not $\varphi$ is true in $M$ (written $M \models \varphi$: don’t worry, it will be much clearer by the end of these slides).

- **Distributed system** abstracts
- **Logical model $M$** represents
- **Required property**
- **Logical formula $\varphi$**
Model checking: definition

Given a model $M$ and a formula $\varphi$, model checking is the problem of verifying whether or not $\varphi$ is true in $M$ (written $M \models \varphi$: don’t worry, it will be much clearer by the end of these slides).
Model checking and temporal logic

Model checking is based on mainly temporal logic. There are various kinds of temporal logic: Linear Temporal Logic (LTL), Computational Tree Logic (CTL), CTL*, µ-calculus, etc.

In this lecture we will cover CTL, a logic to reason about sequence of events. For instance, we will write formally statements such as:

*There exists an execution of the system such that, if the proposition $p$ is true, then in the next computation step $q$ is true*

(at the end of this lecture: try to encode this in CTL).
Basic concepts: syntax and semantics

- **Syntax** tells how to write formulas: syntax gives the rules to write *correct* (i.e., *well-formed*) formulas (wff).

- Semantics gives a *meaning* to well-formed formulas. Semantics is used to decide whether or not a given wff is true or false.
CTL Syntax

We start from a set of atomic propositions $AP = \{p, q, \ldots\}$. Atomic propositions stand for atomic facts which may hold in a system, e.g. “Printer ps706 is busy”, “Process 1486 is idle”, “The value of $x$ is 5”, etc.

The Backus-Naur form form CTL formulae is the following:

$$\phi ::= \top | \bot | p | \neg \phi | \phi \land \phi | \phi \lor \phi | \phi \rightarrow \phi | AX \phi | EX \phi |
\quad AF \phi | EF \phi | AG \phi | EG \phi | A[\phi U \phi] | E[\phi U \phi]$$

IF YOU DON’T KNOW THE MEANING OF THIS EXPRESSION PLEASE RAISE YOUR HAND NOW!!!
CTL Syntax

Each CTL operator is a pair of symbols. The first one is either A ("for All paths"), or E ("there Exists a path"). The second one is one of X ("neXt state"), F ("in a Future state"), G ("Globally in the future") or U ("Until").

NOTICE: U is a binary operator, it could be written $EU(\varphi, \psi)$ or $AU(\varphi, \psi)$. Notice that the quantifier is graphically separated (e.g., $E[pUq]$), but it is in fact a single operator $EU$, which could be written $EU(p, q)$.

Example: $AG(p \rightarrow (EFq))$ is read as “It is Globally the case that, if $p$ is true, then there Exists a path such that at some point in the Future $q$ is true”.
CTL Syntax: parse trees

Parse trees are very useful to understand CTL formulas. For instance:

$$(EF(EGp)) \rightarrow E[pUq]$$
CTL Syntax: parse trees

Parse trees are very useful to understand CTL formulas. For instance:

\[(EF(EGp)) \rightarrow E[pUq]\]
CTL Syntax: EXERCISE

Build the parse tree of the following formula:

$$A[pU\neg q] \land (AF(\neg EGr))$$
CTL Syntax: EXERCISE

Build the parse tree of the following formula:

\[ A[pU\neg q] \land (AF(\neg EG r)) \]
CTL Syntax: EXERCISE

Is it a wff? Why?

1. $EFGr$
2. $A\neg G \neg p$
3. $A[pU(EFr)]$
4. $F[rUq]$
5. $EF(rUq)$
6. $AEFr$
7. $A[rUA[pUq]]$
8. $A[(rUq) \land (pUr)]$
CTL Syntax: EXERCISE

Answers

1. $EFGr$ NO
2. $A\neg G\neg p$ NO
3. $A[pU(EFr)]$ YES
4. $F[rUq]$ NO
5. $EF(rUq)$ NO
6. $AEFr$ NO
7. $A[rUA[pUq]]$ YES
8. $A[(rUq) \land (pUr)]$ NO
CTL Semantics

You should be able to identify well-formed CTL formulae. Now: how to evaluate formulae, i.e., how to decide whether or not a formula is true.
CTL Semantics

You should be able to identify well-formed CTL formulae. Now: how to evaluate formulae, i.e., how to decide whether or not a formula is true.

You should know the meaning of tautology and unsatisfiable formulae:

- $AG(p \lor \neg p)$: tautology
- $AG(p \land \neg p)$: unsatisfiable
CTL Semantics

You should be able to identify well-formed CTL formulae. Now: how to evaluate formulae, i.e., how to decide whether or not a formula is true.

You should know the meaning of tautology and unsatisfiable formulae:

- $AG(p \lor \neg p)$: tautology
- $AG(p \land \neg p)$: unsatisfiable

But what about $EFp$? it may be true or not, depending on how we evaluate formulae.
CTL Semantics: transition systems

We evaluate formulae in transition systems. A transition system model a system by means of states and transitions between states. Formally:

A transition system $M = (S, R_t, L)$ is a set of states $S$ with a binary relation $R_t \subseteq S \times S$ and a labelling function $L : S \to 2^{AP}$ ($AP$ is a set of atomic propositions, see above). The relation $R_t$ is serial, i.e., for every state $s \in S$, there exists a state $s'$ s.t. $s R_t s'$. 
CTL Semantics: transition systems

An example $M = (S, R_t, L)$

Here $S = \{s_0, s_1, s_2\}$, $R_t = \{(s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_2), (s_2, s_2)\}$, and $L(s_0) = \{p, q\}$, $L(s_1) = \{q, r\}$, $L(s_2) = \{r\}$. 
CTL semantics: from transition systems to computation paths

It is useful to visualise all possible computation paths by *unwinding* the transition system:
CTL semantics: computation paths EXERCISE

Unwind the following transition systems
CTL semantics: computation paths EXERCISE SOLUTION

A simple introduction to CTL and model checking
Short summary

- You should be able to recognise well-formed CTL formulas.
- You know what a transition system is \((M = (S, R_t, L))\).
- You know how to unwind a transition system and obtain computation paths.

Next: Given a CTL formula \(\varphi\) and a transition system \(M\), establish whether or not \(\varphi\) is true at a given state \(s\) in \(M\), written as:

\[ M, s \models \varphi \]
CTL semantics (finally!)

Let $M = (S, R_t, L)$ be a transition system (also called a *model* for CTL). Let $\varphi$ be a CTL formula and $s \in S$. $M, s \models \varphi$ is defined inductively on the structure of $\varphi$, as follows (I’m using the first transition system of today as an example on the board):

\[
\begin{align*}
M, s &\models \top \\
M, s &\not\models \bot \\
M, s &\models p \quad \text{iff} \quad p \in L(s) \\
M, s &\models \neg \varphi \quad \text{iff} \quad M, s \not\models \varphi \\
M, s &\models \varphi \land \psi \quad \text{iff} \quad M, s \models \varphi \quad \text{and} \quad M, s \models \varphi \\
M, s &\models \varphi \lor \psi \quad \text{iff} \quad M, s \models \varphi \quad \text{or} \quad M, s \models \varphi
\end{align*}
\]
CTL Semantics (temporal operators)

\[ M, s \models AX \varphi \quad \text{iff} \quad \forall s' \text{ s.t. } sR_t s', \; M, s' \models \varphi \]

\[ M, s \models EX \varphi \quad \text{iff} \quad \exists s' \text{ s.t. } sR_t s' \text{ and } M, s' \models \varphi \]

\[ M, s \models AG \varphi \quad \text{iff} \quad \text{for all paths } (s, s_2, s_3, s_4, \ldots) \text{ s.t. } s_i R_t s_{i+1} \text{ and for all } i, \]

\[ \text{it is the case that } M, s_i \models \varphi \]

\[ M, s \models EG \varphi \quad \text{iff} \quad \text{there is a path } (s, s_2, s_3, s_4, \ldots) \text{ s.t. } s_i R_t s_{i+1} \text{ and for all } i \]

\[ \text{it is the case that } M, s_i \models \varphi \]

\[ M, s \models AF \varphi \quad \text{iff} \quad \text{for all paths } (s, s_2, s_3, s_4, \ldots) \text{ s.t. } s_i R_t s_{i+1}, \text{ there is} \]

\[ \text{a state } s_i \text{ s.t. } M, s_i \models \varphi \]

\[ M, s \models EF \varphi \quad \text{iff} \quad \text{there is a path } (s, s_2, s_3, s_4, \ldots) \text{ s.t. } s_i R_t s_{i+1}, \text{ and there is} \]

\[ \text{a state } s_i \text{ s.t. } M, s_i \models \varphi \]
CTL Semantics (temporal operators)

\[ M, s \models A[\varphi U \psi] \iff \text{for all paths } (s, s_2, s_3, s_4, \ldots) \text{ s.t. } s_i R t s_{i+1} \text{ there is a state } s_j \text{ s.t. } M, s_j \models \psi \text{ and } M, s_i \models \psi \text{ for all } i < j. \]

\[ M, s \models E[\varphi U \psi] \iff \text{there exists a path } (s, s_2, s_3, s_4, \ldots) \text{ s.t. } s_i R t s_{i+1} \text{ and there is a state } s_j \text{ s.t. } M, s_j \models \psi \text{ and } M, s_i \models \psi \text{ for all } i < j. \]
CTL semantics: EXERCISE

Consider the following transition system:

Verify whether or not: (1) $M, s_0 \models EX(\neg p)$; (2) $M, s_0 \models EXEG(r)$; (3) $M, s_1 \models AG(q \lor r)$; (4) $M, s_2 \models A[rUq]$; (5) $M, s_1A[qUAG(r)]$; (6) $M, s_1E[qUEG(r)]$; (7) $M, s_0 \models \neg EG(q)$; (8) $M, s_1 \models EFAG(q)$. 
CTL semantics: EXERCISE SOLUTIONS

(1) YES; (2) YES; (3) YES ; (4) YES; (5) NO (because $AG(r)$ is never true if you keep looping between $s_0$ and $s_1$); (6) YES; (7) YES; (8) YES.
Equivalences between CTL formulas

In the syntax of CTL we introduced all the operators AX, EX, AF, EF, AG, EG, AU, and EU. However, some formulas are equivalent:

\[
AX \varphi \equiv \neg EX \neg \varphi \\
AG \varphi \equiv \neg EF \neg \varphi \\
AF \varphi \equiv \neg EG \neg \varphi
\]

Moreover, \( EF \varphi \equiv E[\top U \varphi] \). Therefore, only three operators are required to express all the remaining: \( EX, EG, EU \) (this is called an adequate set of operators. This is useful when developing algorithms for model checking.
Specification patterns

Temporal logics are useful to express requirements of systems. Typically, requirements have *common and recurring patterns*. For instance, two example of patterns:

- **Liveness**: “Something good will eventually happen”. For instance: “Whenever any process requests to enter its critical section, it will eventually be permitted to do so”. In CTL:

  \[ AG(request \rightarrow AF(critical)) \]

- **Safety**: “Nothing bad will happen”. For instance, “Only one process is in its critical section at any time”. In CTL (with 2 processes only):

  \[ AG(\neg (critical_1 \land critical_2)) \]
Specification patterns: EXERCISE

Write in CTL the following requirements:

1. “From any state it is possible to get a reset state”

2. “Event $p$ precedes $s$ and $t$ on all computation paths” (try to encode the negation of this).

3. “On all computation paths, after $p$, $q$ is never true”.

Specification patterns: EXERCISE

Write in CTL the following requirements:

1. “From any state it is possible to get a reset state”

2. “Event $p$ precedes $s$ and $t$ on all computation paths” (try to encode the negation of this).

3. “On all computation paths, after $p$, $q$ is never true”.

1. $\text{AGEF}(\text{reset})$

2. The negation: there exists in the future a state in which $p$ follows $s \land t$: $\text{EF}((s \land t) \rightarrow \text{EF}(p))$. Its negation:
   $$\neg\text{EF}((s \land t) \rightarrow \text{EF}(p)) \equiv \text{AG}(\neg((s \land t) \rightarrow \text{EF}(p)))$$

3. $\text{AG}(p \rightarrow (\neg\text{EF}(q)))$
Summary

- You are able to recognise and write well-formed CTL formulas.
- You are able to unwind a transition system in computation paths.
- You are able to evaluate whether or not a given CTL formula is true at a given state of a transition system $M$.
- You are able to recognise (simple) equivalent CTL formulas.
- You are able to translate simple requirements from plain English into CTL syntax.
Model checking algorithms
Model checking CTL: introduction

We have seen very simple example in these slides. However, real systems may be composed of hundred of thousand states. Efficient algorithms are needed to verify $M, s \models \varphi$.

We will see example of systems using NuSMV (a model checker) later in the course.

How do you verify a formula in a model? What we did: unwind the transition system $M$. However, a computer cannot check infinite data structures: we need to check finite data structure.

Next: an algorithm to compute the set of states of a model $M$ in which $\varphi$ holds, the labelling algorithm.
The labelling algorithm

- **INPUT**: a CTL model $M = (S, R_t, L)$ and a CTL formula $\varphi$.
- **OUTPUT**: the set of states of $M$ which satisfy $\varphi$.

Sketch: (1) express $\varphi$ using the adequate set of operators $EX$, $EG$, $EU$; (2) operate recursively on the structure of $\varphi$, starting from sub-formulas.
The labelling algorithm, informally

Suppose all the subformulas of $\varphi$ have already been labelled. If $\varphi$ is:

- $p$: label $s$ with $p$ if $p \in L(s)$.
- $\varphi_1 \land \varphi_2$: label $s$ with $\varphi_1 \land \varphi_2$ if $s$ is already labelled both with $\varphi_1$ and $\varphi_2$.
- $\neg \varphi_1$: label $s$ with $\neg \varphi_1$ if $s$ is not already labelled with $\varphi_1$.
- $EX\varphi_1$: label $s$ with $EX\varphi_1$ if one of its successor is labelled with $\varphi_1$. 
The labelling algorithm for $EG$

- If $\varphi$ is $EG\varphi_1$:
  - Label all states with $EG\varphi_1$.
  - If any state $s$ is not labelled with $\varphi_1$, delete the label $EG\varphi_1$.
  - Repeat: delete the label $EG\varphi$ from any state if none of its successors is labelled with $EG\varphi_1$, until there is no change.

- If $\varphi$ is $E[\varphi_1 U \varphi_2]$: see book.

See the code at page 227 of the book for the procedure $SAT(\varphi)$ (the pages 225 — 231 are optional).
Summary

You should be able to understand and to solve exercises about:

- CTL syntax and parse trees.
- CTL semantics: transition systems, computation paths, establish whether or not a formula is true at a state in a model.
- Recognise (simple) equivalent formulas.
- Formalise in CTL temporal requirements expressed in plain English.
- Basic ideas about model checking algorithms.
Additional material if you are interested in the subject

- All chapter 3 of the book is worth reading (it introduces LTL and CTL* and the details of model checking algorithms).

Please feel free to contact me for further information.