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

Department of Computer Science
Clausthal University of Technology, Germany

European Agent Systems Summer School
(Durham 2007)

Lecture Overview

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

Points to Other Courses

- John Lloyd: Quantified Modal Logic for Building Agents (Thursday morning)
- Alessio Lomuscio & Wojciech Penczek: Verification of Multiagent Systems via Model Checking (Thursday afternoon)

Chapter 1. Classical Logic

Classical Logic
1.1 Why Logic?
1.2 Sentential Logic (SL)
1.3 Sudoku
1.4 Calculi for SL
1.5 Wumpus in SL
1. Classical Logic

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**.

---

**Why logic at all?**

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

1. Why Logic?

**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).

2. Sentential Logic (SL)

**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 $\top$: the constants falsum and verum,
- $p, q, r, x_1, x_2, \ldots x_n, \ldots$: a countable set $AT$ of SL-constants,
- $\neg$, $\land$, $\lor$, $\rightarrow$: the sentential connectives ($\neg$ is unary, all others are binary operators),
- $()$: the parentheses to help readability.

In most cases we consider only a finite set of SL-constants. They define a language $L \subseteq L_{SL}$. The set of $L$-formulae $Fml_L$ 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 by $L$ 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 $\bar{\nu}: Fml_L \rightarrow \{\text{true}, \text{false}\}$ so that:

- $\bar{\nu}(\neg p) = \begin{cases} \text{true}, & \text{if } \bar{\nu}(p) = \text{false}, \\ \text{false}, & \text{if } \bar{\nu}(p) = \text{true}. \end{cases}$
- $\bar{\nu}(\phi \land \gamma) = \begin{cases} \text{true}, & \text{if } \bar{\nu}(\phi) = \text{true and } \bar{\nu}(\gamma) = \text{true}, \\ \text{false}, & \text{else} \end{cases}$
- $\bar{\nu}(\phi \lor \gamma) = \begin{cases} \text{true}, & \text{if } \bar{\nu}(\phi) = \text{true or } \bar{\nu}(\gamma) = \text{true}, \\ \text{false}, & \text{else} \end{cases}$
1. Classical Logic

2. Sentential Logic (SL)

Definition (continued)

\[ 
\bar{v}(\varphi \rightarrow \gamma) = \begin{cases} 
\text{true}, & \text{if } \bar{v}(\varphi) = \text{false} \text{ or } (\bar{v}(\varphi) = \text{true} \text{ and } \bar{v}(\gamma) = \text{true}) \\
\text{false}, & \text{else} 
\end{cases} 
\]

Thus each valuation \( v \) uniquely defines a \( \bar{v} \). We call \( \bar{v} \) a \( L \)-structure.

A structure determines for each formula if it is true or false. If a formula \( \varphi \) is true in structure \( \bar{v} \) we also say \( \bar{v} \models \varphi \) or simply \( v \models \varphi \). \( \bar{v} \) is a model of \( \varphi \).

A theory is a set of formulae: \( T \subseteq Fml_L \). \( v \) satisfies \( T \) if \( \bar{v}(\varphi) = \text{true} \) for all \( \varphi \subseteq T \). We write \( v \models T \).

A \( L \)-formula \( \varphi \) is called \( L \)-tautology if for all possible valuations \( v \) 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.

Semantics

The process of mapping a set of \( L \)-formulae into \{true, false\} is called semantics.

1. Classical Logic

2. Sentential Logic (SL)

Definition 1.3 (Validity of a Formula, Tautology)

1. A formula \( \varphi \in Fml_L \) holds under the valuation \( v \) if \( \bar{v}(\varphi) = \text{true} \). We also write \( \bar{v} \models \varphi \) or simply \( v \models \varphi \). \( \bar{v} \) is a model of \( \varphi \).

2. A theory is a set of formulae: \( T \subseteq Fml_L \). \( v \) 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 \( v \) 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.

1. Classical Logic

2. Sentential Logic (SL)

Definition 1.4 (Consequence Set \( Cn(T) \))

A formula \( \varphi \) follows from \( T \) if for all models \( v \) of \( T \) (i.e. \( v \models T \)) also \( v \models \varphi \) holds. We write: \( T \models \varphi \).

We call

\[ Cn_L(T) = \{ \varphi \in Fml_L : T \models \varphi \} \]

or simply \( Cn(T) \), the semantic consequence operator.

Lemma 1.5 (Properties of \( Cn(T) \))

The semantic consequence operator has the following properties:

1. \( T \)-extension: \( T \subseteq Cn(T) \),
2. Monotony: \( T \subseteq T' \Rightarrow Cn(T) \subseteq Cn(T') \),
3. Closure: \( Cn(Cn(T)) = Cn(T) \).

Lemma 1.6 (\( \varphi \notin Cn(T) \))

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

2. Sentential Logic (SL)

**Definition 1.7 (MOD(T), Cn(∪))**

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

\[
\text{MOD}(T) = \{ A : A \models T \}.
\]

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 \( Cn(\mathcal{U}) \):

\[
Cn(\mathcal{U}) = \{ \varphi \in Fml_L : \forall v \in \mathcal{U} : \bar{v}(\varphi) = \text{true} \}.
\]

\( \text{MOD} \) is obviously dual to \( Cn \):

\[
Cn(\text{MOD}(T)) = Cn(T), \quad \text{MOD}(Cn(T)) = \text{MOD}(T).
\]

3. Sudoku

**1.3 Sudoku**
Sudoku

Since some time, Sudoku puzzles are becoming quite famous.

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

Table 1: A simple Sudoku ($S_1$)

We introduce the following language $L_{\text{Sudoku}}$:

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

This completes the language, the syntax.

How many symbols are these?

Can they be solved with sentential logic?

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

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

Table 2: How to construct a model $S$?
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

\( \ldots \)

\( \text{neun}_{i,j} \) 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 \( 1 \leq i, j \leq 9 \)).

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

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

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

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

\( \text{etc.} \)

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

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

How does \( T_S \) look like?

\[
T_S = \{ \text{eins}_{1,4}, \text{eins}_{5,8}, \text{eins}_{6,6}, \\
\text{zwei}_{2,2}, \text{zwei}_{4,8}, \\
\text{drei}_{6,8}, \text{drei}_{8,3}, \text{drei}_{9,4}, \\
\text{vier}_{1,7}, \text{vier}_{2,5}, \text{vier}_{3,1}, \text{vier}_{4,3}, \text{vier}_{8,2}, \text{vier}_{9,8}, \\
\ldots \\
\text{neun}_{3,4}, \text{neun}_{5,2}, \text{neun}_{6,9}, \}
\]
The formulae on the last slide are saying, that
1. The number 1 must appear somewhere in the first square.
2. The number 2 must appear somewhere in the first square.
3. The number 3 must appear somewhere in the first square.
4. etc

Does that mean, that each number 1, ..., 9 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: \(T'_2\)
1. \(\text{eins}_{1,4} \lor \ldots \lor \text{eins}_{3,6} \)
2. \(\text{zwei}_{1,4} \lor \ldots \lor \text{zwei}_{3,6} \)
3. \(\text{drei}_{1,4} \lor \ldots \lor \text{drei}_{3,6} \)
4. \(\text{vier}_{1,4} \lor \ldots \lor \text{vier}_{3,6} \)
5. \(\text{fuenf}_{1,4} \lor \ldots \lor \text{fuenf}_{3,6} \)
6. \(\text{sechs}_{1,4} \lor \ldots \lor \text{sechs}_{3,6} \)
7. \(\text{sieben}_{1,4} \lor \ldots \lor \text{sieben}_{3,6} \)
8. \(\text{acht}_{1,4} \lor \ldots \lor \text{acht}_{3,6} \)
9. \(\text{neun}_{1,4} \lor \ldots \lor \text{neun}_{3,6} \)

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

Analogously for all other columns.

First Row:

- eins, v eins, v v eins, v eins, v eins, v eins, v eins, v eins
- zweii, v zweii, v zweii, v zweii, v zweii, v zweii, v zweii
- drei, v drei, v drei, v drei, v drei, v drei, v drei
- vieren, v vier, v vier, v vier, v vier, v vier, v vier
- sechse, v sechs, v sechs, v sechs, v sechs, v sechs, v sechs
- sieben, v sieben, v sieben, v sieben, v sieben, v sieben, v sieben
- acht, v acht, v acht, v acht, v acht, v acht, v acht
- neun, v neun, v neun, v neun, v neun, v neun, v neun

First Column:

- eins, v eins, v eins, v eins, v eins, v eins, v eins
- zweii, v zweii, v zweii, v zweii, v zweii, v zweii, v zweii
- drei, v drei, v drei, v drei, v drei, v drei, v drei
- vieren, v vier, v vier, v vier, v vier, v vier, v vier
- sechse, v sechs, v sechs, v sechs, v sechs, v sechs, v sechs
- sieben, v sieben, v sieben, v sieben, v sieben, v sieben, v sieben
- acht, v acht, v acht, v acht, v acht, v acht, v acht
- neun, v neun, v neun, v neun, v neun, v neun, v neun

Ninth Row:

- eins, v eins, v eins, v eins, v eins, v eins, v eins
- zweii, v zweii, v zweii, v zweii, v zweii, v zweii, v zweii
- drei, v drei, v drei, v drei, v drei, v drei, v drei
- vieren, v vier, v vier, v vier, v vier, v vier, v vier
- sechse, v sechs, v sechs, v sechs, v sechs, v sechs, v sechs
- sieben, v sieben, v sieben, v sieben, v sieben, v sieben, v sieben
- acht, v acht, v acht, v acht, v acht, v acht, v acht
- neun, v neun, v neun, v neun, v neun, v neun, v neun

Is that sufficient? What if a column contains several 1s?

Analogously for all other columns.

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

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

What is still missing:

- Columns: several numbers from 1 to 9
- Rows: several numbers from 1 to 9

As an example, consider a 3x3 Sudoku grid with the following filled cells:

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

The next number to fill in could be 7, 8, or 9.
All put together:

\[ T_{\text{Sudoku}} = T_1 \cup T'_1 \cup \ldots \cup T_9 \cup T'_9 \]
\[ T_{\text{Row}} \cup \ldots \cup T_{\text{Row}9} \]
\[ T_{\text{Column}} \cup \ldots \cup T_{\text{Column}9} \]

1. Classical Logic

The above formulation is strictly formulated in propositional logic.

Theorem provers, even if they consider only propositional theories, often use predicates, variables etc.

smodels uses a predicate logic formulation, including variables. But as there are no function symbols, such an input can be seen as a compact representation.

It allows to use a few rules as a shorthand for thousands of rules using propositional constants.

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).

Here is a more difficult one.

Table 3: A difficult Sudoku \( S_{\text{difficult}} \)
1. Classical Logic

Statements in *smodels* are written as

head :- body

This corresponds to an implication

\( body \rightarrow head \).

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

For example the theory

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

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.

2. More constructs in *smodels*

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

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

Sudoku formalization

\[
\begin{align*}
\text{row}(0..8). \\
\text{col}(0..8). \\
\text{val}(1..9).
\end{align*}
\]

\[
\begin{align*}
1 \{ p(X,Y,A) : \text{val}(A) \} 1 :- \text{row}(X), \text{col}(Y). \\
1 \{ p(X,Y,A) : \text{row}(X) \} 1 :- \text{val}(A), \text{col}(Y). \\
1 \{ p(X,Y,A) : \text{col}(Y) \} 1 :- \text{row}(X), \text{val}(A). \\
1 \{ p(X,Y,A) : \text{row}(X) : \text{col}(Y) \\
\quad : \text{eq}(\text{div}(X,3), \text{div}(R,3)) : \text{eq}(\text{div}(Y,3), \text{div}(C,3)) \} 1 \\
\quad :- \text{row}(R), \text{col}(C), \text{val}(A).
\end{align*}
\]
A general notion of a certain sort of calculi.

Definition 1.11 (Hilbert-Type Calculi)

A Hilbert-Type calculus over a language $\mathcal{L}$ is a pair $\langle \text{Ax}, \text{Inf} \rangle$ where

- **Ax**: is a subset of $\text{Fml}_\mathcal{L}$, the set of well-formed formulae in $\mathcal{L}$: they are called axioms,
- **Inf**: is a set of pairs written in the form $\phi_1, \phi_2, \ldots, \phi_n, \psi$ where $\phi_1, \phi_2, \ldots, \phi_n, \psi$ are $\mathcal{L}$-formulae: they are called inference rules.

Intuitively, one can assume all axioms as “true formulae” (tautologies) and then use the inference rules to derive even more new formulae.

We now define a particular instance of our general notion.

Definition 1.12 (Calculus for Sentential Logic SL)

We define $\text{Hilbert}^{\text{SL}} = \langle \text{Ax}^{\text{SL}} \mathcal{L}, \{\text{MP}\} \rangle$, the Hilbert-Type calculus, as follows. The underlying language is $\mathcal{L} \subseteq \mathcal{L}_{\text{SL}}$ with the wellformed formulae $\text{Fml}_\mathcal{L}$ as defined in Definition 1.1.

Axioms in SL ($\text{Ax}^{\text{SL}} \mathcal{L}$) are the following formulae:

1. $\phi \rightarrow \top, \Box \rightarrow \phi, \neg \top \rightarrow \Box, \Box \rightarrow \neg \top$,
2. $(\phi \land \psi) \rightarrow \phi, (\phi \land \psi) \rightarrow \psi$,
3. $\phi \rightarrow (\phi \lor \psi), \psi \rightarrow (\phi \lor \psi)$,
4. $\neg \phi \rightarrow \phi, (\phi \rightarrow \psi) \rightarrow ((\phi \rightarrow \neg \psi) \rightarrow \neg \phi)$,
5. $\phi \rightarrow (\psi \rightarrow \phi), \phi \rightarrow (\psi \rightarrow (\phi \land \psi))$.

$\phi, \psi$ stand for arbitrarily complex formulae (not just constants). They represent schemata, rather than formulae in the language.

We now define a particular instance of our general notion.

Definition 1.13 (Proof)

A proof of a formula $\phi$ from a theory $T \subseteq \text{Fml}_\mathcal{L}$ is a sequence $\varphi_1, \ldots, \varphi_n$ of formulae such that $\varphi_n = \phi$ 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: $T \vdash \phi$ ($\phi$ can be derived from $T$).
We have now introduced two important notions:

**Syntactic derivability** $\vdash$: the notion that certain formulae can be derived from other formulae using a certain calculus,

**Semantic validity** $\models$: the notion that certain formulae follow from other formulae based on the semantic notion of a model.

**Definition 1.14 (Correct-, Completeness for a calculus)**

Given an arbitrary **calculus** (which defines a notion $\vdash$) and a **semantics** based on certain models (which defines a relation $\models$), we say that

**Correctness**: The calculus is **correct** with respect to the semantics, if the following holds:

$$ \Phi \vdash \phi \text{ implies } \Phi \models \phi. $$

**Completeness**: The calculus is **complete** with respect to the semantics, if the following holds:

$$ \Phi \models \phi \text{ implies } \Phi \vdash \phi. $$

**Theorem 1.15 (Correct-, Completeness for Hilbert $\mathcal{SL}$)**

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

$$ T \models \phi \text{ if and only if } T \vdash \phi $$

**Theorem 1.16 (Compactness for Hilbert $\mathcal{SL}$)**

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

$$ \text{Cn}(T) = \bigcup \{ \text{Cn}(T') : T' \subseteq T, \text{T' finite} \}. $$
It is well-known, that any formula $\phi$ can be written as a conjunction of disjunctions $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \phi_{i,j}$.

The $\phi_{i,j}$ are just constants or negated constants. The $n$ disjunctions $\bigvee_{j=1}^{m_i} \phi_{i,j}$ are called clauses of $\phi$.

**Normalform**

Instead of working on arbitrary formulae, it is often easier to work on finite sets of clauses.

**Set-notation of clauses**

A disjunction $A \lor \neg B \lor C \lor \ldots \lor \neg E$ is often written as a set $\{A, \neg B, C, \ldots, \neg E\}$.

Thus the set-theoretic union of such sets corresponds again to a clause: $\{A, \neg B\} \cup \{A, \neg C\}$ represents $A \lor \neg B \lor \neg C$. Note that the empty set $\emptyset$ is identified with $\Box$.

**A resolution calculus for SL**

The resolution calculus is defined over the language $L_{res} \subseteq L_{SL}$ where the set of well-formed formulae $Fml_{L_{res}}$ consists of all disjunctions of the following form $A \lor \neg B \lor C \lor \ldots \lor \neg E$, i.e. the disjuncts are only constants or their negations. No implications or conjunctions are allowed. These formulae are called clauses. $\Box$ is also a clause: the empty disjunction.

**We define the following inference rule on $Fml_{L_{res}}$:**

**Definition 1.17 (SL resolution)**

Let $C_1, C_2$ be clauses (disjunctions). Deduce the clause $C_1 \lor C_2$ from $C_1 \lor A$ and $C_2 \lor \neg A$:

\[
\frac{C_1 \lor A, \ C_2 \lor \neg A}{\ C_1 \lor C_2}
\]

If $C_1 = C_2 = \emptyset$, then $C_1 \lor C_2 = \Box$. 

Jürgen Dix and Wojtek Jamroga

European Agent Systems Summer School (Durham 2007)
If we use the set-notation for clauses, we can formulate the inference rule as follows:

**Definition 1.18 (SL resolution (Set notation))**

Deduce the clause \( C_1 \cup C_2 \) from \( C_1 \cup \{A\} \) and \( C_2 \cup \{\neg A\} \):

\[
\frac{C_1 \cup \{A\},\ C_2 \cup \{\neg A\}}{C_1 \cup C_2}
\]

(Res)

Again, we identify the empty set \( \emptyset \) with \( \Box \).

**Definition 1.19 (Resolution Calculus for SL)**

We define the resolution calculus \( \text{Robinson}^{\text{SL}}_{\text{res}} = \langle \emptyset, \{\text{Res}\} \rangle \) as follows. The underlying language is \( L_{\text{res}} \subseteq L_{\text{SL}} \) defined on Slide 57 together with the well-formed formulae \( Fml_{\text{res}} \).

Thus there are no axioms and only one inference rule. The well-formed formulae are just clauses.

**Question:**

Is this calculus correct and complete?

**Answer:**

It is correct, but it is not complete!

But every problem of the kind “\( T \models \phi \)” is equivalent to “\( T \cup \{\neg \phi\} \) is unsatisfiable” or rather to

\( T \cup \{\neg \phi\} \models \Box \)

(\( \models \) stands for the calculus introduced above).

**Theorem 1.20 (Completeness of Resolution Refutation)**

If \( M \) is an unsatisfiable set of clauses then the empty clause \( \Box \) can be derived in \( \text{Robinson}^{\text{SL}}_{\text{res}} \).

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>START</td>
<td>PIT</td>
<td></td>
<td>Breeze</td>
</tr>
<tr>
<td>2</td>
<td>Stench</td>
<td>PIT</td>
<td></td>
<td>Breeze</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>PIT</td>
<td></td>
<td>Breeze</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>PIT</td>
</tr>
</tbody>
</table>

### 5. Wumpus in SL

**Language definition:**
- $S_{i,j}$: stench
- $B_{i,j}$: breeze
- $Pit_{i,j}$: is a pit
- $Gl_{i,j}$: glitters
- $W_{i,j}$: 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.

---

Chapter 2. Modal Logic

**Modal Logic**

- 2.1 Reasoning about Knowledge
- 2.2 Kripke Semantics
- 2.3 Axioms for Modal Logics
- 2.4 Normative Reasoning
- 2.5 References

---

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**.
2. Modal Logic

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

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:

```
\begin{align*}
A &< b \\
\forall \beta (0 + \Gamma = b) &< B \\
b &< B \\
\forall \beta (0 + \Gamma = b) &< B \\
b &< B
\end{align*}
```

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).

2. Modal 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/.
2. Modal Logic

1. Reasoning about Knowledge

2.1 Reasoning about Knowledge

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 find out 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 new, or does-it?
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 \( \mathcal{L}(K_1, \ldots, K_n) \) containing these operators. It remains to define a precise semantics for this notion.
2. Modal Logic

1. Reasoning about Knowledge

### Definition 2.3 (Semantics for partition models)

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

- for $\varphi \in L$: $A, w \models \varphi$ if and only if $w \models \varphi$.
- $A, w \models K_i \varphi$ if and only if for all worlds $w'$, if $w' \in I_i(w)$ then $A, w' \models \varphi$.

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: $w_1 R w_2$ meaning that world $w_2$ can be accessed (is reachable) from world $w_1$.
2. Modal Logic

2. Kripke Semantics

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 \( \lor, \rightarrow, \leftrightarrow, \diamond \) as macros (defined as usual).

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

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

2. Modal Logic

3. Axioms for Modal Logics

2. Kripke Semantics

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

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

\[
\text{Axiom K } (\Box \varphi \land \Box (\varphi \rightarrow \psi)) \rightarrow \Box \psi
\]

and the inference rule

\[
\text{(Necessitation) } \varphi \rightarrow \Box \varphi.
\]
2. Modal Logic

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:

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

called as above for 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 \{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 \( K_i\varphi \) (the box operator \( \square_i \)) as agent \( i \) knows that \( \varphi \). 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.
2. Modal Logic

4. Normative Reasoning

2.4 Normative Reasoning

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 & : O p \leftrightarrow \neg P \neg p \\
OS2 & : \neg P p \lor P \neg p \\
OS3 & : P (p \lor q) \leftrightarrow P p \lor P q
\end{align*}

and the inference rule

\begin{align*}
(OS4) & : \phi \leftrightarrow \psi \\
& : P \phi \leftrightarrow P \psi.
\end{align*}

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

\((OT) \quad O (p \lor \neg p)\)

(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).

\[(KD1) \neg (p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)\]
\[(KD2) \neg p \rightarrow \neg q\]
\[(KD3) q \equiv \neg \neg q\]
\[(KD4) \neg p \equiv \neg q\]
\[(KD5) \neg \neg q\]

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 obliged if it holds in all perfect (ideal) worlds (relative to the world where one is).

2.5 References


http://www.fipa.org/.
2. Modal Logic

References


2. Modal Logic

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!

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.
3. Action and Time
1. Dynamic Logic

3.1 Dynamic Logic

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.

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

3rd idea: Programs/actions can be combined (sequentially, nondeterministically, iteratively), e.g.:

$[\alpha; \beta]\phi$

would mean “after every execution of $\alpha$ and then $\beta$, formula $\phi$ holds”.

Definition 3.1 (Labelled Transition System)

A labelled transition system is a pair

$\langle Q, \{\alpha \rightarrow: \alpha \in L}\rangle$

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

$\rightarrow \subseteq Q \times Q$.

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)**

\[ M,s \models [\alpha]\varphi \text{ iff for every } t \text{ such that } s \xrightarrow{\alpha} t, \text{ we have } M,t \models \varphi. \]

---

**Definition 3.4 (Composite labels)**

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

- “;” means sequential composition,
- “\( \cup \)” means nondeterministic choice,
- “*” means finite iteration (regular expr.),
- “\( \varphi? \)” means test.

Thus we assume that the labels obey the following conditions:

- \( s \xrightarrow{\alpha;\beta} t \) iff \( s \xrightarrow{\alpha} s' \) and \( s' \xrightarrow{\beta} t \),
- \( s \xrightarrow{\alpha \cup \beta} t \) iff \( s \xrightarrow{\alpha} t \) or \( s \xrightarrow{\beta} t \),
- \( s \xrightarrow{\alpha^*} t \) is the reflexive and transitive closure of \( s \xrightarrow{\alpha} t \),
- \( s \xrightarrow{\varphi?} t \) iff \( s \models \varphi \).
What has this to do with programs?

\[
\begin{align*}
\text{if } \varphi & \text{ then } a \text{ else } b & (\varphi?;a) \cup (\neg\varphi?;b) \\
\text{while } \varphi & \text{ do } a & (\varphi?;a)^*; (\neg\varphi)
\end{align*}
\]
3.2 Temporal Logic

**Typical temporal operators**

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

- $\Box ((\neg passport \lor \neg ticket) \rightarrow \Diamond \neg board_flight)$
- $send(msg, rcvr) \rightarrow \Diamond receive(msg, rcvr)$

**Reasoning about Time: CTL**

**Ideas:**

- The accessibility relation represents *time*.
- Time: linear vs. branching

**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:

\[ \square \neg \text{bankrupt} \]
\[ \square (\text{fuelOK} \lor \square \text{fuelOK}) \]
and so on . . .

Usually: \[ \square \neg \ldots \]

3. Action and Time

2. Temporal Logic

Liveness:
“something good will happen”

Typical examples:

\[ \Diamond \text{rich} \]
\[ \text{roL} \rightarrow \Diamond \text{roP} \]
and so on . . .

Usually: \[ \Diamond \ldots \]

3. Action and Time

2. Temporal Logic

Combinations of safety and liveness possible:

\[ \Diamond \square \text{roP} \]
\[ \square (\text{roL} \rightarrow \Diamond \text{roP}) \]
\[ \rightarrow \text{fairness} \]

Strong fairness

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

Typical examples:

\[ \square (\text{attempt} \rightarrow \Diamond \text{success}) \]
\[ \square \Diamond \text{attempt} \rightarrow \square \Diamond \text{success} \]
**Fairness:**
- Useful when scheduling processes, responding to messages, etc.
- Good for specifying properties of the environment.

**Linear Time: LTL**
- **LTL**: Linear Time Logic
- Reasoning about a particular computation of a system
- Time is linear: just one possible future path is included!
- **Models**: paths

**Definition 3.5 (Models of LTL)**
A model of LTL is a sequence of time moments (states). We call such models paths, and denote them by \( \lambda \).
Evaluation of atomic propositions at particular time moments is also needed.

Notation:
- \( \lambda[i] \): \( i \)th time moment
- \( \lambda[i \ldots j] \): all time moments between \( i \) and \( j \)
- \( \lambda[i \ldots \infty] \): all timepoints from \( i \) on

**Important: computational vs. behavioral structure**
- **System**
- **Computational str.**
### 3. Action and Time

#### 2. Temporal Logic

**Important: computational vs. behavioral structure**

**Computational str.**

**Behavioral str.**

LTL models are defined as behavioral structures!

**Definition 3.6 (Semantics of LTL)**

\[ \lambda \models p \iff p \text{ is true at moment } \lambda[0]; \]

\[ \lambda \models \Box \varphi \iff \lambda[i..\infty] \models \varphi; \]

\[ \lambda \models \Diamond \varphi \iff \lambda[i..\infty] \models \varphi \text{ for some } i \geq 0; \]

\[ \lambda \models \square \varphi \iff \lambda[i..\infty] \models \varphi \text{ for all } i \geq 0; \]

\[ \lambda \models \varphi \cup \psi \iff \lambda[i..\infty] \models \psi \text{ for some } i \geq 0, \text{ and } \]

\[ \lambda[j..\infty] \models \varphi \text{ for all } 0 \leq j \leq i. \]

**Note that:**

\[ \square \varphi \equiv \neg \Diamond \neg \varphi \]

\[ \Diamond \varphi \equiv \neg \square \neg \varphi \]

\[ \Diamond \varphi \equiv \top \cup \varphi \]

\[ \lambda = \lambda[1..\infty] \models \text{pos}_1 \]

\[ \text{pos}_1 \in \pi(\lambda'[0]) \]
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.

**CTL models**: transition systems

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

**Definition 3.9 (Semantics of CTL*: state formulae)**

\[ M, q \models E \varphi \quad \text{if, by definition, there is a path } \lambda, \text{ starting from } q, \text{ such that } M, \lambda \models \varphi; \]

\[ M, q \models A \varphi \quad \text{if, by definition, for all paths } \lambda, \text{ starting from } q, \text{ we have } M, \lambda \models \varphi. \]

**Definition 3.10 (Semantics of CTL*: path formulae)**

Exactly like for LTL!

- \[ M, \lambda \models \bigcirc \varphi \quad \text{iff } M, \lambda[1^\infty] \models \varphi; \]
- \[ M, \lambda \models \varphi \mathcal{U} \psi \quad \text{iff } M, \lambda[i^\infty] \models \varphi \text{ for some } i \geq 0, \text{ and } M, \lambda[j^\infty] \models \varphi \text{ for all } 0 \leq j \leq i. \]
3. Action and Time

Example: Rocket and Cargo

Exercise:
How to express that there is no possibility of a deadlock?

3. ATL

ATL: What Agents Can Achieve

- Temporal logic meets game theory
- Main idea: cooperation modalities

\(\langle\langle A\rangle\rangle \Phi\): coalition A has a collective strategy to enforce \(\Phi\)
3. Action and Time

3. ATL

- \(\langle\text{jamesbond}\rangle \Diamond \text{win}\): “James Bond has an infallible plan to eventually win”
- \(\langle\text{jamesbond}, \text{bondsgirl}\rangle \text{fun} \sqcup \text{shot}\): “James Bond and his girlfriend are able to have fun until someone shoots at them”
- "Vanilla" ATL: every temporal operator preceded by exactly one cooperation modality;
- ATL*: no syntactic restrictions;

Definition 3.11 (Concurrent Game Structure)

A **concurrent game structure** is a tuple 

\[ M = (\text{Agt}, Q, \pi, \text{Act}, d, o) \], where:

- **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**
- \(d : \text{Agt} \times Q \rightarrow \mathcal{P}(\text{Act})\) 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

ATL Models: Concurrent Game Structures

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

Rocket Example: Simultaneous Actions
Definition 3.12 (Strategy)
A strategy is a conditional plan. We represent strategies with functions $s_A : Q \rightarrow \text{Act}$.

Function $\text{out}(q, S_A)$ returns the set of all paths that may result from agents $A$ executing strategy $S_A$ from state $q$ onward.

Example: Robots and Carriage

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.
3. Action and Time

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: \(\Rightarrow\) Chapter 5.

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?

4. Combining Modalities

1. BDI (Cohen et al.)

4.1 BDI (Cohen et al.)

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
4. Combining Modalities

1. BDI (Cohen et al.)

Operator | Meaning
--- | ---
\( B_i \phi \) | agent \( i \) believes \( \phi \)
\( G_i \phi \) | agent \( i \) has goal of \( \phi \)
\( \bigcirc \alpha \) | action \( \alpha \) will happen next
Done \( \alpha \) | action \( \alpha \) has just happened

Additionally:

- Action constructors “;” and “?”, as in DL;
- Derived operators: \( \lozenge \alpha \) (sometime \( \alpha \)), \( \Box \alpha \) (always \( \alpha \)), \( (\text{Later } \phi) \): strict sometime, \( (\text{Before } \phi, \psi) \): \( \phi \) holds before \( \psi \).

All goals have to be dropped eventually:
\[ \lozenge \neg (G_i (\text{Later } \phi)) \]

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

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

4.2 BDI (Rao et al.)

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>( \lozenge \alpha )</td>
<td>( \exists x (\bigcirc \alpha ; \alpha) )</td>
<td>sometime ( \alpha )</td>
</tr>
<tr>
<td>( \Box \alpha )</td>
<td>( \neg \lozenge \neg \alpha )</td>
<td>always ( \alpha )</td>
</tr>
<tr>
<td>( (\text{Later } \phi) )</td>
<td>( \neg \phi \land \lozenge \phi )</td>
<td>strict sometime</td>
</tr>
<tr>
<td>( (\text{Before } \phi, \psi) )</td>
<td>( \phi \rightarrow \Box \psi )</td>
<td>( \phi ) holds before ( \psi )</td>
</tr>
</tbody>
</table>

Examples:

\[ G_{\text{citizen}} \Box \text{safe}_{\alpha} \]
\[ G_{\text{police}} B_{\text{citizen}} \Box \text{safe}_{\text{citizen}} \]
4. Combining Modalities  2. BDI (Rao et al.)

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

Example: Card Play

\[
\begin{array}{c}
W_1 \\
q_0 \\
AK \\
B_{opt} \\
B_{opt} \\
D_{opt} \\
D_{opt} \\
W_2 \\
q_0 \\
AK \\
AQ \\
KQ \\
W_3 \\
q_0 \\
AK \\
AQ \\
KQ \\
W_4 \\
q_0 \\
AK \\
AQ \\
KQ \\
end{array}
\]

\[\text{win}\]

\[\text{win}\]

\[\text{win}\]

\[\text{win}\]

\[\text{win}\]

\[\text{win}\]

\[\text{win}\]

\[\text{win}\]

\[\text{win}\]

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).

\[
\begin{array}{c}
\text{Bel}_A E \bigcirc \text{win} : \text{Agent } A \text{ believes that there is a way to win in one step}
\\
\text{Des}_A E \bigcirc \text{win} : \text{the agent desires that every path leads to a victory, so he does not have to worry about his decisions}
\\
\text{However, he does not believe it is possible:}
\\
\neg \text{Bel}_A E \bigcirc \text{win}
\\
\text{On the other hand, he believes that a real optimist would believe that winning forever is inevitable: Bel}_A \text{Bel}_{opt} E \bigcirc \Box \text{win.}
\end{array}
\]
4. Combining Modalities

2. BDI (Rao et al.)

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.

3. Beware!

4.3 Beware!

- Let’s put ATL and epistemic logic in one box. Alternating-time Temporal Epistemic Logic
- Problems!
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 \( q'' \).

\[
\text{start} \rightarrow \langle\langle a\rangle\rangle \Diamond \text{win} \\
\text{start} \rightarrow K_a \langle\langle a\rangle\rangle \Diamond \text{win}
\]
4. Combining Modalities


4. References


5. Modalities in Action

Chapter 5. Modalities in Action

In this chapter:

- 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 $q$?
- Global model checking: Return the exact set of states $q$ in $M$ such that $\varphi$ holds in $M, q$
- Natural for verification of existing systems; also during design (“prototyping”)

Verification example: we want to make sure that the cargo can be always moved to the other location.

$$A(\vartriangleright_{\text{caL}} \rightarrow \vartriangleright_{\text{caP}} \land \vartriangleright_{\text{caP}} \rightarrow \vartriangleright_{\text{caL}} \land \vartriangleright_{\text{caR}} \rightarrow (\vartriangleright_{\text{caL}} \lor \vartriangleright_{\text{caP}}))$$

Function: $mcheck(M, \varphi)$.
Model checking formulae of CTL.
Returns the exact subset of $Q$ for which formula $\varphi$ holds.

\[
\begin{align*}
\text{case } \varphi & \equiv p : \quad \text{return } \{ q \in Q \mid p \in \pi(q) \} \\
\text{case } \varphi & \equiv \neg \psi : \quad \text{return } Q \setminus mcheck(M, \psi) \\
\text{case } \varphi & \equiv \psi_1 \land \psi_2 : \quad \text{return } mcheck(M, \psi_1) \cap mcheck(M, \psi_2) \\
\text{case } \varphi & \equiv \vartriangleright_{\psi} : \quad \text{return } \text{pre}(mcheck(M, \psi)) \\
\text{case } \varphi & \equiv \varsquare_{\psi} :
\begin{align*}
Q_1 & := Q; \quad Q_2 := Q_3 := mcheck(M, \psi); \\
& \text{while } Q_1 \not\subseteq Q_2 \text{ do } Q_1 := Q_1 \cap Q_2; \quad Q_2 := \text{pre}(Q_1) \cap Q_3 \text{ od}; \\
& \text{return } Q_1 \\
\text{case } \varphi & \equiv \vartriangleright_{\psi_1} U \psi_2 :
\begin{align*}
Q_1 & := \emptyset; \quad Q_2 := mcheck(M, \psi_2); \quad Q_3 := mcheck(M, \psi_1); \\
& \text{while } Q_2 \not\subseteq Q_1 \text{ do } Q_1 := Q_1 \cup Q_2; \quad Q_2 := \text{pre}(Q_1) \cap Q_3 \text{ od}; \\
& \text{return } Q_1
end case
\end{align*}
\end{align*}
\]
5. Modalities in Action

Example: Modified Rocket Domain, $\Box \text{caL}$

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.

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

Function $\text{plan}(\psi)$:
- Returns a subset of $Q$ for which formula $\psi$ holds, together with a (conditional) plan to achieve $\psi$. The plan is sought within the context of concurrent game structure $S = \langle Agt, Q, \Pi, \pi, o \rangle$.

```plaintext
function plan(\psi):
    return a subset of Q for which formula \psi holds, together with a (conditional)
    plan to achieve \psi. The plan is sought within the context of concurrent
    game structure S = \langle Agt, Q, \Pi, \pi, o \rangle.

    case \psi \in \Pi:
       return \{ \langle q, - \rangle | q \in \pi(\psi) \}
    case \psi = \neg \varphi:
       P_1 := plan(\varphi);
       return \{ \langle q, - \rangle | q \notin states(P_1) \}
    case \psi = \psi_1 \lor \psi_2:
       P_1 := plan(\psi_1);
       P_2 := plan(\psi_2);
       return \{ \langle q, - \rangle | q \in states(P_1) \cup states(P_2) \}
    case \psi = [\langle A \rangle] \Box \varphi:
       return pre(A, states(plan(\psi)))
    case \psi = [\langle A \rangle] \Box \varphi:
       P_1 := plan(\psi_1);
       P_2 := plan(\psi_2);
       Q_1 := states(P_1);
       do while states(P_1) \notin states(P_2)
       Q_2 := states(P_2);
       P_2 := pre(A, states(P_1)) \cap Q_2 od;
       return P_2 \setminus states(P_1)
    case \psi = [\langle A \rangle] \Box \varphi:
       P_1 := \emptyset;
       Q_1 := states(plan(\psi_1));
       P_2 := plan(true) \setminus states(plan(\psi_2));
       while states(P_2) \notin states(P_1)
       P_2 := pre(A, states(P_1)) \cap Q_2 od;
       return P_2
end case
```
Auxiliary functions:

- **Weakest precondition:**
  \[ \text{pre}(A, Q_1) = \{ (q, \sigma_A) \mid \forall \sigma_{\Sigma \setminus A} \delta(q, \sigma_A, \sigma_{\Sigma \setminus A}) \in Q_1 \} \]

- **States for which plan \( P \) is defined:**
  \[ \text{states}(P) = \{ q \in Q \mid \exists \sigma(q, \sigma) \in P \} \]

- **Extending plan \( P_1 \) with \( P_2 \):**
  \[ P_1 \oplus P_2 = P_1 \cup \{ (q, \sigma) \in P_2 \mid q \notin \text{states}(P_1) \} \]

- **\( P \) restricted to the states from \( Q_1 \):**
  \[ P|_{Q_1} = \{ (q, \sigma) \in P \mid q \in Q_1 \} \]

---

5. Modalities in Action

**5. Modalities in Action**

**1. Verification and Planning**

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.

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

ATL model checking (wrt CGS) is $\Delta^P_3$-complete with respect to the number of states and agents. For positive ATL model checking (wrt CGS) is even $\Sigma^P_2$-complete.
Some Complexity Classes

- **P**: problems solvable in **polynomial time** by a deterministic Turing machine
- **NP**: problems solvable in **polynomial time** by a non-deterministic Turing machine
- **PSPACE**: problems solvable in **polynomial space**
- **EXPTIME**: problems solvable in **exponential time**

What is this about? **Scalability!**

Complexity Class $\Sigma^P_2$

- $\Sigma^P_1$: problems solvable in **polynomial time** by a non-deterministic Turing machine making adaptive queries to a $\Sigma^P_1$ 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

Complexity of Temporal and Strategic Logics

<table>
<thead>
<tr>
<th></th>
<th>$m, l$</th>
<th>$n, k, l$</th>
<th>$n_{local}, k, l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATL$_{ir}$</td>
<td>$\Delta^P_1$-complete [4,7]</td>
<td>$\Delta^P_1$-complete [7]</td>
<td>PSPACE-complete [9]</td>
</tr>
</tbody>
</table>

Some variants: Information vs. recall

- **ATL$_P$**: Perfect information, no memory (**imperfect recall**). This is classical ATL, see [3].
- **ATL$_{IR}$**: Perfect information, full memory (**perfect recall**). Same complexity as ATL: polynomial in $m, l$. See [3].
- **ATL$_{IR}$**: Imperfect information, imperfect recall: see last slide.
- **ATL$_{IR}$**: Imperfect information, perfect recall: **undecidable** (currently working on a proof).
Main message:
- Complexity is very sensitive to the context!
- In particular, the way we define the input, and measure its size, is crucial.

Even if model checking appears very easy, it can be very hard.

Still, people do automatic model checking!
- LTL: SPIN
- CTL/ATL: MOCHA, MCMAS, VeriCS

Even if model checking is theoretically hard, it can be feasible in practice.

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 $\mathcal{L}_p$ of predicate symbols, a set $\mathcal{L}_v$ of variable symbols, a set $\mathcal{L}_c$ of constant symbols, the quantifiers $\forall$ and $\exists$, and the set $\mathcal{L}_t$ of terms (constants and variables). The set $\text{Fml}$ is defined by:

- If $t_1, \ldots, t_n$ are in $\mathcal{L}_t$ and $p$ is a predicate symbol of arity $n$, then $p(t_1, \ldots, t_n)$ is in $\text{Fml}$.
- $\top$ (true) and $\bot$ (false) are in $\text{Fml}$.
- If $A$ and $B$ are in $\text{Fml}$, then so are $\neg A, A \land B, A \mathop{U} B, A \mathop{S} B$, and $(A)$.
- If $A$ is in $\text{Fml}$ and $v$ is in $\mathcal{L}_v$, then $\exists v.A$ and $\forall v.A$ are both in $\text{Fml}$.

Temporal operators:

- $\phi \mathop{U} \psi$ holds until $\psi$ (primitive)
- $\phi \mathop{S} \psi$ since $\psi$ (primitive)
- $\Box \phi$ is true in the next state
- $\Diamond \phi$ the current state is not the initial state, and $\phi$ was true in the previous state
- $\lozenge \phi$ if the current state is not the initial state, then $\phi$ was true in the previous state
- $\Diamond \phi$ $\phi$ will be true in some future state
- $\lozenge \phi$ $\phi$ was true in some past state
- $\Box \phi$ $\phi$ will be true in all future states
- $\Diamond \phi$ $\phi$ was true in all past states

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 $\mathcal{D}$ which is assumed to be constant for every state,
- and mappings from elements of the language into denotations.

Definition 5.1 (FML model)

A FML model is a tuple $\mathcal{M} = (\sigma, \mathcal{D}, h_c, h_p)$ where

- $\sigma$ is the ordered set of states $s_0, s_1, s_2, \ldots$,
- $h_c$ is a map from the constants into $\mathcal{D}$, and
- $h_p$ is a map from $\mathcal{N} \times \mathcal{L}_p$ into $\mathcal{D}^{n} \rightarrow \{ \text{true}, \text{false} \}$ (the first argument of $h_p$ is the index $i$ of the state $s_i$).

Thus, for a particular state $s$, and a particular predicate $p$ of arity $n$, $h(s, p)$ gives truth values to atoms constructed from $n$-tuples of elements of $\mathcal{D}$.

A variable assignment $h_v$ is a mapping from the variables into elements of $\mathcal{D}$. Given a variable and the valuation function $h_v$, a term assignment $\tau_{vh}$ is a mapping from terms into $\mathcal{D}$ defined in the usual way.
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.

\[
\begin{align*}
(M, i, h_i) & \models \top \\
(M, i, h_i) & \not\models \bot \\
(M, i, h_i) & \models p(x_1, \ldots, x_n) \iff h_p(i, p)(\tau_{ih}(x_1), \ldots, \tau_{ih}(x_n)) = \text{true} \\
(M, i, h_i) & \models \neg \varphi \iff \langle M, i, h_i \rangle \not\models \varphi \\
(M, i, h_i) & \models \varphi \lor \psi \iff \langle M, i, h_i \rangle \models \varphi \text{ or } \langle M, i, h_i \rangle \models \psi \\
(M, i, h_i) & \models \varphi, \psi \iff \text{for some } k \text{ s.t. } i < k, \langle M, k, h_i \rangle \models \psi \\
(M, i, h_i) & \models \forall x. \varphi \iff \text{for some } k \text{ s.t. } 0 \leq k < i, \langle M, k, h_i \rangle \models \psi \\
(M, i, h_i) & \models \exists x. \varphi \iff \text{for all } k \text{ s.t. } k < j < i \text{ then } \langle M, j, h_i \rangle \models \psi \\
(M, i, h_i) & \models [d/x] \varphi \iff \text{there exists } d \text{ s.t. } \langle M, i, h_i[d/x] \rangle \models \varphi
\end{align*}
\]

**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.

**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:

antecedent: past \( \rightarrow \) consequent: future

The intuitive interpretation of such a rule is:

on the basis of the past, do the future
5. Modalities in Action
2. Temporal Programming

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);
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 Concurrent METATEM program for the seller agent may be as follows.

The interface of the seller agent is the following

\[ \text{seller}(\text{contractProposal})[\text{accept, refuse, contractProposal, ship}] \]

meaning that:

- the seller agent, identified by the seller identifier, is able to recognize a contractProposal message with its arguments, not specified in the interface;
- the messages that the seller agent is able to broadcast to the environment, including both communicative acts and actions on the environment, are accept, refuse, contractProposal, ship.

The internal knowledge base of the seller agent contains the following rigid predicates (predicates whose value never changes):

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

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

- \( \text{storing} \text{(orange, 1000)}. \)

The program rules of the seller agent are the following ones (lowercase symbols = constants, uppercase = variables):

\[
\forall \text{Buyer, Merchandise, Req_Amnt, Price} \\
\bigvee \text{contractProposal(Buyer, seller, Merchandise, Req_Amnt, Price)} \land \\
\text{storing} \text{(Merchandise, Old_Amout)} \land \text{Old_Amount} \geq \text{Req_Amout} \land \\
\text{max} \text{price(Merchandise, Max)} \land \text{Price} \geq \text{Max} \implies \\
[\text{ship(Buyer, Merchandise, Req_Amnt, Price)} \land \\
\text{accept(seller, Buyer, Merchandise, Req_Amnt, Price)}] \
\]

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

\[ \text{contractProposal}(Buyer, seller, Merchandise, Req_Amnt, Price) \land \\
\text{storing}(Merchandise, Old_Amount) \land \\
\text{min} - \text{price}(Merchandise, Min) \land \\
Old_Amount < Req_Amnt \lor Price \geq Min \land \\
\text{refuse}(seller, Buyer, Merchandise, Req_Amnt, Price) \]

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.

\[ \text{contractProposal}(Buyer, seller, Merchandise, Req_Amnt, Price) \land \\
\text{storing}(Merchandise, Old_Amount) \land \\
\text{min} - \text{price}(Merchandise, Min) \land \\
\text{max} - \text{price}(Merchandise, Max) \land Old_Amount \geq Req_Amnt \land \\
Price > Min \land Price < Max \land \text{New_Price} = (\text{Max} + Price) / 2 \implies \\
\text{contractProposal}(seller, Buyer, Merchandise, Req_Amnt, New_Price) \]

If there was a previous state where Buyer sent a contractProposal message to seller, and in that previous state the conditions were met to send a contractProposal back to Buyer, then send a contractProposal message to Buyer with a new proposed price.

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.
5. Modalities in Action  
3. Agent Programming

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:
- \( \text{is\_friend}(a,b) \) (facts),
- \( \text{turn}(I, left)^t \) (instantaneous actions).

No distinction between actions and facts.

Beliefs:
- \( B_I^{\text{now}} \text{go\_swimming}(I, CLZ)^t \): I believe now that I am going to swim in CLZ at time \( t \).
- \( B_I^{\text{now}} \text{ontable}(\text{block}_b)^t \).
- \( B_a^3 B_b^4 \text{is\_friend}(a,b) \).
Obligations: (also called commitments) are the beliefs that one agent will create the truth of a statement (for another agent)

\[ \text{OBL}_{t,\text{you}} \text{go\_swimming}(\text{I}, \text{CLZ})^t+1, \]
(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: \[ \text{OBL}_{t,\text{you}} \text{in\_lecture}(\text{you})^t+1. \]

A decision is an obligation to oneself: \[ \text{DEC}_t \varphi := \text{OBL}_{t,a,a} \varphi. \]

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:

\[ \text{CAN}_t \text{now in\_lecture}(\text{you})^t \]
(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

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

“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).”

agent: this is the name of an agent;

action: a private or communicative action. Only primitive actions are allowed! Action “Find the gold and bring it home” is, most likely, not primitive. It requires a plan (a sequence of primitive actions).

msgcond: a message condition of the form (Sender Type Content).

mntlcond: this describes a condition about the own mental state of the agent (i.e., the agent’s beliefs and commitments);
5. Modalities in Action

3. Agent Programming

**Cycle**

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

1. At each time step:
   - gather incoming messages and update the mental state accordingly,
   - execute commitments (using capabilities).

More recent agent-oriented programming languages follow similar ideas!

What makes an agent do something?
5. Modalities in Action

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: 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.

Concurrent METATEM – A language for modeling reactive
systems.
In A. Bode, M. Reeve, and G. Wolf (Eds.), Proceedings of the 5th
International Conference on Parallel Architectures and Language,
Europe (PARLE’93), pp. 185–196. Springer-Verlag.
LNCS 694.

Concurrent METATEM processes – A language for distributed AI.
In E. Mosekilde (Ed.), Proceedings of the 1991 European Simulation
Multiconference. SCS Press.

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.

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.

Executable temporal logic for distributed AI.
In K. Sycara (Ed.), Proceedings of the 12th International Workshop
of Distributed AI (IWDAI’93), pp. 131–142.

[GiunchigliaTraverso00] Fausto Giunchiglia and Paolo Traverso.
In Susanne Biundo and Maria Fox (Eds.) Recent Advances in AI
Planning, 5th European Conference on Planning, ECP’99, Durham,
UK, September 8-10, 1999, Proceedings, 1–20, LNCS 1809,
5. Modalities in Action

4. References


