SS 2017
Software Verification
Modal Logics

Prof. Dr. Holger Schlingloff \(^1,^2\)
Dr. Esteban Pavese \(^1\)

(1) Institut für Informatik der Humboldt Universität
(2) Fraunhofer Institut für offene Kommunikationssysteme FOKUS
Quick recap

- Predicate logic vs. Propositional logic?
- Propositional logic satisfiability complexity?
- Is PL a correct and complete logic?
- FOL vs. Propositional logic
- Is FOL a correct and complete logic?
- What about satisfiability?
- Recall the semantics of FOL
Modal logics – a motivation

• Main drive behind MLs in the 1960s
  ▪ Influencial people: Saul Kripke (1940-), Jaako Hintikka (1929-2015)

• However, already introduced by C. I. Lewis (1883-1964) in 1910
  ▪ And actually Aristotle had already discussed about syllogistic logic being too absolute
  ▪ Aristotle wondered about the consequences of potentiallity and time

• Amir Pnueli (1941-2009, 1996 Turing Award) famously proposed the use of a particular ML (temporal logic) for formalising concurrent software behaviour
Modal logics – a motivation

C. I. Lewis (and others) had problems with the *material conditional*

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Modal logics – a motivation

C. I. Lewis (and others) had problems with the *material conditional*

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

*Transitivity*

\{A \to B, B \to C\} \vdash A \to C
Modal logics – a motivation

C. I. Lewis (and others) had problems with the *material conditional*

\[
\begin{array}{ccc}
A & B & A \rightarrow B \\
T & T & T \\
T & F & F \\
F & T & T \\
F & F & T \\
\end{array}
\]

*Transitivity*

\{A \rightarrow B, B \rightarrow C\} \vdash A \rightarrow C

If James Bond was Russian, he would spy for Russia 👍
If James Bond spies for Russia, he would be a traitor to the Crown 👍
If James Bond was Russian, he would be a traitor to the Crown 😐
Modal logics – a motivation

C. I. Lewis (and others) had problems with the *material conditional*

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

*Monotonicity (Left-strengthening)*

\[ A \rightarrow C \models (A \land B) \rightarrow C \]
Modal logics – a motivation

C. I. Lewis (and others) had problems with the *material conditional*

\[
\begin{array}{|c|c|c|}
\hline
A & B & A \rightarrow B \\
\hline
T & T & T \\
T & F & F \\
F & T & T \\
F & F & T \\
\hline
\end{array}
\]

**Monotonicity (Left-strengthening)**

\[A \rightarrow C \vdash (A \land B) \rightarrow C\]

If I put Ketchup in my Bratwurst, it tastes great 👍

If I put Ketchup **and** curry in my Bratwurst, it tastes great 💩 (*)

(*) Yeah, I hate Currywurst!
Modal logics – a motivation

Dorothy Edgington famously “proved” God’s existence

If God does not exist, then it is not the case that when I pray, the prayer is answered

\[ \neg G \rightarrow \neg (P \rightarrow A) \]

Ok, so I don’t pray

\[ \neg P \]

But!

\[ \{ \neg G \rightarrow \neg (P \rightarrow A), \neg P \} \vdash G \]

If I don’t pray, God exists! 😱
A question of necessity

Essentially, there is a deep difference between an absolute necessity (material implication), a *contingent* necessity, and a *possibility*

- “All men are mortal. Socrates is a man. Socrates is mortal” (absolute)
- “Nothing can travel faster than the speed of light” (contingent on the laws of nature)
- “If it rains, I get wet” (possibility)
Modal logic syntax

- ML is an extension of PL with an additional *modal necessity* operator noted $\Box$
- Formulae are built inductively:

$$ML := \bot \mid p \in P \mid \neg ML \mid (ML \rightarrow ML) \mid \Box ML$$
Modal logic syntax - shorthand

The shorthand and derived operators from Propositional Logic can also be used in Modal Logic.

Additionally, the *possibility* operator is also derived from the *necessity* operator.

\[ \Diamond \phi = \neg \Box \neg \phi \]
Modal logic semantics

- As was the case for PL and FOL, we require a *model* in order to establish whether a formula is true or not.
- Again, a formula will be valid if it is true for any possible model.

- We must also specify an axiomatic system defining ML:
  - Actually, several different axiomatic systems, that define several different logics.
  - We will see to this later on.
Modal logic – Kripke models

- A Kripke model is a tuple $M = <W, R, V>$ where
  - $W$ is a non-empty set. The elements of $W$ are called *worlds*.
  - $R \subseteq W \times W$ is called the *accessibility relation*. We say world $v$ is *accessible* from world $u$ whenever $uRv$.
  - $V: \mathcal{P} \to 2^W$ is the valuation function mapping propositional variables from $\mathcal{P}$ to the set of worlds where the variable is assigned the true value.

- The graph described by $(W, R)$ is called the *frame* of the Kripke Model.
Modal logic – Kripke models

Example

\[ W = \{ s_0, s_1, s_2, s_3 \} \]

\[ R = \{ (s_0, s_1), (s_1, s_2), (s_1, s_3), (s_2, s_3), (s_3, s_2), (s_3, s_3) \} \]

\[ V(p) = \{ s_0, s_1 \} \]
\[ V(q) = \{ s_0, s_1, s_2 \} \]
\[ V(r) = \{ s_0, s_2, s_3 \} \]
Modal logic – Satisfiability

Recall $W$ is a set of worlds. A formula may be true in some world in a frame, but false in another world. The satisfiability definition must take into account

\begin{align*}
M, w \models \bot & \iff False \tag{1} \\
M, w \models p & \iff w \in V(p) \tag{2} \\
M, w \models \neg \phi & \iff M, w \not\models \phi \tag{3} \\
M, w \models \phi \rightarrow \psi & \iff M, w \models \phi \implies M, w \models \psi \tag{4} \\
M, w \models \Box \phi & \iff \forall v \ wRv \implies M, v \models \phi \tag{5}
\end{align*}

Additionally, we note $M \models \phi$ if every world in $W$ satisfies the formula $\phi$.
OK, stop!...wasn’t this a Software Verification course?
Modal logic as a Software Engineering tool (*)

Yes! Although modal logic has been historically defined as necessity logic, a modality can really be any predicate that modifies a predicate.

Example:

Once a message is sent by the Server, a Client:

- *Necessarily* receives it
- *May* receive it
- *Will* receive it *at the next time point*
- *May* receive it *at the next time point*
- *Knows* that it has been sent
- *Believes* that it has been sent
- *Is necessarily authorised* to read it
- ... and many more

We will see later that the (future) time modality is really useful for verification.

(*) “Coincidentally” the title of an insightful paper by D. Hirsch and C. Areces
Modal logic – a bit more

- As said before, we can also think of ML in terms of axiomatic systems
- The system that corresponds to the model of Kripke structures is captured by PL axioms, the operator \( \Box \) and two additional axioms
  - Necessitation (N) \( p \vdash \Box p \)
  - Distribution (K) \( \vdash \Box (p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q) \)
- Theorem: every Kripke model satisfies both axioms N and K
Modal logic – a bit more

- The axioms of this modal logic are the axioms of PL plus N and K
- The logic K as it is called is the weakest modal logic

\[ \square p \rightarrow p \]
\[ \square p \rightarrow \square \square p \]
\[ \square \square p \rightarrow \square p \]

Some formulae that seem intuitively true, cannot be proven in K.

The solution is easy! Add them as axioms and claim a new logic!
Other desirable properties

\[ \Box p \rightarrow p \]

\[ \Box p \rightarrow \Box \Box p \]

\[ \Box \Box p \rightarrow \Box p \]

Some formulae that seem intuitively true, cannot be proven in \( \textbf{K} \).

The solution is easy! Add them as axioms and claim a new logic!
Modal logic – a bit more

\[ \square p \rightarrow p \quad \text{(Logic } T) \]

\[ \square p \rightarrow \square \square p \quad \text{(Logic } S4) \]

\[ \square \square p \rightarrow \square p \]

So the axiomatic system is fixed, but what about the Kripke model semantics?
Modal logic – frame conditions

It turns out it suffices to place restrictions on the *frames* alone in order to imply the required axioms.

\[ \Box p \rightarrow p \]

Reflexive frame: \( \forall w \ wRw \)

\[ \Box p \rightarrow \Box \Box p \]

Transitive frame: \( \forall u, v, w \ (uRv \land vRw \rightarrow uRw) \)

\[ \Box \Box p \rightarrow \Box p \]

Dense frame: \( \forall u, v \ (uRv \rightarrow \exists w \ uRw \land wRv) \)
Modal logic – expressibility

Question 1: given a (satisfiable) set of ML formulas, is the satisfying model unique?

Question 2: given a (satisfiable) set of ML formulas, and two satisfying models, what can we do with ML to distinguish them?

Question 3: is it always the case that ML can distinguish between two different models?

What are the implications of these answers, if we will be modelling and analysing software systems with ML?
ML vs. PL and FOL

- It can easily be seen that PL \(\subset\) ML. We just prescind with the necessity operator.
- It is more interesting to note that ML \(\subset\) FOL.

- The *standard translation* gives a procedure for translating an ML formula \(\phi\) into an equivalent FOL formula \(\psi\).
- That is, such that \(\phi\) has an ML model iff. \(\psi\) has an FOL model.
Standard translation

\[
ST_x(\bot) \equiv \bot \quad (1)
\]

\[
ST_x(\neg \varphi) \equiv \neg ST_x(\varphi) \quad (2)
\]

\[
ST_x(\varphi \to \psi) \equiv ST_x(\varphi) \to ST_x(\psi) \quad (3)
\]

\[
ST_x(\Box \varphi) \equiv \forall y \, R(x, y) \to ST_y(\varphi) \quad (4)
\]

Through careful reuse of variables during the translation, it can be shown that \(ST(ML)\) corresponds to a fragment of FOL with

- One (binary) predicate symbol \((R)\)
- And only using two variables

This fragment known as **FO2** is **decidable**, therefore \(ML\) is decidable.
Modal logic as a Software Engineering tool (*)

We have already hinted at the usefulness of Modal Logic for verification

However dealing with Kripke Structures is cumbersome

In practice, *Finite State Machines* (also called Labelled Transition Systems) are good abstractions of software systems, that also happen to be embeddable in Kripke structures

The tools and techniques that we will explore in the rest of the semester are based on these Finite State Machines and some of their variants

(*) “Coincidentally” the title of an insightful paper by D. Hirsch and C. Areces
Finite State Machines

A Finite State Machine (FSM) is a tuple $M=\langle S, s_0, A, AP, R, V \rangle$ where

- $S$ is a non-empty, finite set of states
- $s_0 \in S$ is the initial state
- $A$ is a finite set of action labels
- $AP$ is the set of propositional variables
- $R \subseteq S \times A \times S$ is the (labelled) transition relation
- $V: AP \rightarrow 2^S$ is the valuation function

Essentially, an FSM is a Kripke Structure where the edges are labelled with an action
FSM example
Future lectures

- We will revisit those pesky modelling questions
- We will start modelling software with FSMs!
- We will focus on *concurrent* system modelling
  - (after all, that’s where the fun is)