Modal Logics for Multi-Agent Systems
Jürgen Dix and Wojtek Jamroga

Department of Computer Science
Clausthal University of Technology, Germany
{dix,wjamroga}@in.tu-clausthal.de

European Agent Systems Summer School
Annecy 2006

Lecture Overview

1. Classical Logic
2. Modal Logic
3. Logics for Action and Time
4. Combining Modalities
5. Modalities in Action

About this lecture
This course is based, among other papers listed in the particular chapters, mainly on the following books and articles.


Organization:
Part 1: Jürgen Dix,
Part 2: Wojtek Jamroga,

Extended version of the slides (with references etc.) will be available at:
http://www2.in.tu-clausthal.de/~wjamroga/courses/MAS2006EASSS/

Feel free to write if you have any questions!
Chapter 1. Classical Logic

Classical Logic

1.1 Why Logic?
1.2 Sentential Logic
1.3 Sudoku
1.4 Calculi for SL
1.5 Wumpus in SL

In this chapter:
- We make the case for using formal logic as a representation formalism for reasoning about the world.
- We present classical propositional logic to introduce the standard language and formalism.
- The semantics of propositional logic is the basis of all other logics. It can be used to model many things: we show its versatility by modeling Sudoku-puzzles.
- We also describe two calculi for this logic and consider briefly correctness and completeness issues.
- We illustrate how to use propositional logic with the well-known Wumpus world.

Temporal and modal logic.

*Temporal Logics*.
Kluwer.

Logic-based specification languages for intelligent software agents.

*Multi Agent Systems*.

*Heterogenous Active Agents*.
MIT-Press.

*Multi-Agent Systems*.
MIT-Press.

*Reasoning about Rational Agents*.
MIT Press.

*An Introduction to MultiAgent Systems*.
John Wiley & Sons.
1. Classical Logic

1. Why Logic?

1.1 Why Logic?

Why logic at all?

- framework for **thinking** about systems,
- makes one **realise** many **assumptions**,
- ... and then we can:
- **investigate** them, **accept or reject** them,
- **relax** some of them and still use a part of the formal and conceptual machinery,

Symbolic AI: Symbolic representation, e.g. sentential or first order logic. **Agent as a theorem prover**.

Traditional: Theory about agents. Implementation as stepwise process (Software Engineering) over many abstractions.

Symbolic AI: View the theory itself as **executable specification**. Internal state: **Knowledge Base** (KB), often simply called **D** (database).
1.2 Sentential Logic

Definition 1.1 (Sentential Logic $L_{SL}$, Language $L \subseteq L_{SL}$)

The language $L_{SL}$ of propositional (or sentential) logic consists of
- $\square$ and $\top$: falsum and verum,
- $p, q, r, x_1, x_2, \ldots, x_n, \ldots$: a countable set $\mathcal{AT}$ of SL-constants,
- $\neg, \land, \lor, \rightarrow$: the sentential connectives ($\neg$ is unary, all others are binary operators),
- $\langle, \rangle$: the parantheses to help readability.

Generally we consider only a finite set of SL-constants. They define a language $L \subseteq L_{SL}$. The set of $L$-formulae $\text{Fml}_L$ is defined inductively.

Definition 1.2 (Semantics, Valuation, Model)

A valuation $\nu$ for a language $L \subseteq L_{SL}$ is a mapping from the set of SL-constants defined through $L$ and $\{\top, \square\}$ into the set $\{\text{true}, \text{false}\}$ with $\nu(\square) = \text{false}$, $\nu(\top) = \text{true}$. Each valuation $\nu$ can be uniquely extended to a function $\bar{\nu}: \text{Fml}_L \to \{\text{true}, \text{false}\}$ so that:
- $\bar{\nu}(\neg p) = \begin{cases} \text{true}, & \text{if } \bar{\nu}(p) = \text{false} \\ \text{false}, & \text{if } \bar{\nu}(p) = \text{true} \end{cases}$
- $\bar{\nu}(\varphi \land \gamma) = \begin{cases} \text{true}, & \text{if } \bar{\nu}(\varphi) = \text{true} \text{ and } \bar{\nu}(\gamma) = \text{true} \\ \text{false}, & \text{else} \end{cases}$
- $\bar{\nu}(\varphi \lor \gamma) = \begin{cases} \text{true}, & \text{if } \bar{\nu}(\varphi) = \text{true} \text{ or } \bar{\nu}(\gamma) = \text{true} \\ \text{false}, & \text{else} \end{cases}$

Thus each valuation $\nu$ uniquely defines a $\bar{\nu}$. We call $\bar{\nu}$ $L$-structure or model $\mathcal{A}_\nu$. From now in we will speak of models and valuations.

A model determines for each formula if it is true or false. The process of mapping a set of $L$-formulae into $\{\text{true}, \text{false}\}$ is called semantics.
**Definition 1.3 (Validity of a Formula, Tautology)**

1. A formula $\varphi \in \text{Fml}_L$ holds under the valuation $v$ if $\bar{v}(\varphi) = \text{true}$. We also write $v \models \varphi$ or simply $v \models \varphi$.
2. A **theory** is a set of formulae: $T \subseteq \text{Fml}_L$. $v$ satisfies $T$ if $\bar{v}(\varphi) = \text{true}$ for all $\varphi \subseteq T$. We write $v \models T$.
3. A $L$-formula $\varphi$ is called **$L$-tautology** if for all possible valuations (models) $v$ in $L$ $v \models \varphi$ holds.

From now on we suppress the language $L$, because it is obvious from context. Nevertheless it needs to be carefully defined.

**Definition 1.4 (Consequence Set $\text{Cn}(T)$)**

A formula $\varphi$ results from $T$ if for all models $v$ with $v \models T$ also $v \models \varphi$ holds. We write: $T \models \varphi$.

We call $\text{Cn}_L(T) = \text{def} \left\{ \varphi \in \text{Fml}_L : T \models \varphi \right\}$, or simply $\text{Cn}(T)$, the **semantic consequence operator**.

**Lemma 1.5 (Properties of $\text{Cn}(T)$)**

The semantic consequence operator has the following properties:

1. **$T$-expansion**: $T \subseteq \text{Cn}(T)$,
2. **Monotony**: $T \subseteq T' \Rightarrow \text{Cn}(T) \subseteq \text{Cn}(T')$,
3. **Closure**: $\text{Cn}(\text{Cn}(T)) = \text{Cn}(T)$.

**Lemma 1.6 ($\varphi \notin \text{Cn}(T)$)**

$\varphi \notin \text{Cn}(T)$ if and only if there is a model $v$ with $v \models T$ and $\bar{v}(\varphi) = \text{false}$.

**Definition 1.7 (MOD($T$), $\text{Cn}(\mathcal{U})$)**

If $T \subseteq \text{Fml}_L$ then we denote with $\text{MOD}(T)$ the set of all $L$-structures $A$ which are models of $T$:

$$\text{MOD}(T) = \text{def} \left\{ A : A \models T \right\}.$$

If $\mathcal{U}$ is a set of models, we consider all those sentences, which are valid in all models of $\mathcal{U}$. We call this set $\text{Cn}(\mathcal{U})$:

$$\text{Cn}(\mathcal{U}) = \text{def} \left\{ \varphi \in \text{Fml}_L : \forall v \in \mathcal{U} : \bar{v}(\varphi) = \text{true} \right\}.$$

$\text{MOD}$ is obviously dual to $\text{Cn}$:

$$\text{Cn}(\text{MOD}(T)) = \text{Cn}(T), \quad \text{MOD}(\text{Cn}(T)) = \text{MOD}(T).$$
1. Classical Logic 2. Sentential Logic

**Definition 1.8 (Completeness of a Theory \( T \))**

\( T \) is called **complete** if for each formula \( \varphi \in \text{Fml} \): \( T \models \varphi \) or \( T \models \neg \varphi \) holds.

**Attention:**

Do not mix up this last condition with the property of a valuation (model) \( \nu \): each model is complete in the above sense.

---

**Definition 1.9 (Consistency of a Theory)**

\( T \) is called **consistent** if there is a valuation (model) \( \nu \) with \( \nu(\varphi) = \text{true} \) for all \( \varphi \in T \).

**Lemma 1.10 (Ex Falso Quodlibet)**

\( T \) is consistent if and only if \( \text{Cn}(T) \neq \text{Fml}_L \).

---


**1.3 Sudoku**

Since some time, **Sudoku** puzzles are becoming quite famous.

Table 1: A simple Sudoku (\( S_1 \))
Can they be solved with sentential logic?

Idea: Given a Sudoku-Puzzle $S$, construct a language $L_{\text{Sudoku}}$ and a theory $T_S \subseteq \text{Fml}_{L_{\text{Sudoku}}}$ such that

$$\text{MOD}(T_S) = \text{Solutions of the puzzle } S$$

Solution

In fact, we construct a theory $T_{\text{Sudoku}}$ and for each (partial) instance of a $9 \times 9$ puzzle $S$ a particular theory $T_S$ such that

$$\text{MOD}(T_{\text{Sudoku}} \cup T_S) = \{S : S \text{ is a solution of } S\}$$

We have to give our symbols a meaning: the semantics!

$\text{eins}_{i,j}$ means $i, j$ contains a 1
$\text{zwei}_{i,j}$ means $i, j$ contains a 2
$\text{drei}_{i,j}$ means $i, j$ contains a 3
$\text{vier}_{i,j}$ means $i, j$ contains a 4
$\text{fuenf}_{i,j}$ means $i, j$ contains a 5
$\text{sechs}_{i,j}$ means $i, j$ contains a 6
$\text{sieben}_{i,j}$ means $i, j$ contains a 7
$\text{acht}_{i,j}$ means $i, j$ contains a 8
$\text{neun}_{i,j}$ means $i, j$ contains a 9

To be precise: given a $9 \times 9$ square that is completely filled out, we define our valuation $v$ as follows (for all $1 \leq i, j \leq 9$).

$$v(\text{eins}_{i,j}) = \begin{cases} \text{true}, & \text{if 1 is at position } (i, j), \\ \text{false}, & \text{else} \end{cases}$$

\[ v(\text{zwei}_{i,j}) = \begin{cases} \text{true}, & \text{if 2 is at position } (i, j), \\ \text{false}, & \text{else}. \end{cases} \]

\[ v(\text{drei}_{i,j}) = \begin{cases} \text{true}, & \text{if 3 is at position } (i, j), \\ \text{false}, & \text{else}. \end{cases} \]

\[ v(\text{vier}_{i,j}) = \begin{cases} \text{true}, & \text{if 4 is at position } (i, j), \\ \text{false}, & \text{else}. \end{cases} \]

usw.

\[ v(\text{neun}_{i,j}) = \begin{cases} \text{true}, & \text{if 9 is at position } (i, j), \\ \text{false}, & \text{else}. \end{cases} \]

Therefore any \(9 \times 9\) square can be seen as a model or valuation with respect to the language \(L_{\text{Sudoku}}\).

---

How should the theory \(T_{\text{Sudoku}}\) look like (s.t. models of \(T_{\text{Sudoku}} \cup T_S\) correspond to solutions of the puzzle)?

**First square:** \(T_1\)

1. \(\text{eins}_{1,1} \lor \ldots \lor \text{eins}_{3,3}\)
2. \(\text{zwei}_{1,1} \lor \ldots \lor \text{zwei}_{3,3}\)
3. \(\text{drei}_{1,1} \lor \ldots \lor \text{drei}_{3,3}\)
4. \(\text{vier}_{1,1} \lor \ldots \lor \text{vier}_{3,3}\)
5. \(\text{fuenf}_{1,1} \lor \ldots \lor \text{fuenf}_{3,3}\)
6. \(\text{sechs}_{1,1} \lor \ldots \lor \text{sechs}_{3,3}\)
7. \(\text{sieben}_{1,1} \lor \ldots \lor \text{sieben}_{3,3}\)
8. \(\text{acht}_{1,1} \lor \ldots \lor \text{acht}_{3,3}\)
9. \(\text{neun}_{1,1} \lor \ldots \lor \text{neun}_{3,3}\)

The formulae on the last slide are saying, that

- The number 1 must appear somewhere in the first square.
- The number 2 must appear somewhere in the first square.
- The number 3 must appear somewhere in the first square.
- etc

\textbf{Does that mean, that each number }1, \ldots, 9 \textbf{ occurs exactly once in the first square?}
No! We have to say, that each number occurs only once:

\[ T'_1 : \]
1. \( \neg(\text{eins}_{i,j} \land \text{zwei}_{i,j}), \ 1 \leq i, j \leq 3, \]
2. \( \neg(\text{eins}_{i,j} \land \text{drei}_{i,j}), \ 1 \leq i, j \leq 3, \]
3. \( \neg(\text{eins}_{i,j} \land \text{vier}_{i,j}), \ 1 \leq i, j \leq 3, \]
4. etc
5. \( \neg(\text{zwei}_{i,j} \land \text{drei}_{i,j}), \ 1 \leq i, j \leq 3, \]
6. \( \neg(\text{zwei}_{i,j} \land \text{vier}_{i,j}), \ 1 \leq i, j \leq 3, \]
7. \( \neg(\text{zwei}_{i,j} \land \text{fuenf}_{i,j}), \ 1 \leq i, j \leq 3, \]
8. etc

How many formulae are these?

Second square: \( T'_2 \)

- \( \text{eins}_{1,4} \lor \ldots \lor \text{eins}_{3,6} \)
- \( \text{zwei}_{1,4} \lor \ldots \lor \text{zwei}_{3,6} \)
- \( \text{drei}_{1,4} \lor \ldots \lor \text{drei}_{3,6} \)
- \( \text{vier}_{1,4} \lor \ldots \lor \text{vier}_{3,6} \)
- \( \text{fuenf}_{1,4} \lor \ldots \lor \text{fuenf}_{3,6} \)
- \( \text{sechs}_{1,4} \lor \ldots \lor \text{sechs}_{3,6} \)
- \( \text{sieben}_{1,4} \lor \ldots \lor \text{sieben}_{3,6} \)
- \( \text{acht}_{1,4} \lor \ldots \lor \text{acht}_{3,6} \)
- \( \text{neun}_{1,4} \lor \ldots \lor \text{neun}_{3,6} \)

And all the other formulae from the previous slides (adapted to this case): \( T'_2 \)

The same has to be done for all 9 squares.

What is still missing:

**Rows:** Each row should contain exactly the numbers from 1 to 9 (no number twice).

**Columns:** Each column should contain exactly the numbers from 1 to 9 (no number twice).
1. Classical Logic

First Row: $T_{\text{Row 1}}$

1. $\text{eins}_{1,1} \lor \text{eins}_{1,2} \lor \ldots \lor \text{eins}_{1,9}$
2. $\text{zwei}_{1,1} \lor \text{zwei}_{1,2} \lor \ldots \lor \text{zwei}_{1,9}$
3. $\text{drei}_{1,1} \lor \text{drei}_{1,2} \lor \ldots \lor \text{drei}_{1,9}$
4. $\text{vier}_{1,1} \lor \text{vier}_{1,2} \lor \ldots \lor \text{vier}_{1,9}$
5. $\text{fuenf}_{1,1} \lor \text{fuenf}_{1,2} \lor \ldots \lor \text{fuenf}_{1,9}$
6. $\text{sechs}_{1,1} \lor \text{sechs}_{1,2} \lor \ldots \lor \text{sechs}_{1,9}$
7. $\text{sieben}_{1,1} \lor \text{sieben}_{1,2} \lor \ldots \lor \text{sieben}_{1,9}$
8. $\text{acht}_{1,1} \lor \text{acht}_{1,2} \lor \ldots \lor \text{acht}_{1,9}$
9. $\text{neun}_{1,1} \lor \text{neun}_{1,2} \lor \ldots \lor \text{neun}_{1,9}$

First Column: $T_{\text{Column 1}}$

1. $\text{eins}_{1,1} \lor \text{eins}_{2,1} \lor \ldots \lor \text{eins}_{9,1}$
2. $\text{zwei}_{1,1} \lor \text{zwei}_{2,1} \lor \ldots \lor \text{zwei}_{9,1}$
3. $\text{drei}_{1,1} \lor \text{drei}_{2,1} \lor \ldots \lor \text{drei}_{9,1}$
4. $\text{vier}_{1,1} \lor \text{vier}_{2,1} \lor \ldots \lor \text{vier}_{9,1}$
5. $\text{fuenf}_{1,1} \lor \text{fuenf}_{2,1} \lor \ldots \lor \text{fuenf}_{9,1}$
6. $\text{sechs}_{1,1} \lor \text{sechs}_{2,1} \lor \ldots \lor \text{sechs}_{9,1}$
7. $\text{sieben}_{1,1} \lor \text{sieben}_{2,1} \lor \ldots \lor \text{sieben}_{9,1}$
8. $\text{acht}_{1,1} \lor \text{acht}_{2,1} \lor \ldots \lor \text{acht}_{9,1}$
9. $\text{neun}_{1,1} \lor \text{neun}_{2,1} \lor \ldots \lor \text{neun}_{9,1}$

Analogously for all other rows, eg.

Ninth Row: $T_{\text{Row 9}}$

1. $\text{eins}_{9,1} \lor \text{eins}_{9,2} \lor \ldots \lor \text{eins}_{9,9}$
2. $\text{zwei}_{9,1} \lor \text{zwei}_{9,2} \lor \ldots \lor \text{zwei}_{9,9}$
3. $\text{drei}_{9,1} \lor \text{drei}_{9,2} \lor \ldots \lor \text{drei}_{9,9}$
4. $\text{vier}_{9,1} \lor \text{vier}_{9,2} \lor \ldots \lor \text{vier}_{9,9}$
5. $\text{fuenf}_{9,1} \lor \text{fuenf}_{9,2} \lor \ldots \lor \text{fuenf}_{9,9}$
6. $\text{sechs}_{9,1} \lor \text{sechs}_{9,2} \lor \ldots \lor \text{sechs}_{9,9}$
7. $\text{sieben}_{9,1} \lor \text{sieben}_{9,2} \lor \ldots \lor \text{sieben}_{9,9}$
8. $\text{acht}_{9,1} \lor \text{acht}_{9,2} \lor \ldots \lor \text{acht}_{9,9}$
9. $\text{neun}_{9,1} \lor \text{neun}_{9,2} \lor \ldots \lor \text{neun}_{9,9}$

Is that sufficient? What if a row contains several 1’s?

Analogously for all other columns.

All put together:

$T_{\text{Sudoku}} = T_1 \cup T'_1 \cup \ldots \cup T_9 \cup T'_9$

$T_{\text{Row 1}} \cup \ldots \cup T_{\text{Row 9}}$

$T_{\text{Column 1}} \cup \ldots \cup T_{\text{Column 9}}$

Is that sufficient? What if a column contains several 1’s?
Here is a more difficult one.

Table 3: A difficult Sudoku S\text{Schwer}

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

For example smodels uses the following constructs:

1. row(0..8) is a shorthand for row(0), row(1), ..., row(8).
2. val(1..9) is a shorthand for val(1), val(2), ..., val(9).
3. The constants 1, ..., 9 will be treated as numbers (so there are operations available to add, subtract or divide them).

Statements in smodels are written as

\[
\text{head} \leftarrow \text{body}
\]

This corresponds to an implication

\[
\text{body} \rightarrow \text{head}.
\]

An important feature in smodels is that all atoms that do not occur in any head, are automatically false.

For example the theory

\[
p(X, Y, 5) \leftarrow \text{row}(X), \text{col}(Y)
\]

means that the whole grid is filled with 5’s and only with 5’s: eg. \( \neg p(X, Y, 1) \) is true for all \( X, Y \), as well as \( \neg p(X, Y, 2) \) etc.
More constructs in **smodels**

1. \{ \ p(X,Y,A) : \text{val}(A) \ \}  
   \[\text{- row}(X), \text{col}(Y)\]
   this makes sure that in all entries of the grid, exactly one number (\text{val}()) is contained.

2. \{ \ p(X,Y,A) : \text{row}(X) : \text{col}(Y)  
   : \text{eq}(\text{div}(X,3), \text{div}(R,3))  
   : \text{eq}(\text{div}(Y,3), \text{div}(C,3)) \}  
   \[\text{- val}(A), \text{row}(R), \text{col}(C)\]
   this rule ensures that in each of the 9 squares each number from 1 to 9 occurs only once.

3. More detailed info is on the web-page of this lecture.

---

**Definition 1.11 (Axiom, Inference Rule)**

Axioms in SL are the following formulae:

1. \( \varphi \rightarrow \top, \square \rightarrow \varphi, \neg \top \rightarrow \square, \square \rightarrow \neg \top \),
2. \((\varphi \land \psi) \rightarrow \varphi, (\varphi \land \psi) \rightarrow \psi, \)
3. \(\varphi \rightarrow (\varphi \lor \psi), \psi \rightarrow (\varphi \lor \psi), \)
4. \(\neg \neg \varphi \rightarrow \varphi, (\varphi \rightarrow \psi) \rightarrow ((\varphi \rightarrow \neg \psi) \rightarrow \neg \varphi), \)
5. \(\varphi \rightarrow (\psi \rightarrow \varphi), \varphi \rightarrow (\psi \rightarrow (\varphi \land \psi)). \)

Note that \(\varphi, \psi\) stand for arbitrarily complex formulae (not just constants).

---

**Definition (continued)**

The only inference rule in SL is **modus ponens**: 

\[ MP : \text{Fml} \times \text{Fml} \rightarrow \text{Fml} : (\varphi, \varphi \rightarrow \psi) \rightarrow \psi. \]

or short 

\[ (MP) \quad \varphi, \varphi \rightarrow \psi. \]

(\(\varphi, \psi\) are arbitrary complex formulae).
Definition 1.12 (Proof)

A proof of a formula $\varphi$ from a theory $T \subseteq \text{Fml}_L$ is a sequence $\varphi_1, \ldots, \varphi_n$ of formulae such that $\varphi_n = \varphi$ and for all $i$ with $1 \leq i \leq n$ one of the following conditions holds:

- $\varphi_i$ is substitution instance of an axiom,
- $\varphi_i \in T$,
- there is $\varphi_l, \varphi_k = (\varphi_l \rightarrow \varphi_i)$ with $l, k < i$. Then $\varphi_i$ is the result of the application of modus ponens on the predecessor-formulae of $\varphi_i$.

We write: $T \vdash \varphi$ (can be derived from $T$).

Theorem 1.13 (Completeness)

A formula follows semantically from a theory $T$ if and only if it can be derived:

$T \models \varphi$ if and only if $T \vdash \varphi$

Theorem 1.14 (Compactness)

A formula semantically follows from a theory $T$ if and only if it already follows from a finite subset of $T$:

$Cn(T) = \bigcup \{Cn(T') : T' \subseteq T, T' \text{finite}\}$.

Although the axioms from above and modus ponens suffice it is reasonable to consider more general systems. Therefore we introduce the following term:

Definition 1.15 (Rule System $\text{MP} + D$)

Let $D$ be a set of general inference rules, e.g. mappings, which assign a formula $\psi$ to a finite set of formulae $\varphi_1, \varphi_2, \ldots, \varphi_n$. We write

$\varphi_1, \varphi_2, \ldots, \varphi_n \rightarrow \psi$.

$\text{MP} + D$ is the rule systems which emerges from adding the rules in $D$ to modus ponens. For $W \subseteq \text{Fml}$ be

$Cn^D(W)$

in the following the set of all formulae $\varphi$, which can be derived from $W$ and the inference rules from $\text{MP} + D$. We show that:

1. $\vdash \top$,
2. $\vdash \neg \square$,
3. $A \vdash A \lor B$,
4. $A \lor \neg A$,
5. the rule $A \rightarrow \varphi, \neg A \rightarrow \psi$ can be derived.
We bear in mind that $Cn(W)$ is defined semantically but $Cn^D(W)$ is defined syntactically (using the notion of proof). Both sets are equal according to the completeness-theorem in the special case $D = \emptyset$.

**Lemma 1.16 (Properties of $Cn^D$)**

Let $D$ be a set of general inference rules and $W \subseteq Fml$. Then:

1. $Cn(W) \subseteq Cn^D(W)$.
2. $Cn^D(Cn^D(W)) = Cn^D(W)$.
3. $Cn^D(W)$ is the smallest set which is closed in respect to $D$ and contains $W$.

**Question:**

What is the difference between an inference rule $\phi \psi$ and the implication $\phi \rightarrow \psi$?

Assume we have a set $T$ of formulae and we choose two constants $p, q \in L$. We could either consider

1. $T$ together with MP and $\{p q\}$

   or

2. $T \cup \{p \rightarrow q\}$ together with MP

If $T = \{-q\}$, then we have in (2):

$\neg p \in Cn(T \cup \{p \rightarrow q\})$, but not in (1).

**Normalform**

Instead of working on arbitrary formulae, it is sometimes easier to work on finite sets of clauses.
A resolution calculus for SL

Given be a set $M$ of clauses of the form

$$A \lor \neg B \lor C \lor \ldots \lor \neg E$$

Such a clause is also written as the set

$$\{A, \neg B, C, \ldots, \neg E\}.$$  

We define the following inference rule:

**Definition 1.17 (SL resolution)**

Deduce the clause $C_1 \cup C_2$ from $C_1 \cup \{A\}$ and $C_2 \cup \{\neg A\}$.

**Question:**

Is this calculus correct and complete?

**Answer:**

No!

But every problem of the kind $\vdash T \models \phi$ is equivalent to

$\vdash T \cup \{\neg \phi\}$ is unsatisfiable

or rather to

$\vdash T \cup \{\neg \phi\} \vdash \Box$

($\vdash \Box$ stands for the calculus introduced above).

**Theorem 1.18 (Completeness of Resolution Refutation)**

If $M$ is an unsatisfiable set of clauses then the empty clause $\Box$ can be derived from $M$ using only the resolution rule.

We also say that resolution is refutation complete.
1. Classical Logic

5. Wumpus in SL

Language definition:

\[ S_{i,j} \quad \text{stench} \]
\[ B_{i,j} \quad \text{breeze} \]
\[ P_{i,j} \quad \text{is a pit} \]
\[ G_{i,j} \quad \text{glitters} \]
\[ W_{i,j} \quad \text{contains Wumpus} \]

General knowledge:

\[ \neg S_{1,1} \rightarrow (\neg W_{1,1} \land \neg W_{1,2} \land \neg W_{2,1}) \]
\[ \neg S_{2,1} \rightarrow (\neg W_{1,1} \land \neg W_{2,1} \land \neg W_{2,2} \land \neg W_{3,1}) \]
\[ \neg S_{1,2} \rightarrow (\neg W_{1,1} \land \neg W_{1,2} \land \neg W_{2,2} \land \neg W_{1,3}) \]
\[ S_{1,2} \rightarrow (W_{1,3} \lor W_{1,2} \lor W_{2,2} \lor W_{1,1}) \]

Knowledge after the 3rd move:

\[ \neg S_{1,1} \land \neg S_{2,1} \land S_{1,2} \land \neg B_{1,1} \land \neg B_{2,1} \land \neg B_{1,2} \]

Question:
Can we deduce that the wumpus is located at (1,3)?

Answer:
Yes. Either via resolution or using our Hilbert-calculus.
Programming versus knowledge engineering

choose language
write program
write compiler
run program

choose logic
define knowledge base
implement calculus
derive new facts

In this chapter:
- We extend propositional logic by modality operators. They can express beliefs, knowledge, etc.
- We introduce the semantics of modal logics, based on Kripke structures.
- Finally we consider variants of modal logics representing obligations and permissions.

Modal logic is an extension of classical logic by new connectives □ and ◊: necessity and possibility.

- “□p is true” means p is necessarily true, i.e. true in every possible scenario,
- “◊p is true” means p is possibly true, i.e. true in at least one possible scenario.

Independently of the precise definition, the following holds

◊p ↔ ¬□¬p
2. Modal Logic

Various modal logics:
- knowledge → epistemic logic,
- beliefs → doxastic logic,
- obligations → deontic logic,
- actions → dynamic logic,
- time → temporal logic,
- and combinations of the above:
  - most famous multimodal logics: BDI logics of beliefs, desires, intentions (and time).

Most of it can be translated to classical logic;
- ... but it looks horribly UGLY there;
- ... and in most cases it’s not automatizable any more.

Remember Frege’s notation of predicate logic:

Good to know:
- MSPASS is a theorem prover implementing many modal logics,
- also description logics, relational calculus, etc
- built upon SPASS, a resolution prover for first-order logic with equality,
- check out http://www.cs.man.ac.uk/~schmidt/mspass/.
Example 2.1 (Muddy children – Shoham’s version)

A group of \( n \) children enters the house after having played in the mud outside. They are greeted by their father, who notices that \( k \) of them have mud on their foreheads (no kid can see whether she herself has a muddy forehead, but they can see all other foreheads).

Can the kids determine by pure thinking whether they have a muddy forehead?

The father announces: At least one of you has mud on her forehead.

He also says If you know (can prove) that your forehead is muddy, then raise your hands now.

Nothing happens. The father keeps repeating the question.

After exactly \( k \) rounds, all the children with muddy foreheads raise their hands.

How is that possible? The announcement of the father does not reveal anything, or does-it?

Definition 2.2 (Partition Model)

An \( n \)-agent partition model over a language \( \mathcal{L} \) is a tuple \((\mathcal{W}, I_1, \ldots, I_n)\), where

- \( \mathcal{W} \): is a set of possible worlds, i.e. a set of \( \mathcal{L} \)-structures (each \( \mathcal{L} \) sentence \( \varphi \) is either true in each of \( w \in \mathcal{W} \) or not).
- \( I_i \): each \( I_i \) is a partition of \( \mathcal{W} \):
  \[ I_i = \{W_{i1}, W_{i2}, \ldots, W_{ir}\} \] with \( W_{ij} \cap W_{ik} = \emptyset \) for \( j \neq k \) and \( \bigcup_{1 \leq j \leq r} W_{ij} = \mathcal{W} \).

We also use the notation \((w \in \mathcal{W})\)

\[ I_i(w) := \{w' : w \in W_{ij} \text{ and } w' \in W_{ij}\} \]

all worlds in the partition containing world \( w \), according to agent \( i \)'s view of the universe.

The worlds in \( \mathcal{W} \) can be propositional valuations or even first-order structures. In this lecture, we will mainly consider the propositional version.

The language \( \mathcal{L} \) in our example consists just of the propositional constants \( \text{mud}_1, \ldots, \text{mud}_n \).

How can we formalise the notion of one agent knowing something? And reason about what agents know and draw conclusions?
We introduce new operators $K_i$ with the intuitive meaning $K_i \phi$ “agent $i$ knows that $\phi$ holds”. We can now formulate statements in a language $L(K_1, \ldots, K_n)$ containing these operators. It remains to define a precise semantics for this notion.

Definition 2.3 (Semantics for partition models)
Let an $n$-agent partition model $\mathcal{A} = (W, I_1, \ldots, I_n)$ over a language $L$ be given. Then we define the relation $|=\text{ for } L(K_1, \ldots, K_n)$ formulae as follows:

- for $\phi \in L$: $\mathcal{A}, w |= \phi$ if and only if $w |= \phi$,
- $\mathcal{A}, w |= K_i \phi$ if and only if for all worlds $w'$, if $w' \in I_i(w)$ then $\mathcal{A}, w' |= \phi$.

Muddy children revisited (blackboard).
2. Modal Logic

A slight generalisation of partition models leads to modal logic.

How can one look at a partition? It is like a set of equivalence classes: for agent \(i\), all worlds in one partition are equivalent. They are all possible scenarios.

So let’s generalise and introduce a binary relation \(R\) on all worlds: \(w_1Rw_2\) meaning that world \(w_2\) can be accessed (is reachable) from world \(w_1\).

Definition 2.4 (Modal Logic with \(n\) modalities)

The language \(L_{\text{modal}_n}\) of modal logic with \(n\) modal operators \(\Box_1, \ldots, \Box_n\) is the smallest set containing the propositional constants of \(L\), and with formulae \(\varphi, \psi\) also the formulae \(\Box_i\varphi, \neg\varphi, \varphi \land \psi\). We treat \(\lor, \to, \leftrightarrow, \lozenge\) as macros (defined as usual).

Note that the \(\Box\) operators can be nested:

\[(\Box_1 \Box_2 \Box_1 p) \lor \Box_3 \neg p\]

Definition 2.5 (Semantics of Modal Logic)

The truth of a \(L_{\text{modal}_n}\)-formula is evaluated relative to a world \(w\) and a Kripke structure \(\langle W, R\rangle\), where \(W\) is a set of (propositional) models (possible worlds) and \(R\) is a binary relation on them, the accessibility relation.

The relation \(\models\) is defined as follows:

- \(\langle W, R\rangle \models_w p\) if \(p\) is primitive and \(w \models p\),
- \(\langle W, R\rangle \models_w \varphi \land \psi\) if \(\langle W, R\rangle \models_w \varphi\) and \(\langle W, R\rangle \models_w \psi\),
- \(\langle W, R\rangle \models_w \neg \varphi\) if not \(\langle W, R\rangle \models_w \varphi\),
- \(\langle W, R\rangle \models_w \Box \varphi\) if for any \(w' \in W\) with \(wRw'\): \(\langle W, R\rangle \models_{w'} \varphi\)
2. Modal Logic

2.3 Axioms for Modal Logics

As in classical logic, one can ask about a complete axiom system. Is there a calculus that allows to derive all sentences true in all Kripke models?

**Definition 2.6 (System K)**

The system $K$ is an extension of the propositional calculus by the axiom

$$\text{Axiom K} \quad (\Box \phi \land \Box (\phi \to \psi)) \to \Box \psi$$

and the inference rule

$$\text{(Necessitation)} \quad \phi \quad \Box \phi.$$

**Theorem 2.7 (Soundness/completeness of System K)**

System $K$ is sound and complete with respect to arbitrary Kripke models.

If we allow $n$ modalities, the theorem as well as the definitions extend in an obvious way. The calculus is then called System $K_n$ to account for the $n$ modalities.

- Note that we have not assumed any properties of the accessibility relation $R$: it is just any binary relation.
- How does our partition model fit in this framework?
- A partition is nothing else than an equivalence relation!
- Assuming that $R$ is an equivalence relation, what additional statements (axioms) are true in all Kripke models?
To which axioms do the following properties lead?

**Reflexivity:** \( xRx \) for all \( x \).

**Transitivity:** \( xRy \) and \( yRz \) implies \( xRz \) for all \( x, y, z \).

**Euclidean:** \( xRy \) and \( xRz \) implies \( yRz \) for all for all \( x, y, z \).

**Serial:** for all \( x \) there is a \( x' \) such that \( xRx' \).

**Lemma 2.8**

A binary relation is an equivalence relation if and only if it is reflexive, transitive and euclidean.

**Definition 2.9 (Extending K by Axioms D, T, 4, 5)**

The system \( K \) is often extended by (a subset of) the following axioms:

- \( K \) \((K_i \varphi \land K_i (\varphi \rightarrow \psi)) \rightarrow K_i \psi\)
- \( D \) \(\neg K_i (\varphi \land \neg \varphi)\)
- \( T \) \(K_i \varphi \rightarrow \varphi\)
- \( 4 \) \(K_i \varphi \rightarrow K_i K_i \varphi\)
- \( 5 \) \(\neg K_i \varphi \rightarrow K_i \neg K_i \varphi\)

called as above by historical reasons. The system consisting of \( KT45 \) is also called \( S5 \).

**Theorem 2.10 (Sound/complete Subsystems of KDT45)**

Let \( X \) be any subset of \( \{D, T, 4, 5\} \) and let \( X \) be any subset of \( \{\text{serial, reflexive, transitive, euclidean}\} \) corresponding to \( X \).

Then \( K \cup X \) is sound and complete with respect to Kripke models the accessibility relation of which satisfies \( X \).

**Corollary 2.11 (Sound-, completeness of KT45 (S5))**

System \( KT45 \) is sound and complete with respect to Kripke models the accessibility relation of which is reflexive, transitive and euclidean.
Exercise: Show that

1. The axiom D follows from KT45.
2. Show that KD45 is not equivalent to K45: axiom D does not follow from K45.

Up to now we were thinking of $K_i \phi$ (the box operator $\square_i$) as agent i knows that $\phi$. What if we interpret the operator as belief?

Under such an interpretation axiom T has to be dropped. But all other axioms make sense.

Axiom system KD45 is called the standard logic of beliefs. Axiom K is called logical omniscience, axiom D is called consistency, axiom 4 (resp. axiom 5) is called positive (resp. negative) introspection.

Deontic Logic is concerned with normative law and normative reasoning in law. The operators used denote obligation, permission, prohibition etc.

We describe two systems: system OS (Von Wright, 1951) and KD (Åqvist, 1984).
Definition 2.12 (System OS)

The system OS is based on two modalities (deontic operators): O, meaning obligation, and P meaning permission. It is an extension of the propositional calculus by the axioms

\[
\begin{align*}
OS1 & : Op \leftrightarrow \neg P \neg p \\
OS2 & : pp \lor P \neg p \\
OS3 & : P(p \lor q) \leftrightarrow Pp \lor Pq
\end{align*}
\]

and the inference rule

\[
(O4) \quad \frac{\phi \leftrightarrow \psi}{P\phi \leftrightarrow P\psi}
\]

It can be shown that this system can be seen as a modal logic with O as primitive modality. But then, because of necessitation, one has to accept the derivable sentence

\[
(O\top) \quad O(p \lor \neg p)
\]

(stating the existence of an empty normative system): von Wright originally rejected this as an axiom.

The KD System

The KD System is a von Wright-type system including the F (forbidden) operator and consisting of the following additional axioms and rules (again the classical axioms and modus ponens are included).

\[
\begin{align*}
(KD1) & : O(p \rightarrow q) \rightarrow (O p \rightarrow O q) \\
(KD2) & : O p \rightarrow P p \\
(KD3) & : P p \equiv \neg O \neg p \\
(KD4) & : F p \equiv \neg P p \\
(KD5) & : O\text{-necessitation: } \frac{p}{O p}
\end{align*}
\]

Axiom (KD1) is the so called K-axiom; (KD2) is the D-axiom, stating that obligatory implies permitted; (KD3) states that permission is the dual of obligation and (KD4) says that forbidden is not permitted. (KD1) holds for any modal necessity operator.

O is the basic operator (all others are definable with it). The axioms reflect the idea that something is obligated if it holds in all perfect (ideal) worlds (relative to the world where one is).
2.5 References


2. Modal Logic

5. References

Logic-Based Specification Environments for Multi-Agent Systems.
Ph. D. thesis, Computer Science Department of Genova University, Genova, Italy.
DISI-TH-2002-04.

Logic-based specification languages for intelligent software agents.

Extending UML for agents.
In G. Wagner, Y. Lesperance, and E. Yu (Eds.), Proceedings of the 2nd Workshop on Agent-Oriented Information Systems (AOIS’00).
 Held in conjunction with 17th National conference on Artificial Intelligence (AAAI’00), pp. 3–17.

Representing agent interaction protocols in UML.
In P. Ciancarini and M. Wooldridge (Eds.), Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE’00), pp. 121–140. Springer-Verlag.
LNCS 1957.

Deontic logic.
Mind 60, 1–15.
3. Action and Time

**Up to now:**
- Several operators $K_i$, each defines an accessibility relation on worlds.
- Description of static systems: no possibility of change

**But:**
- MAS are dynamic!

3. Action and Time

**1. Dynamic Logic**

3.1 Dynamic Logic

**In this chapter:**
- We add program/action modalities $[\alpha], \langle \alpha \rangle$, and define dynamic logic, where we can talk about actions and their outcomes.
- We discuss two basic kinds of temporal logic, i.e. linear and branching time temporal logic.
- We end with introducing ATL, in which temporal logic is combined with elements of game theory.

**1st. Idea:** Consider actions or programs $\alpha$. Each such $\alpha$ defines a transition (accessibility relation) from worlds into worlds.

**2nd. Idea:** We need statements about the outcome of actions:
- $[\alpha] \phi$: “after every execution of $\alpha$, $\phi$ holds;
- $\langle \alpha \rangle \phi$: “after some executions of $\alpha$, $\phi$ holds.

Naturally, $\langle \alpha \rangle \phi \equiv \neg [\alpha] \neg \phi$. 

### 3. Action and Time

#### 1. Dynamic Logic

**3rd. Idea:** Programs/actions can be combined (sequentially, nondeterministically, iteratively):

\[
[\alpha; \beta]\varphi \rightarrow [\gamma]\varphi' 
\]

meaning “if after **execution of** \(\alpha\) and **then** \(\beta\) the formula \(\varphi\) holds, then after **executing** \(\gamma\) the formula \(\varphi'\) holds”.

---

**Definition 3.1 (Labelled Transition System)**

A labelled transition system is a pair

\[
\langle Q, \{\frac{\alpha}{\rightarrow}: \alpha \in L}\rangle 
\]

where \(Q\) is a non-empty set of states and \(L\) is a non-empty set of labels and for each \(\alpha \in L\):

\[\frac{\alpha}{\rightarrow} \subseteq Q \times Q.\]

**Definition 3.2 (Dynamic Logic: Models)**

A model of propositional dynamic logic is given by a labelled transition systems and an evaluation of propositions.

---

**Definition 3.3 (Semantics of DL)**

\[s \models [\alpha]\varphi\iff \text{for every } t \text{ such that } s \xrightarrow{\alpha} t, \text{ we have } t \models \varphi.\]
3. Action and Time 1. Dynamic Logic

Definition 3.4 (Composite labels)

We assume that the set of labels forms a Kleene algebra \( \langle L, ;, \cup, * \rangle \). In addition, we assume that the set of labels contains constructs of the form “\( \varphi? \)”, whenever \( \varphi \) is a formula not involving any modalities. Such labels are called composite labels.

What has this to do with programs?

If \( \varphi \) then \( a \) else \( b \) \hspace{1em} (\( \varphi?;a \) \cup (\neg \varphi?;b) \)

While \( \varphi \) do \( a \) \hspace{1em} (\( \varphi?;a \)\(^*\); \neg \varphi?)
### Reasoning about Time: CTL

**Ideas:**
- The accessibility relation can be seen as representing **time**.
- time: linear vs. branching

**Typical temporal operators**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigcirc \phi$</td>
<td>$\phi$ is true in the <em>next</em> moment in time</td>
</tr>
<tr>
<td>$\square \phi$</td>
<td>$\phi$ is true in <em>all</em> future moments</td>
</tr>
<tr>
<td>$\Diamond \phi$</td>
<td>$\phi$ is true in <em>some</em> future moment</td>
</tr>
<tr>
<td>$\phi U \psi$</td>
<td>$\phi$ is true <em>until</em> the moment when $\psi$ becomes true</td>
</tr>
</tbody>
</table>

- $(\neg \text{passport} \lor \neg \text{ticket}) \rightarrow \bigcirc \neg \text{board_flight}$
- $\text{send}(msg, rcvr) \rightarrow \Diamond \text{receive}(msg, rcvr)$
Temporal logic was originally developed in order to represent tense in natural language.

Within Computer Science, it has achieved a significant role in the formal specification and verification of concurrent and distributed systems. Much of this popularity has been achieved because a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.

- safety properties
- liveness properties
- fairness properties

### 3. Action and Time 2. Temporal Logic

**Safety / maintenance:**

“something bad will not happen”

“something good will always hold”

Typical examples:

- □¬bankrupt
- □(fuelOK ∨ ◊fuelOK)

and so on...

**Usually:** □¬....

### 3. Action and Time 2. Temporal Logic

**Liveness:**

“something good will happen”

Typical examples:

- ◊rich
- roL → ◊roP

and so on...

**Usually:** ◊....

### 3. Action and Time 2. Temporal Logic

**Combinations of safety and liveness possible:**

- ◊□roP
- □(roL → ◊roP) \(\rightsquigarrow\) fairness
Strong fairness

"if something is attempted/requested, then it will be successful/allocated"

Typical examples:

\[ \Box (\text{attempt} \rightarrow \Diamond \text{success}) \]
\[ \Box \Diamond \text{attempt} \rightarrow \Box \Diamond \text{success} \]

Fairness:

- Useful when scheduling processes, responding to messages, etc.
- Good for specifying properties of the environment.

Linear Time Logic: Semantics

Definition 3.5 (Models of LTL)

A model of LTL is a sequence of time moments. We call such models paths, and denote them by \( \lambda \).

Evaluation of atomic propositions at particular time moments, denoted by \( \lambda[i] \), is also needed. We also use the notation \( \lambda[i...j] \) to denote all time moments between \( i \) and \( j \) (including both). \( \lambda[i...\infty] \) denotes the denumerable infinite set of all timepoints from \( i \) on into the future.

Definition 3.6 (Semantics of LTL)

- \( \lambda \models p \) iff \( p \) is true at moment \( \lambda[0] \);
- \( \lambda \models \Box \varphi \) iff \( \lambda[1...\infty] \models \varphi \);
- \( \lambda \models \Diamond \varphi \) iff \( \lambda[i...\infty] \models \varphi \) for some \( i \geq 0 \);
- \( \lambda \models \Diamond \varphi \) iff \( \lambda[i...\infty] \models \varphi \) for all \( i \geq 0 \);
- \( \lambda \models \varphi \U \psi \) iff \( \lambda[i...\infty] \models \psi \) for some \( i \geq 0 \), and \( \lambda[j...\infty] \models \varphi \) for all \( 0 \leq j \leq i \).

Note that:

- \( \Box \varphi \equiv \neg \Diamond \neg \varphi \)
- \( \Diamond \varphi \equiv \neg \Box \neg \varphi \)
- \( \Diamond \varphi \equiv \top \U \varphi \)
3. Action and Time 2. Temporal Logic

**Branching Time: CTL**

- **CTL**: Computational Tree Logic.
- Reasoning about possible computations of a system.
- Time is branching: we want all alternative paths included!
- **Models include**: states (time points, situations), transitions (changes).
- **Paths**: courses of action, computations.

**Definition 3.7 (Transition System)**

A transition system is a pair

\[ \langle Q, \rightarrow \rangle \]

where \( Q \) is a non-empty set of states \( \rightarrow \subseteq Q \times Q \) is a transition relation.

Note that, formally, a transition relation is just a modal accessibility relation.

**Definition 3.8 (Paths)**

A path \( \lambda \) is an infinite sequence of states that can be affected by subsequent transitions.

A path must be full, i.e. either infinite, or ending in a state with no outgoing transition.

Usually, we assume that the transition relation is **serial** (time flows forever).

Then, all paths are infinite.

**Reasoning about Time: CTL**

- **Path quantifiers**: \( A \) (for all paths), \( E \) (there is a path);
- **Temporal operators**: \( \bigcirc \) (nexttime), \( \Diamond \) (sometime), \( \square \) (always) and \( \mathcal{U} \) (until);
- “Vanilla” CTL: every temporal operator must be immediately preceded by exactly one path quantifier;
- **CTL**: no syntactic restrictions;
- Reasoning in “vanilla” CTL can be automatized.
Example: Rocket and Cargo

- A rocket and a cargo,
- The rocket can be moved between London (proposition $roL$) and Paris (proposition $roP$),
- The cargo can be in London ($caL$), Paris ($caP$), or inside the rocket ($caR$),
- The rocket can be moved only if it has its fuel tank full ($fuelOK$),
- When it moves, it consumes fuel, and $nofuel$ holds after each flight.

Definition 3.9 (Semantics of CTL*: state formulae)

Let $\phi$ be a state formula. Then:

$M, q \models E\phi$ if there is a path $\lambda$, starting from $q$, such that $M, \lambda \models \phi$;

$M, q \models A\phi$ if for all paths $\lambda$, starting from $q$, we have $M, \lambda \models \phi$.

Definition 3.10 (Semantics of CTL*: path formulae)

Exactly like for LTL!

$M, \lambda \models \Diamond \phi$ if $M, \lambda[1\ldots\infty] \models \phi$;

$M, \lambda \models \phi \mathcal{U} \psi$ if $M, \lambda[i\ldots\infty] \models \phi$ for some $i \geq 0$, and $M, \lambda[j\ldots\infty] \models \psi$ for all $0 \leq j \leq i$. 

Jürgen Dix and Wojtek Jamroga
Exercise:
How to express that there is no possibility of a deadlock?

ATL: What Agents Can Achieve

- **ATL: Agent Temporal Logic** [Alur et al. 1997]
- Temporal logic meets game theory
- Main idea: cooperation modalities

\[ \langle A \rangle \phi: \text{coalition } A \text{ has a collective strategy to enforce } \phi \]
**3. Action and Time**

**3. ATL**

**ATL Models: Concurrent Game Structures**

- Agents, actions, transitions, atomic propositions
- Atomic propositions + interpretation
- Actions are abstract

**Definition 3.11 (Concurrent Game Structure)**

A concurrent game structure is a tuple $M = \langle \text{Agt}, Q, \pi, \text{Act}, d, o \rangle$, where:

- $\text{Agt}$: a finite set of all agents
- $Q$: a set of states
- $\pi$: a valuation of propositions
- $\text{Act}$: a finite set of (atomic) actions
- $d: \text{Agt} \times Q \rightarrow P(\text{Act})$ defines actions available to an agent in a state
- $o$: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

**Rocket Example: Simultaneous Actions**

**Definition 3.12 (Strategy)**

A strategy is a conditional plan.
We represent strategies with functions $s_\alpha: Q \rightarrow \text{Act}$.

Function $\text{out}(q, S_\alpha)$ returns the set of all paths that may result from agents $A$ executing strategy $S_\alpha$ from state $q$ onward.
**Definition 3.13 (Semantics of ATL)**

\[ M, q \models \langle \langle A \rangle \rangle \Phi \iff \text{there is a collective strategy } S_A \text{ such that, for every path } \lambda \in \text{out}(q, S_A), \text{ we have } M, \lambda \models \Phi. \]

**Note:**

When we combine time (actions, strategies...) with knowledge (beliefs, desires, intentions, obligations...), we finally obtain a fairly realistic model of MAS.

\[ \leadsto \] Which is exactly the topic of the next section.

**Practical importance of temporal and strategic logics:**

Automatic verification in principle possible (model checking):

- good complexity results for “vanilla” CTL and “vanilla” ATL
- model checkers exist for LTL and “vanilla” CTL.

Can be used for automated planning.

Executable specifications can be used for programming.

More: \[ \leadsto \] Chapter 5.
3. Action and Time

3.4 References


4. Combining Modalities

Chapter 4. Combining Modalities

Combining Modalities

4.1 BDI (Cohen et al.)
4.2 BDI (Rao et al.)
4.3 Beware!
4.4 References

4. Combining Modalities

We have seen how to model belief, knowledge, time and action using modal logic.

How about combining them?

Beliefs, Desires, Intentions

BDI according to Cohen and Levesque:

- Mental primitives: beliefs and goals,
- Separate operators and relations for each agent
- Time and action: LTL and DL.
- Altogether: multi-modal logic

Operator | Meaning
--- | ---
$B_i \varphi$ | agent $i$ believes $\varphi$
$G_i \varphi$ | agent $i$ has goal of $\varphi$
$\Diamond \alpha$ | action $\alpha$ will happen next
$\Box \alpha$ | action $\alpha$ has just happened

Additionally:

- Action constructors “;” and “?”, as in DL;
- Derived operators: $\Diamond \alpha$ (sometime $\alpha$), $\Box \alpha$ (always $\alpha$), $(\text{Later } \varphi)$: strict sometime, $(\text{Before } \varphi, \psi)$: $\varphi$ holds before $\psi$. 
The primitive constructs above allow us to define the following operators:

<table>
<thead>
<tr>
<th>Derived Operator</th>
<th>Definition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>♦α</td>
<td>⊮(♦x;α?)</td>
<td>sometime α</td>
</tr>
<tr>
<td>□α</td>
<td>◊¬α</td>
<td>always α</td>
</tr>
<tr>
<td>(Later ϕ)</td>
<td>¬ϕ ∧ ◊ϕ</td>
<td>strict sometime</td>
</tr>
<tr>
<td>(Before ϕ, ψ)</td>
<td>ϕ → □ψ</td>
<td>ϕ holds before ψ</td>
</tr>
</tbody>
</table>

Examples:
- \( G_{\text{citizen}} □_{safety} \)
- \( G_{\text{police}} B_{\text{citizen}} □_{safety}_{\text{citizen}} \)

All goals have to be dropped eventually:
- \( ♦\neg(G_i \ (\text{Later } ϕ)) \)

How to define a persistent goal?
- \( P-\text{Goal}_i ϕ = (G_i \ (\text{Later } ϕ)) \land B_i \neg ϕ \land \text{Before } ((B_i ϕ) \lor (B_i □\neg ϕ) \ (\neg(G_i \text{Later } ϕ))) \)

How to define intention?
- \( \text{Intend}_i ϕ = (P-\text{Goal}_i \ [\text{Done}_i (B_i (\bigcirc ϕ)) ; ϕ] \)

BDI according to Rao and Georgeff:
- Mental primitives: beliefs, desires and intentions
- Time: CTL
- Sophisticated semantic structure
Example: Card Play

\[
\begin{array}{c}
\text{Winning paths:} \\
\text{AK1, AK2, AQ1, AQ2, KQ2} \\
\text{Win conditions:} \\
\text{AK1, AK2, AQ1, AQ2, KQ2}
\end{array}
\]

Bela & E \bigcirc \text{win} & : & \text{Agent } A \text{ believes that there is a way to win in one step} \\
Desa & A \bigcirc \text{win} & : & \text{the agent desires that every path leads to a victory, so he does not have to worry about his decisions} \\
\neg \text{Bel}_a & \text{A} \bigcirc \text{win} & & \text{However, he does not believe it is possible:} \\
\text{Bel}_a & \text{Bel}_{opt} & \text{A} \bigcirc \square \text{win} & \text{On the other hand, he believes that a real optimist would believe that winning forever is inevitable:} \\
\end{array}
\]

Of course, it is possible to extend BDI:

**Horizontally:** with other modal dimensions (e.g., BOID);

**Vertically:** to a language of higher order (e.g., LORA).

**What do we use these frameworks for?**

**Analysis & Design**
- Modeling systems (the frameworks provide intuitive conceptual structures, and a systematic approach);
- Specifying desirable properties of systems.

**Verification & Exploration**
- Reasoning about concrete systems;
- Correctness testing.
4. Combining Modalities

What do we use these frameworks for?

**Automatic Generation of Behaviours**
- Programming with executable specifications;
- Automatic planning.

**Philosophy of Mind and Agency**
- Characterization of mental attitudes;
- Discussion of rational agents;
- Testing rationality assumptions.

---

4. Combining Modalities

3. Beware!

Let’s put **ATL** and **epistemic logic** in one box. **Alternating-time Temporal Epistemic Logic**

- Problems!
3. Beware!

**Problem: The Agent Knows too Much**

Agent $a$ cannot really enforce $\Diamond \text{win}$! Incomplete information, deterministic choices: if $q$ is indistinguishable from $q''$ then $a$ shouldn’t be allowed to plan one action in $q$, and another in $q''$.

**Example**

Your car runs on either unleaded or diesel petrol (you don’t know which). You are at a petrol station. **Available actions**: fill\text{unleaded}, fill\text{diesel}.

**Winning strategy**: if the car runs on unleaded, do fill\text{unleaded}; otherwise do fill\text{diesel}.

Does it make sense?

**Beware:**

Strategic and epistemic abilities are *not* independent!
4. Combining Modalities


4. References


5. Modalities in Action

Chapter 5. Modalities in Action

We suggest that model checking formulas of temporal and strategic logics can be used for verification and planning in Multi-Agent Systems.

We present several complexity results for CTL and ATL model checking.

We mention briefly two ideas of programming agents with specifications based on modal logics:

- MetaTeM (programming in temporal logics)
- Agent-0 (programming with beliefs, capabilities and commitments), the very first agent logic developed by Shoham in the early 90ies.
5. Modalities in Action
1. Verification and Planning

5.1 Verification and Planning

Model Checking Formulae of CTL and ATL

- Model checking: Does $\varphi$ hold in model $M$ and state $q$?
- Natural for verification of existing systems; also during design (“prototyping”)
- Nice results: model checking CTL and ATL is tractable!

---

Theorem (Clarke, Emerson & Sistla 1986)

CTL model checking is $P$-complete, and can be done in time linear in the size of the model and the length of the formula.

Theorem (Alur, Kupferman & Henzinger 1998)

ATL model checking is $P$-complete, and can be done in time linear in the size of the model and the length of the formula.

So, let’s model-check!

Not as easy as it seems.
Two perspectives to model checking MAS:

Verification
- Model represents the view of an objective observer
- Formula: specification to be met

Planning
- Model represents the subjective view of an agent
- Formula: goal to be achieved

Agent Planning with ATL

- Automated verification of ATL properties: Can the agents do it?
- Planning: How can we do it?
- Planning as model checking: CTL (Giunchiglia and Traverso, 1999).
- Agent planning with ATL: even more natural.

Auxiliary functions:
- Weakest precondition:
  \[ \text{pre}(A, Q_1) = \{ \langle q, \sigma_A \rangle \mid \exists \sigma \langle q, \sigma \rangle \in Q_1 \} \]
- States for which plan \( P \) is defined:
  \[ \text{states}(P) = \{ q \in Q \mid \exists \sigma \langle q, \sigma \rangle \in P \} \]
- Extending plan \( P_1 \) with \( P_2 \):
  \[ P_1 \oplus P_2 = P_1 \cup \{ \langle q, \sigma \rangle \in P_2 \mid q \notin \text{states}(P_1) \} \]
- \( P \) restricted to the states from \( Q_1 \):
  \[ P|_{Q_1} = \{ \langle q, \sigma \rangle \in P \mid q \in Q_1 \} \]
5. Modalities in Action  1. Verification and Planning

Complexity of Model Checking CTL and ATL

- Nice results: model checking CTL and ATL is tractable.
- But: the result is relative to the size of the model and the formula
- Well known catch (CTL): size of models is exponential wrt a higher-level description
- Another problem (ATL): transitions are labeled
- So: the number of transitions can be exponential in the number of agents.

3 agents/attributes, 12 states, 216 transitions
Proposition (Jamroga & Dix 2005)

ATL model checking is $\Sigma^P_2$-complete with respect to the number of states and agents.

**Complexity Class $\Sigma^P_2$**

- $\Sigma^P_1$: problems solvable in polynomial time by a non-deterministic Turing machine making adaptive queries to a $\Sigma^P_{i-1}$ oracle
- $\Sigma^P_2 = \text{NP}^\text{NP}$: problems solvable in polynomial time by a non-deterministic Turing machine making adaptive queries to an NP oracle

**Summary of Complexity Results**

<table>
<thead>
<tr>
<th></th>
<th>$m,l$</th>
<th>$n,k,l$</th>
<th>$n_{\text{local}},k,l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATL$_{ir}$</td>
<td>NP-complete [5,6]</td>
<td>$\Sigma^P_2$-complete [6]</td>
<td>?</td>
</tr>
</tbody>
</table>


More on verification (and model checking) in the afternoon course “Verification of Multiagent Systems via Model Checking” by Alessio Lomuscio and Wojciech Penczek.
Instead of propositional models, why not using first-order models?

METATEM is based on a first-order temporal logic based on discrete, linear models with finite past and infinite future, called FML.

Syntax of FML.

The formulae for FML are generated as usual, starting from a set $L_p$ of predicate symbols, a set $L_v$ of variable symbols, a set $L_c$ of constant symbols, the quantifiers $\forall$ and $\exists$, and the set $L_t$ of terms (constants and variables). The set $Fml$ is defined by:

- If $t_1,...,t_n$ are in $L_t$ and $p$ is a predicate symbol of arity $n$, then $p(t_1,...,t_n)$ is in $Fml$.
- $\top$ (true) and $\bot$ (false) are in $Fml$.
- If $A$ and $B$ are in $Fml$, then so are $\neg A$, $A \land B$, $A U B$, $A S B$, and $(A)$.
- If $A$ is in $Fml$ and $v$ is in $L_v$, then $\exists v.A$ and $\forall v.A$ are both in $Fml$.

Temporal operators:

- $\phi U \psi$ \hspace{1cm} $\phi$ holds until $\psi$ \hspace{1cm} $\text{primitive}$
- $\phi S \psi$ \hspace{1cm} $\phi$ since $\psi$ \hspace{1cm} $\text{primitive}$
- $\diamond \phi$ \hspace{1cm} $\phi$ is true in the next state \hspace{1cm} $\bot U \phi$
- $\Box \phi$ \hspace{1cm} the current state is not the initial state, and $\phi$ was true in the previous state \hspace{1cm} $\bot S \phi$
- $\lozenge \phi$ \hspace{1cm} if the current state is not the initial state, then $\phi$ was true in the previous state \hspace{1cm} $\neg \Box \neg \phi$
- $\Diamond \phi$ \hspace{1cm} $\phi$ will be true in some future state \hspace{1cm} $\top U \phi$
- $\Diamond \phi$ \hspace{1cm} $\phi$ will be true in some past state \hspace{1cm} $\top S \phi$
- $\lozenge \phi$ \hspace{1cm} $\phi$ will be true in all future states \hspace{1cm} $\neg \Diamond \neg \phi$
- $\Box \phi$ \hspace{1cm} $\phi$ will be true in all past states \hspace{1cm} $\neg \lozenge \neg \phi$
5. Modalities in Action 2. Temporal Programming

Semantics of FML.

The models for FML formulae are given by

- a structure consisting of a sequence of worlds (also called states), together with
- an assignment of truth values to atomic sentences within states,
- a domain \( D \) which is assumed to be constant for every state,
- and mappings from elements of the language into denotations.

The semantics of FML is given by the \( \models \) relation that gives the truth value of a formula in a model \( M \) at a particular moment in time \( i \) and with respect to a variable assignment.

\[
\begin{align*}
(M, i, h_v) & \models \top \quad \\
(M, i, h_v) & \models \bot \\
(M, i, h_v) & \models p(x_1, \ldots, x_n) \quad \text{iff} \quad h_v(p, i, \tau_{v_h}(x_1), \ldots, \tau_{v_h}(x_n)) = \text{true} \\
(M, i, h_v) & \models \neg \phi \quad \text{iff} \quad (M, i, h_v) \not\models \phi \\
(M, i, h_v) & \models \phi \lor \psi \quad \text{iff} \quad (M, i, h_v) \models \phi \quad \text{or} \quad (M, i, h_v) \models \psi \\
(M, i, h_v) & \models \phi \land \psi \quad \text{iff} \quad \text{for some } k \text{ s.t. } i < k, \ (M, k, h_v) \models \psi \\
&M \quad \text{and for all } j, \text{ if } i < j < k \text{ then } (M, j, h_v) \models \phi \\
(M, i, h_v) & \models \phi \land \psi \quad \text{iff} \quad \text{for some } k \text{ s.t. } 0 \leq k < i, \ (M, k, h_v) \models \psi \\
&M \quad \text{and for all } j, \text{ if } k < j < i \text{ then } (M, j, h_v) \models \phi \\
(M, i, h_v) & \models \forall x. \phi \quad \text{iff} \quad \text{for all } d \in D, \ (M, i, h_v[d/x]) \models \phi \\
(M, i, h_v) & \models \exists x. \phi \quad \text{iff} \quad \text{there exists } d \in D \text{ s.t. } (M, i, h_v[d/x]) \models \phi
\end{align*}
\]

Concurrent METATEM is a programming language for distributed AI based on FML.

- A system contains a number of concurrently executing agents which are able to communicate through message passing.
- Each agent executes a specification of its desired behavior.
- Each agent has two main components:
  - an interface which defines how the agent may interact with its environment (i.e., other agents);
  - a computational engine, defining how the agent may act.
5. Modalities in Action  2. Temporal Programming

An agent interface consists of three components:

- a unique agent identifier which names the agent
- a set of predicates defining what messages will be accepted by the agent – they are called environment predicates;
- a set of predicates defining messages that the agent may send—these are called component predicates.

Besides environment and component predicates, an agent has a set of internal predicates with no external effect.

The computational engine of an object is based on the METATEM paradigm of executable temporal logics. The idea behind this approach is to directly execute a declarative agent specification given as a set of program rules.

Program rules are temporal logic formulae of the form:

\[
\text{antecedent: past } \rightarrow \text{ consequent: future}
\]

The intuitive interpretation of such a rule is:

on the basis of the past, do the future

Contract Proposal from [MascardiMS04]

- Seller agent may receive a contractProposal message from a buyer agent.
- According to the amount of merchandise required and the price proposed by the buyer, the seller may accept the proposal, refuse it or try to negotiate a new price by sending a contractProposal message back to the buyer.
- The buyer agent can do the same (accept, refuse or negotiate) when it receives a contractProposal message back from the seller.
### Behaviour of seller

If the received message is `contractProposal(merchandise, amount, proposed price)` then
- If there is enough merchandise in the warehouse and the price is greater or equal than a `max` value, the seller accepts by sending an `accept` message to the buyer and concurrently ships the required merchandise to the buyer (if no concurrent actions are available, answering and shipping merchandise will be executed sequentially);
- If there is not enough merchandise in the warehouse or the price is lower or equal than a `min` value, the seller refuses by sending a `refuse` message to the buyer;
- If there is enough merchandise in the warehouse and the price is between `min` and `max`, the seller sends a `contractProposal` to the buyer with a proposed price evaluated as the mean of the price proposed by the buyer and `max`.

---

The merchandise to be exchanged are oranges, with minimum and maximum price 1 and 2 euro respectively. The initial amount of oranges that the seller possesses is 1000.
The internal knowledge base of the seller agent contains the following rigid predicates (predicates whose value never changes):

\[\text{min-price}(orange, 1).\]
\[\text{max-price}(orange, 2).\]

The internal knowledge base of the seller agent contains the following flexible predicates (predicates whose value changes over time):

\[\text{storing}(orange, 1000).\]

If there was a previous state where Buyer sent a contractProposal message to seller, and in that previous state all the conditions were met to accept the proposal, then accept the Buyer's proposal and ship the required merchandise.

\[\forall \text{Buyer, Merchandise, Req.Amnt, Price}\]
\[\odot \left( \text{contractProposal}(\text{Buyer, seller, Merchandise, Req.Amnt, Price}) \land \text{storing}(\text{Merchandise, Old_Amount}) \land \text{min} - \text{price}(\text{Merchandise, Min}) \land \text{Old_Amount} < \text{Req.Amnt} \land \text{Price} \geq \text{Min} \right) \implies \text{accept(seller, Buyer, Merchandise, Req.Amnt, Price)}\]

If there was a previous state where Buyer sent a contractProposal message to seller, and in that previous state the conditions were not met to accept the Buyer's proposal, then send a refuse message to Buyer.

\[\forall \text{Buyer, Merchandise, Req.Amnt, Price}\]
\[\odot \left( \text{contractProposal}(\text{Buyer, seller, Merchandise, Req.Amnt, Price}) \land \text{storing}(\text{Merchandise, Old_Amount}) \land \text{min} - \text{price}(\text{Merchandise, Min}) \land \text{Old_Amount} \geq \text{Req.Amnt}\land \text{Price} > \text{Min} \land \text{Price} < \text{Max} \land \text{New_Price} = (\text{Max} + \text{Price})/2 \right) \implies \text{contractProposal(seller, Buyer, Merchandise, Req.Amnt, New_Price)}\]
5. Modalities in Action 3. Agent Programming

5.3 Agent Programming

Shoham suggests the following three components of an Agent Oriented Programming system:

- A formal language with clear syntax for describing the mental state.
- A programming language for defining agents.
- A method of transforming legacy code into an agent.

Shoham: an agent has commitments (or obligations). One such commitment can be to perform an action.

- No need for action selection. But the language specification must have a mechanism for adopting commitments or not.
- All rules must be created at compile time: huge amount of possible scenarios.

AGENT-0 Programming Language

AGENT-0 is based on a quantified multi-modal logic, with direct reference to time. Three modalities: Beliefs, Commitments, Capabilities.
**Atoms:** Sentences in a point-based temporal framework:
- $is\_friend(a, b)$ (facts),
- $turn(I, left)^t$ (instantaneous actions).

No distinction between actions and facts.

**Beliefs:** $B_I^{now} go\_swimming(I, CLZ)^t$: I believe now that I am going to swim in CLZ at time $t$.
- $B_I^{now} ontalbe(block_b)^t$.
- $B_a^3 B_b^4 is\_friend(a, b)$.

**Obligations:** (also called commitments) are the beliefs that one agent will create the truth of a statement (for another agent)
- $OBL_I^{t, you} go\_swimming(I, CLZ)^{t+1}$, (at time $t$, I am obligated to you to go swimming in CLZ at time $t+1$).

The argument does not need to be an action: $OBL_I^{t, you} in\_lecture(you)^{t+1}$.

A decision is an obligation to oneself: $DEC_I^t \varphi := OBL_I^{a, a} \varphi$.

**Capabilities:** An agent is said to be capable of a statement if it has the ability to see that that statement hold at the specified time:
- $CAN_I^{now} in\_lecture(you)^t$ (I am capable of seeing to it that you are in a lecture at time $t$).

Note: capabilities may change.
AGENT-0 Programs

An agent in AGENT-0 consists of (1) a set of initial beliefs, (2) a set of capabilities, (3) a set of initial commitments, and (4) a set of commitment rules of the form

\[ \text{COMMIT} \text{msgcond} \text{mntlcond} \ (\text{agent}, t, \text{action}) \],

“Commit to perform action for agent at time t (if msgcond holds of the new incoming messages, if mntlcond holds in the mental state, if the agent is currently capable of doing action).”

Cycle

AGENT-0 follows the following simple control loop when executing a program:

1. gather incoming messages and update the mental state accordingly,
2. execute commitments (using capabilities).

More recent agent-oriented programming languages follow similar ideas!
What makes an agent do something?

5. Modalities in Action 4. References

METATEM: A framework for programming in temporal logic.
In J. W. de Bakker, W. P. de Roever, and G. Rozenberg (Eds.),
Proceedings of the 1989 REX Workshop on Stepwise Refinement of
Distributed Systems: Models, Formalisms, Correctness, pp. 94–129.
Springer-Verlag.
LNCS 430.

METATEM at work: Modelling reactive systems using executable
temporal logic.
In P. Chung, G. L. Lovegrove, and M. Ali (Eds.), Proceedings of the
6th International Conference on Industrial and Engineering
Applications of Artificial Intelligence and Expert Systems

A normal form for first-order temporal formulae.
In D. Kapur (Ed.), Proceedings of the 11th International Conference
Springer-Verlag.
LNCS 607.


5. Modalities in Action

4. References


