Formal Verification

Lecture 1: Introduction to Model Checking and Temporal Logic[1]

Jacques Fleuriot
jdf@inf.ed.ac.uk

[1]Acknowledgement: Adapted from original material by Paul Jackson, including some additions by Bob Atkey.

Formal Verification (in a nutshell)

- Create a *formal model* of some system of interest
  - Hardware
  - Communication protocol
  - Software, esp. concurrent software

Formal Verification (in a nutshell)

- Create a *formal model* of some system of interest
  - Hardware
  - Communication protocol
  - Software, esp. concurrent software
- Describe formally a *specification* that we desire the model to satisfy

Formal Verification (in a nutshell)

- Create a *formal model* of some system of interest
  - Hardware
  - Communication protocol
  - Software, esp. concurrent software
- Describe formally a *specification* that we desire the model to satisfy
- Check the model satisfies the specification
  - theorem proving (usually interactive but not necessarily)
  - Model checking

Introduction to Model Checking

- Specifications as Formulas, Programs as Models
- Programs are abstracted as Finite State Machines
- Formulas are in Temporal Logic

Interpretation $\models$ Formula

The relationship between interpretations M and formulas ϕ:

$$M \models \phi$$

We say M **models** ϕ.

Questions we can ask:

Interpretation $\models$ Formula

The relationship between interpretations M and formulas ϕ:

$$M \models \phi$$

We say M **models** ϕ.

Questions we can ask:

1. For a fixed ϕ, is $M \models \phi$ true for all M?
   - Validity of ϕ
   - This can be done via proof in a theorem prover e.g. Isabelle.

Interpretation $\models$ Formula

The relationship between interpretations M and formulas ϕ:

$$M \models \phi$$

We say M **models** ϕ.

Questions we can ask:

1. For a fixed ϕ, is $M \models \phi$ true for all M?
   - Validity of ϕ
   - This can be done via proof in a theorem prover e.g. Isabelle.
2. For a fixed ϕ, is $M \models \phi$ true for some M?
   - Satisfiability

Interpretation $\models$ Formula

The relationship between interpretations M and formulas ϕ:

$$M \models \phi$$

We say M **models** ϕ.

Questions we can ask:

1. For a fixed ϕ, is $M \models \phi$ true for all M?
   - Validity of ϕ
   - This can be done via proof in a theorem prover e.g. Isabelle.
2. For a fixed ϕ, is $M \models \phi$ true for some M?
   - Satisfiability
3. For a fixed (class of) M, what ϕs make $M \models \phi$ true?
   - "Theory discovery"/"Learning from Data"/"Generalisation"
   - Not in this course

Interpretation $\models$ Formula

The relationship between interpretations M and formulas ϕ:

$$M \models \phi$$

We say M **models** ϕ.

Questions we can ask:

1. For a fixed ϕ, is $M \models \phi$ true for all M?
   - Validity of ϕ
   - This can be done via proof in a theorem prover e.g. Isabelle.
2. For a fixed ϕ, is $M \models \phi$ true for some M?
   - Satisfiability
3. For a fixed (class of) M, what ϕs make $M \models \phi$ true?
   - "Theory discovery"/"Learning from Data"/"Generalisation"
   - Not in this course
4. For a fixed M and P, is it the case that $M \models \phi$?
   - Model Checking

Model Checking

At a high level, many tasks can be rephrased as model checking.

"Interpretations" M	$\models$	**"Formulas"** ϕ	**Task**
sequences of tokens	$\models$	grammars	parsing
database tables	$\models$	SQL queries	query execution
email texts	$\models$	spam rules	spam detection
sequences of letters	$\models$	dictionary	spellchecking
audio data	$\models$	acoustic/lang. model	speech recognition
finite state machines	$\models$	temporal logic	specification checking

Details differ widely, but question of "is this data consistent with this statement? (and to what degree?)" is extremely common.

Historically, "Model Checking" usually refers to the last one. This is the one we will cover over the next few lectures.

Uses of Model Checking

Model Checking has been used to:

- Check Microsoft Windows device drivers for bugs
  - The "Static Driver Verifier" tool
- The SPIN tool (`http://spinroot.com`):
  - `http://spinroot.com/spin/success.html`
  - Flood control barrier control software
  - Call processing software at Lucent
  - Parts of *Mars Science Laboratory*, *Deep Space 1*, *Cassini*, *the Mars Exploration Rovers*, *Deep Impact*
  - ...
- PEPA (Performance Evaluation Process Algebra) `http://www.dcs.ed.ac.uk/pepa/`
  - Multiprocessor systems
  - Biological systems
- ...

Model Checking – Models

A model of some system has:

- A finite set of **states**
- A subset of states considered as the **initial states**
- A **transition relation** which, given a state, describes all states that can be reached “in one time step”.

Good for

- Software, sequential and concurrent
- Digital hardware
- Communication protocols

Refinements of this setup can handle: **Infinite state spaces**, **Continuous state spaces**, **Continuous time**, **Probabilistic Transitions**. Good for hybrid (*i.e.,* discrete and continuous) and control systems.

Model Checking – Models

Models are *always* abstractions of reality.

Model Checking – Models

Models are *always* abstractions of reality.

- We must choose what to model and what not to model

Model Checking – Models

Models are *always* abstractions of reality.

- We must choose what to model and what not to model
- There will limitations forced by the formalism
  - *e.g.*, here we are limited to **finite state** models

Model Checking – Models

Models are *always* abstractions of reality.

- We must choose what to model and what not to model
- There will limitations forced by the formalism
  - *e.g.*, here we are limited to **finite state** models
- There will be things we do not understand sufficiently to model
  - *e.g.*, people

Model Checking – Models

Models are *always* abstractions of reality.

- We must choose what to model and what not to model
- There will limitations forced by the formalism
  - *e.g.*, here we are limited to **finite state** models
- There will be things we do not understand sufficiently to model
  - *e.g.*, people

In the words of the **The Cure**'s *Pictures of You*:

I've been looking so long at these pictures of you
That I almost believe that they're real
I've been living so long with my pictures of you
That I almost believe that the pictures are
All I can feel

Model Checking – Models

Models are *always* abstractions of reality.

- We must choose what to model and what not to model
- There will limitations forced by the formalism
  - *e.g.*, here we are limited to **finite state** models
- There will be things we do not understand sufficiently to model
  - *e.g.*, people

In the words of the **The Cure**'s *Pictures of You*:

I've been looking so long at these pictures of you
That I almost believe that they're real
I've been living so long with my pictures of you
That I almost believe that the pictures are
All I can feel

Do not do this: the pictures are not real.

Model Checking – Models

La trahison des images by René Magritte taken from a University of Alabama site, “Approaches to Modernism”: `http://www.tcf.ua.edu/Classes/Jbutler/T311/Modernism.htm`. Licensed under Fair use via Wikipedia - `http://en.wikipedia.org/wiki/File:MagrittePipe.jpg#mediaviewer/File:MagrittePipe.jpg`

Model Checking – Specifications

We are interested in specifying behaviours of systems over time.

- Use **Temporal Logic**

Model Checking – Specifications

We are interested in specifying behaviours of systems over time.

- Use **Temporal Logic**

Specifications are built from:

1. Primitive properties of individual states
   e.g., "is on", "is off", "is active", "is reading";
2. propositional connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$;
3. and **temporal** connectives: *e.g.*,
   At **all times**, the system is not simultaneously *reading* and *writing*.
   If a *request* signal is asserted **at some time**, a corresponding *grant* signal will be asserted **within 10 time units**.

Model Checking – Specifications

We are interested in specifying behaviours of systems over time.

- Use **Temporal Logic**

Specifications are built from:

1. Primitive properties of individual states
   e.g., “is on”, “is off”, “is active”, “is reading”;
2. propositional connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$;
3. and **temporal** connectives: *e.g.*,
   At **all times**, the system is not simultaneously *reading* and *writing*.
   If a *request* signal is asserted **at some time**, a corresponding *grant* signal will be asserted **within 10 time units**.

The exact set of temporal connectives differs across temporal logics.
Logics can differ in how they treat time:

- **Linear time** vs. **Branching time**

These differ in reasoning about *non-determinism*.

Non-determinism

In general, system descriptions are *non-deterministic*.

A system is *non-deterministic* when, from some state there are **multiple** alternative next states to which the system could transition.

Non-determinism is good for:

- Modelling alternative inputs to the system from its environment (*External non-determinism*)
- Under-specifying the model, allowing it to capture many possible system implementations (*Internal non-determinism*)

Linear vs. Branching Time

- **Linear Time**
  - Considers paths (sequences of states)
  - If system is non-deterministic, many paths for each initial state
  - Questions of the form:
    - For all paths, does some path property hold?
    - Does there exist a path such that some path property holds?

Linear vs. Branching Time

- **Linear Time**
  - Considers paths (sequences of states)
  - If system is non-deterministic, many paths for each initial state
  - Questions of the form:
    - For all paths, does some path property hold?
    - Does there exist a path such that some path property holds?
- **Branching Time**
  - Considers tree of possible future states from each initial state
  - If system is non-deterministic from some state, tree forks
  - Questions can become more complex, *e.g.*,
    - For all states reachable from an initial state, does there exist an onwards path to a state satisfying some property?
  - Most-basic branching-time logic (CTL) is complementary to most-basic linear-time logic (LTL)
  - Richer branching-time logic (CTL*) incorporates CTL and LTL.

A Taste of LTL – Syntax

LTL = Linear(-time) Temporal Logic

Assume some set *Atom* of atomic propositions

Syntax of LTL formulas ϕ:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \rightarrow \phi \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \phi\mathbf{U}\phi$$

where $p \in Atom$.

Pronunciation:

- $\mathbf{X}\phi$ – neXt ϕ
- $\mathbf{F}\phi$ – Future ϕ
- $\mathbf{G}\phi$ – Globally ϕ
- $\phi\mathbf{U}\psi$ – ϕ Until ψ

Other common connectives: **W** (weak until), **R** (release).

Precedence high-to-low: $(\mathbf{X}, \mathbf{F}, \mathbf{G}, \neg), (\mathbf{U}), (\wedge, \vee), \rightarrow$

A Taste of LTL – Informal Semantics

LTL formulas are evaluated at a position i along a path π through the system (a path is a sequence of states connected by transitions)

- An atomic p holds if p is true for the state at position i.
- The propositional connectives $\neg, \wedge, \vee, \rightarrow$ have their usual meanings.
- Meaning of LTL connectives:
  - $\mathbf{X}\phi$ holds if ϕ holds at the next position;
  - $\mathbf{F}\phi$ holds if there exists a future position where ϕ holds;
  - $\mathbf{G}\phi$ holds if, for all future positions, ϕ holds;
  - $\phi\mathbf{U}\psi$ holds if there is a future position where ψ holds, and ϕ holds for all positions prior to that.

This will be made more formal in the next lecture.

A Taste of LTL – Examples

1. G *invariant*

   invariant is true for all future positions

A Taste of LTL – Examples

1. $\mathbf{G}\ \mathit{invariant}$

   invariant is true for all future positions

2. $\mathbf{G}\ \neg(\mathit{read} \wedge \mathit{write})$

   In all future positions, it is not the case that *read* and *write*

A Taste of LTL – Examples

1. $\mathbf{G}\ \mathit{invariant}$

   invariant is true for all future positions

2. $\mathbf{G}\ \neg(\mathit{read} \wedge \mathit{write})$

   In all future positions, it is not the case that *read* and *write*

3. $\mathbf{G}(\mathit{request} \rightarrow \mathbf{F}\mathit{grant})$

   At every position in the future, a *request* implies that there exists a future point where *grant* holds.

A Taste of LTL – Examples

1. $\mathbf{G}\ invariant$

   invariant is true for all future positions

2. $\mathbf{G}\ \neg(read \wedge write)$

   In all future positions, it is not the case that *read* and *write*

3. $\mathbf{G}(request \rightarrow \mathbf{F} grant)$

   At every position in the future, a *request* implies that there exists a future point where *grant* holds.

4. $\mathbf{G}(request \rightarrow (request\ \mathbf{U}\ grant))$

   At every position in the future, a *request* implies that there exists a future point where *grant* holds, and *request* holds up until that point.

A Taste of LTL – Examples

1. $\mathbf{G}\ invariant$

   invariant is true for all future positions

2. $\mathbf{G}\ \neg(read \wedge write)$

   In all future positions, it is not the case that *read* and *write*

3. $\mathbf{G}(request \rightarrow \mathbf{F} grant)$

   At every position in the future, a *request* implies that there exists a future point where *grant* holds.

4. $\mathbf{G}(request \rightarrow (request\, \mathbf{U}\ grant))$

   At every position in the future, a *request* implies that there exists a future point where *grant* holds, and *request* holds up until that point.

5. $\mathbf{G}\ \mathbf{F}\ enabled$

   In all future positions, there is a future position where *enabled* holds.

A Taste of LTL – Examples

1. $\mathbf{G}\ \mathit{invariant}$

   invariant is true for all future positions

2. $\mathbf{G}\ \neg(\mathit{read} \land \mathit{write})$

   In all future positions, it is not the case that *read* and *write*

3. $\mathbf{G}(\mathit{request} \rightarrow \mathbf{F}\mathit{grant})$

   At every position in the future, a *request* implies that there exists a future point where *grant* holds.

4. $\mathbf{G}(\mathit{request} \rightarrow (\mathit{request}\ \mathbf{U}\ \mathit{grant}))$

   At every position in the future, a *request* implies that there exists a future point where *grant* holds, and *request* holds up until that point.

5. $\mathbf{G}\ \mathbf{F}\ \mathit{enabled}$

   In all future positions, there is a future position where *enabled* holds.

6. $\mathbf{F}\ \mathbf{G}\ \mathit{enabled}$

   There is a future position, from which all future positions have *enabled* holding.

Summary

- Introduction to Model Checking (H&R 3.1, 3.2)
  - The Model Checking problem
  - Informal introduction to LTL
- Next time:
  - Formal introduction to LTL.