1. Basics about Logic

Chapter 1. Basics about Logic

Basics about Logic
1.1 Why Logic?
1.2 Sentential Logic (SL)
1.3 Examples
1.4 Calculi
1.5 First-Order Logic (FOL)
1.6 Logic Programming (LP)
1.7 References

We make the case for using logics as a representation formalism for reasoning about the world.

While almost everything can be done with logic, the formalization is often awkward and cumbersome. We illustrate this with the Wumpus world and Sudoku-puzzles.

We introduce two sorts of calculi for propositional logics: a Hilbert type and a resolution calculus.

We introduce first-order logic (FOL) and reconsider the Wumpus world. The dynamics of the changing world can be modeled with the terms: they enable us to explicitly denote the situation we are in and to reason about it: McCarthy's situation calculus.
1. Basics about Logic

One of the main features is to ask queries of the form $\exists \phi(x)$ to a theory $T$. We expect not just a “yes/no” answer, but an instantiation of the variable $x$. This can be achieved with the resolution calculus for FOL.

While resolution is much more efficiently implementable (compared to Hilbert-type calculi), the search space is still huge. Thus it was suggested to restrict resolution and apply it to a smaller class of formulae: Horn clauses.

This leads to PROLOG as an efficient inference engine for definite logic programs. However, full declarativness is lost and PROLOG is not (yet) the answer: emphasis is put on computing answer substitutions.

---

Why logic at all?

- framework for thinking about systems,
- makes one realise hidden 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,
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).

---

1. Basics about Logic

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

- \( \Box \) and \( \top \): the constants 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),
- \( (, ) \): 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 \( v \) 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 \( v(\Box) = \text{false}, \ v(\top) = \text{true} \). Each valuation \( v \) can be uniquely extended to a function \( \bar{v} : Fml_L \rightarrow \{ \text{true}, \text{false} \} \) so that:

- \( \bar{v}(\neg p) = \begin{cases} \text{true}, & \text{if } \bar{v}(p) = \text{false} \\ \text{false}, & \text{if } \bar{v}(p) = \text{true} \end{cases} \)
- \( \bar{v}(\varphi \land \gamma) = \begin{cases} \text{true}, & \text{if } \bar{v}(\varphi) = \text{true and } \bar{v}(\gamma) = \text{true} \\ \text{false}, & \text{else} \end{cases} \)
- \( \bar{v}(\varphi \lor \gamma) = \begin{cases} \text{true}, & \text{if } \bar{v}(\varphi) = \text{true or } \bar{v}(\gamma) = \text{true} \\ \text{false}, & \text{else} \end{cases} \)
Definition (continued)

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

Thus each valuation \(v\) uniquely defines a \(\bar{v}\). We call \(\bar{v}\) a \(\mathcal{L}\)-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 \(\mathcal{A}\) is a model of \(\varphi\). From now on we will speak of models, structures and valuations synonymously.

Semantics

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

Definition 1.3 (Validity of a Formula, Tautology)

1. A formula \(\varphi \in \text{Fml}_{\mathcal{L}}\) 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: \(T \subseteq \text{Fml}_{\mathcal{L}}\). \(v\) satisfies \(T\) if \(\bar{v}(\varphi) = \text{true}\) for all \(\varphi \in T\). We write \(v \models T\).

3. A \(\mathcal{L}\)-formula \(\varphi\) is called \(\mathcal{L}\)-tautology if for all possible valuations \(v\) in \(\mathcal{L}\) \(v \models \varphi\) holds.

From now on we suppress the language \(\mathcal{L}\), because it is obvious from context. Nevertheless it needs to be carefully defined.

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

\[Cn_{\mathcal{L}}(T) = \text{def} \{ \varphi \in \text{Fml}_{\mathcal{L}} : T \models \varphi \},\]

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. Basics about Logic

2. Sentential Logic (SL)

**Definition 1.7 (MOD(T), Cn(˚U))**

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

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

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

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

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

\( T \) is called complete if for each formula \( \phi \in Fml_L \) \( T \models \phi \) or \( T \not\models \neg \phi \) holds.

**Attention:**

Do not mix up this last condition with the property of a valuation (model) \( v \): 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) \( v \) with \( v(\phi) = \text{true} \) for all \( \phi \in T \).

**Lemma 1.10 (Ex Falso Quodlibet)**

\( T \) is consistent if and only if \( \text{Cn}(T) \neq Fml_L \).
### Wumpus World

1. Basics about Logic

#### 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:
\[-S_{1,1} \land -S_{2,1} \land S_{1,2} \land -B_{1,1} \land -B_{2,1} \land -B_{1,2}\]

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

Answer:
Yes. Either via resolution or using a Hilbert-calculus.

Problem:
We want more: given a certain situation we would like to determine the best action, i.e. to ask a query which gives us back such an action. This is impossible in SL: we can only check for each action whether it is good or not and then, by comparison, try to find the best action.

But we can check for each action if it should be done or not. Therefore we need additional axioms:

\[
\begin{align*}
A_{1,1} \land East \land W_{2,1} & \rightarrow -Forward \\
A_{1,1} \land East \land Pit_{2,1} & \rightarrow -Forward \\
A_{i,j} \land Gl_{i,j} & \rightarrow TakeGold
\end{align*}
\]

Disadvantages

- actions can only be guessed
- database must be changed continuously
- the set of rules becomes very big because there are no variables

Using an appropriate formalisation (additional axioms) we can check if

\[KB \vdash \neg action \text{ or } KB \vdash \neg action\]

But it can happen that neither one nor the other is deducible.
**Example 1.11 (CLIMA/Agent Contest)**

A simple grid where agents are supposed to collect gold. Maze-like environments. Different roles of agents: scouts, collectors. Evaluating and testing MAS.

- 2005: [http://clima.deis.unibo.it/contest.html](http://clima.deis.unibo.it/contest.html)
- 2006: [http://cig.in.tu-clausthal.de/CLIMAContest/](http://cig.in.tu-clausthal.de/CLIMAContest/)

---

**Sudoku**

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

![Sudoku puzzle](image)

**Table 1:** A simple Sudoku ($S_1$)

---

**Can they be solved with sentential logic?**

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

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

**Solution**

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

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

---

This completes the language, the **syntax**. How many symbols are these?
1. Basics about Logic

We distinguished between the puzzle $S$ and a solution $S$ of it.

**What is a model (or valuation) in the sense of Definition 1.2?**

<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></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>3</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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
- $\text{neun}_{i,j}$ 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 $1 \leq i, j \leq 9$).

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

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

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

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

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?**

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

First square: $T_1$

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

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

Second square: $T_2$

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.

**First Row:** $T_{\text{Row 1}}$
- $\text{eins}_{1,1} \lor \text{eins}_{1,2} \lor \ldots \lor \text{eins}_{1,9}$
- $\text{zwei}_{1,1} \lor \text{zwei}_{1,2} \lor \ldots \lor \text{zwei}_{1,9}$
- $\text{drei}_{1,1} \lor \text{drei}_{1,2} \lor \ldots \lor \text{drei}_{1,9}$
- $\text{vier}_{1,1} \lor \text{vier}_{1,2} \lor \ldots \lor \text{vier}_{1,9}$
- $\text{fuenf}_{1,1} \lor \text{fuenf}_{1,2} \lor \ldots \lor \text{fuenf}_{1,9}$
- $\text{sechs}_{1,1} \lor \text{sechs}_{1,2} \lor \ldots \lor \text{sechs}_{1,9}$
- $\text{sieben}_{1,1} \lor \text{sieben}_{1,2} \lor \ldots \lor \text{sieben}_{1,9}$
- $\text{acht}_{1,1} \lor \text{acht}_{1,2} \lor \ldots \lor \text{acht}_{1,9}$
- $\text{neun}_{1,1} \lor \text{neun}_{1,2} \lor \ldots \lor \text{neun}_{1,9}$

**Ninth Row:** $T_{\text{Row 9}}$
- $\text{eins}_{9,1} \lor \text{eins}_{9,2} \lor \ldots \lor \text{eins}_{9,9}$
- $\text{zwei}_{9,1} \lor \text{zwei}_{9,2} \lor \ldots \lor \text{zwei}_{9,9}$
- $\text{drei}_{9,1} \lor \text{drei}_{9,2} \lor \ldots \lor \text{drei}_{9,9}$
- $\text{vier}_{9,1} \lor \text{vier}_{9,2} \lor \ldots \lor \text{vier}_{9,9}$
- $\text{fuenf}_{9,1} \lor \text{fuenf}_{9,2} \lor \ldots \lor \text{fuenf}_{9,9}$
- $\text{sechs}_{9,1} \lor \text{sechs}_{9,2} \lor \ldots \lor \text{sechs}_{9,9}$
- $\text{sieben}_{9,1} \lor \text{sieben}_{9,2} \lor \ldots \lor \text{sieben}_{9,9}$
- $\text{acht}_{9,1} \lor \text{acht}_{9,2} \lor \ldots \lor \text{acht}_{9,9}$
- $\text{neun}_{9,1} \lor \text{neun}_{9,2} \lor \ldots \lor \text{neun}_{9,9}$

Is that sufficient? What if a row contains several 1's?
1. Basics about Logic

First Column: \( T_{\text{Column 1}} \)

\[
\begin{align*}
1 & \text{ eins}_1 \lor \text{ eins}_2 \lor \ldots \lor \text{ eins}_9 \\
2 & \text{ zwei}_1 \lor \text{ zwei}_2 \lor \ldots \lor \text{ zwei}_9 \\
3 & \text{ drei}_1 \lor \text{ drei}_2 \lor \ldots \lor \text{ drei}_9 \\
4 & \text{ vier}_1 \lor \text{ vier}_2 \lor \ldots \lor \text{ vier}_9 \\
5 & \text{ fuenf}_1 \lor \text{ fuenf}_2 \lor \ldots \lor \text{ fuenf}_9 \\
6 & \text{ sechs}_1 \lor \text{ sechs}_2 \lor \ldots \lor \text{ sechs}_9 \\
7 & \text{ sieben}_1 \lor \text{ sieben}_2 \lor \ldots \lor \text{ sieben}_9 \\
8 & \text{ acht}_1 \lor \text{ acht}_2 \lor \ldots \lor \text{ acht}_9 \\
9 & \text{ neun}_1 \lor \text{ neun}_2 \lor \ldots \lor \text{ neun}_9
\end{align*}
\]

Analogously for all other columns.

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

All put together:

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

\[
T_{\text{Row 1}} \cup \ldots \cup T_{\text{Row 9}}
\]

\[
T_{\text{Column 1}} \cup \ldots \cup T_{\text{Column 9}}
\]

Here is a more difficult one.

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

Table 3: A difficult Sudoku \( S_{\text{difficult}} \)
1. Basics about Logic  4. Calculi

A general notion of a certain sort of calculi.

**Definition 1.12 (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 \( Fml_{\mathcal{L}} \), the set of well-formed formulae in \( \mathcal{L} \): they are called **axioms**,
- **Inf:** is a set of pairs written in the form
  \[
  \phi_1, \phi_2, \ldots, \phi_n \rightarrow \psi
  \]
  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.

**Definition (continued)**

We now define a particular instance of our general notion.

**Definition 1.13 (Calculus for Sentential Logic SL)**

We define \( \text{Hilbert}^{\text{SL}}_{\mathcal{L}} \) as follows. The underlying language is \( \mathcal{L} \) with the well-formed formulae \( Fml_{\mathcal{L}} \) as defined in Definition 1.1.

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

1. \( \phi \rightarrow \top, \top \rightarrow \phi, \top \rightarrow \top \)
2. \( (\phi \land \psi) \rightarrow \phi, (\phi \land \psi) \rightarrow \psi \)
3. \( \phi \rightarrow (\phi \lor \psi), (\phi \lor \psi) \rightarrow \psi \)
4. \( \lnot \phi \rightarrow \phi, (\phi \rightarrow \psi) \rightarrow ((\phi \rightarrow \lnot \psi) \rightarrow \phi) \)
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.

**Definition 1.14 (Proof)**

A **proof** of a formula \( \phi \) from a theory \( T \subseteq Fml_{\mathcal{L}} \) 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 a substitution instance of an axiom,
- \( \varphi_i \in T \),
- there is \( \varphi_i, \varphi_k = (\varphi_l \rightarrow \varphi_i) \) 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 1.15 (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:

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

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

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

**Theorem 1.16 (Correct-, Completeness for Hilbert \text{SL})**

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

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

---

We define the following inference rule on \( Fml_{L_{res}}^{Res} \):

**Definition 1.18 (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 \( C_1 \lor C_2 = \emptyset \).
If we use the set-notation for clauses, we can formulate the inference rule as follows:

**Definition 1.19 (SL resolution (Set notation))**

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

\[
\begin{array}{c}
C_1 \cup \{A\}, \quad C_2 \cup \{\neg A\} \\
\hline
C_1 \cup C_2
\end{array}
\]

(Res)

Again, we identify the empty set $\emptyset$ with $\square$.

**Definition 1.20 (Resolution Calculus for SL)**

We define the resolution calculus $\text{Robinson}^{SL}_{\text{Res}} = \langle \emptyset, \{\text{Res}\} \rangle$ as follows. The underlying language is $L_{\text{res}} \subseteq L_{SL}$ defined on Slide 58 together with the well-formed formulae $Fml_{L_{\text{res}}}$.

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

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

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

**Theorem 1.21 (Completeness of Resolution Refutation)**

If $M$ is an unsatisfiable set of clauses then the empty clause $\square$ can be derived in $\text{Robinson}^{SL}_{\text{Res}}$.

We also say that resolution is refutation complete.
Definition 1.22 (First order logic $L_{\text{FOL}}$, $L \subseteq L_{\text{FOL}}$)

The language $L_{\text{FOL}}$ of first order logic (Praedikatenlogik erster Stufe) is:

- $x, y, z, x_1, x_2, \ldots, x_n, \ldots$: a countable set $\text{Var}$ of variables
- for each $k \in \mathbb{N}_0$: $P^k_1, P^k_2, \ldots, P^k_n, \ldots$: a countable set $\text{Pred}^k$ of $k$-dimensional predicate symbols (the 0-dimensional predicate symbols are the propositional logic constants from $\text{At}$ of $L_{\text{SL}}$). We suppose that $\square$ and $\top$ are available.
- for each $k \in \mathbb{N}_0$: $f^k_1, f^k_2, \ldots, f^k_n, \ldots$: a countable set $\text{Funct}^k$ of $k$-dimensional function symbols
- $\neg, \land, \lor, \rightarrow$: the sentential connectives
- $(, )$: the parentheses
- $\forall, \exists$: quantifiers

Definition (continued)

The 0-dimensional function symbols are called *individuum constants* — we leave out the parentheses. In general we will need — as in propositional logic — only a certain subset of the predicate or function symbols. These define a language $L \subseteq L_{\text{FOL}}$ (analogously to definition 1.1 on page 2). The used set of predicate and function symbols is also called *signature* $\Sigma$.

Definition (continued)

The concept of an $L$-term $t$ and an $L$-formula $\phi$ are defined inductively:

**Term:** $L$-terms $t$ are defined as follows:

1. Each variable is an $L$-term.
2. If $f^k$ is a $k$-dimensional function symbol from $L$ and $t_1, \ldots, t_k$ are $L$-terms, then $f^k(t_1, \ldots, t_k)$ is an $L$-Term.

The set of all $L$-terms that one can create from the set $X \subseteq \text{Var}$ is called $\text{Term}_L(X)$ or $\text{Term}_\Sigma(X)$. Using $X = \emptyset$ we get the set of basic terms $\text{Term}_L(\emptyset)$, short: $\text{Term}_L$.

Definition (continued)

**Formula:** $L$-formulae $\phi$ are also defined inductively:

1. If $P^k$ is a $k$-dimensional predicate symbol from $L$ and $t_1, \ldots, t_k$ are $L$-terms then $P^k(t_1, \ldots, t_k)$ is a $L$-formula.
2. For all $L$-formulae $\phi$ is ($\neg \phi$) a $L$-formula.
3. For all $L$-formulae $\phi$ and $\psi$ are ($\phi \land \psi$) and ($\phi \lor \psi$) $L$-formulae.
4. If $x$ is a variable and $\phi$ a $L$-formula then are ($\exists x \phi$) and ($\forall x \phi$) $L$-formulae.
Definition (continued)

Atomic $\mathcal{L}$-formulae are those which are composed according to 1., we call them $At_{\mathcal{L}}(X)$ ($X \subseteq \text{Var}$). The set of all $\mathcal{L}$-formulae in respect to $X$ is called $Fml_{\mathcal{L}}(X)$.

Positive formulae ($Fml_{\mathcal{L}}^+(X)$) are those which are composed using only 1, 3. and 4.

If $\phi$ is a $\mathcal{L}$-formula and is part of another $\mathcal{L}$-formula $\psi$ then $\phi$ is called sub-formula of $\psi$.

An illustrating example

Example 1.23 (From semigroups to rings)

We consider $\mathcal{L} = \{0, 1, +, \cdot, \leq, =\}$, where 0, 1 are constants, +, $\cdot$ binary operations and $\leq, =$ binary relations. What can be expressed in this language?

Ax 1: $\forall x \forall y \forall z \ x + (y + z) = (x + y) + z$
Ax 2: $\forall x \ (x + 0 = 0 + x) \wedge (0 + x = x)$
Ax 3: $\forall x \exists y \ (x + y = 0) \wedge (y + x = 0)$
Ax 4: $\forall x \forall y \ x + y = y + x$
Ax 5: $\forall x \forall y \forall z \ x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Ax 6: $\forall x \forall y \forall z \ (x + y) \cdot z = x \cdot (y + z)$
Ax 7: $\forall x \forall y \forall z \ (x \cdot (y + z)) = (x \cdot y) + (x \cdot z)$

Axiom 1 describes an semigroup, the axioms 1-2 describe a monoid, the axioms 1-3 a group, and the axioms 1-7 a ring.

Definition 1.24 ($\mathcal{L}$-structure $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$)

A $\mathcal{L}$-structure or a $\mathcal{L}$-interpretation is a pair $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ with $U_{\mathcal{A}}$ being an arbitrary non-empty set, which is called the basic set (the universe or the individuum range) of $\mathcal{A}$. Further $I_{\mathcal{A}}$ is a mapping which

- assigns to each $k$-dimensional predicate symbol $P^k$ in $\mathcal{L}$ a $k$-dimensional predicate over $U_{\mathcal{A}}$
- assigns to each $k$-dimensional function symbol $f^k$ in $\mathcal{L}$ a $k$-dimensional function on $U_{\mathcal{A}}$

In other words: the domain of $I_{\mathcal{A}}$ is exactly the set of predicate and function symbols of $\mathcal{L}$.

Definition (continued)

The range of $I_{\mathcal{A}}$ consists of the predicates and functions on $U_{\mathcal{A}}$. We write:

$$I_{\mathcal{A}}(P) = P^A, \ I_{\mathcal{A}}(f) = f^A.$$ 

$\phi$ be a $\mathcal{L}_1$-formula and $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ a $\mathcal{L}$-structure. $\mathcal{A}$ is called matching with $\phi$ if $I_{\mathcal{A}}$ is defined for all predicate and function symbols which appear in $\phi$, i.e. if $\mathcal{L}_1 \subseteq \mathcal{L}$. 

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007)
Definition 1.25 (Variable assignment \( \rho \))

A variable assignment \( \rho \) over a \( L \)-structure \( \mathcal{A} = (U_\mathcal{A}, I_\mathcal{A}) \) is a function

\[ \rho : \text{Var} \rightarrow U_\mathcal{A}; \; x \mapsto \rho(x). \]

Definition (continued)

We define inductively the logical value of a formula \( \varphi \) in \( \mathcal{A} \):

1. if \( \varphi = \text{def} P^k(t_1, \ldots, t_k) \) with the terms \( t_1, \ldots, t_k \) and the \( k \)-dimensional predicate symbol \( P^k \), then
   \[ \mathcal{A}(\varphi) = \begin{cases} 
   \text{true}, & \text{if } (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \in P^k, \\
   \text{false}, & \text{else}.
   \end{cases} \]

2. if \( \varphi = \text{def} \neg \psi \), then
   \[ \mathcal{A}(\varphi) = \begin{cases} 
   \text{true}, & \text{if } \mathcal{A}(\psi) = \text{false}, \\
   \text{false}, & \text{else}.
   \end{cases} \]

3. if \( \varphi = \text{def} \psi \land \eta \), then
   \[ \mathcal{A}(\varphi) = \begin{cases} 
   \text{true}, & \text{if } \mathcal{A}(\psi) = \text{true and } \mathcal{A}(\eta) = \text{true}, \\
   \text{false}, & \text{else}.
   \end{cases} \]

4. if \( \varphi = \text{def} (\psi \lor \eta) \), then
   \[ \mathcal{A}(\varphi) = \begin{cases} 
   \text{true}, & \text{if } \mathcal{A}(\psi) = \text{true or } \mathcal{A}(\eta) = \text{true}, \\
   \text{false}, & \text{else}.
   \end{cases} \]

5. if \( \varphi = \text{def} \forall x \psi \), then
   \[ \mathcal{A}(\varphi) = \begin{cases} 
   \text{true}, & \text{if } \forall d \in U_\mathcal{A} : \mathcal{A}_{[x/d]}(\psi) = \text{true}, \\
   \text{false}, & \text{else}.
   \end{cases} \]

6. if \( \varphi = \text{def} \exists x \psi \), then
   \[ \mathcal{A}(\varphi) = \begin{cases} 
   \text{true}, & \text{if } \exists d \in U_\mathcal{A} : \mathcal{A}_{[x/d]}(\psi) = \text{true}, \\
   \text{false}, & \text{else}.
   \end{cases} \]

In the cases 5. and 6. the notation \([x/d]\) was used. It is defined as follows: For \( d \in U_\mathcal{A} \) let \( \mathcal{A}_{[x/d]} \) be the structure \( \mathcal{A}' \), which is identical to \( \mathcal{A} \) except for the definition of \( x' \):

\[ x' = \text{def} d \] (whether \( I_\mathcal{A} \) is defined for \( x \) or not).
Definition (continued)

We write:

- \( A \models \varphi[p] \) for \( A(\varphi) = \text{true} \): \( A \) is a model for \( \varphi \) with respect to \( p \).
- If \( \varphi \) does not contain free variables, then \( A \models \varphi[p] \) is independent from \( p \). We simply leave out \( p \).
- If there is at least one model for \( \varphi \), then \( \varphi \) is called satisfiable or consistent.

A free variable is a variable which is not in the scope of a quantifier. For instance, \( z \) is a free variable of \( \forall x P(x, z) \) but not free (or bounded) in \( \forall z \exists x P(x, z) \).

Definition 1.27 (Tautology)

A theory is a set of formulae without free variables: \( T \subseteq \text{Fml}_L \). The structure \( A \) satisfies \( T \) if \( A \models \varphi \) holds for all \( \varphi \in T \). We write \( A \models T \) and call \( A \) a model of \( T \).

A \( L \)-formula \( \varphi \) is called \( L \)-tautology, if for all matching \( L \)-structures \( A \) the following holds: \( A \models \varphi \).

From now on we suppress the language \( L \), because it is obvious from context. Nevertheless it has to be defined.

Definition 1.28 (Consequence set \( Cn(T) \))

A formula \( \varphi \) follows semantically from \( T \), if for all structures \( A \) with \( A \models T \) also \( A \models \varphi \) holds. We write: \( T \models \varphi \).

In other words: all models of \( T \) do also satisfy \( \varphi \).

We denote by \( \text{Cn}_L(T) = \{ \varphi \in \text{Fml}_L : T \models \varphi \} \), or simply \( \text{Cn}(T) \), the semantic consequence operator.

Lemma 1.29 (Properties of \( \text{Cn}(T) \))

The semantic consequence operator has the following properties

1. \( T \)-extension: \( T \subseteq \text{Cn}(T) \),
2. Monotony: \( T \subseteq T' \Rightarrow \text{Cn}(T) \subseteq \text{Cn}(T') \),
3. Closure: \( \text{Cn}(\text{Cn}(T)) = \text{Cn}(T) \).
1. Basics about Logic

5. First-Order Logic (FOL)

**Lemma 1.30 (ϕ \not\in \text{Cn}(T))**

ϕ \not\in \text{Cn}(T) if and only if there is a structure \( A \) with \( A \models T \) and \( A \models \neg \varphi \).

In other words: ϕ \not\in \text{Cn}(T) if and only if there is a counterexample: a model of T in which ϕ is not true.

**Definition 1.31 (MOD(T), \text{Cn}(\mathcal{U}))**

If \( T \subseteq \text{Fml}_L \), then we denote by \( \text{MOD}(T) \) 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 structures then we can consider all sentences, which are true in all structures. We call this set also \( \text{Cn}(\mathcal{U}) \):

\[
\text{Cn}(\mathcal{U}) = \{ \varphi \in \text{Fml}_L : \forall A \in \mathcal{U} : A \models \varphi \}.
\]

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

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

1. Basics about Logic

5. First-Order Logic (FOL)

**Definition 1.32 (Completeness of a theory \( T \))**

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

Attention:

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

**Lemma 1.33 (Ex Falso Quodlibet)**

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

1. Basics about Logic

5. First-Order Logic (FOL)

**Example 1.34 (Natural numbers in different languages)**

- \( \mathcal{N}_{\text{PF}} = (\mathbb{N}_0, 0^\mathcal{N}, +^\mathcal{N}, =^\mathcal{N}) \) ("Presburger Arithmetic"),
- \( \mathcal{N}_{\text{PA}} = (\mathbb{N}_0, 0^\mathcal{N}, +^\mathcal{N}, \cdot^\mathcal{N}, =^\mathcal{N}) \) ("Peano Arithmetic"),
- \( \mathcal{N}_{\text{PA}^v} = (\mathbb{N}_0, 0^\mathcal{N}, 1^\mathcal{N}, +^\mathcal{N}, \cdot^\mathcal{N}, =^\mathcal{N}) \) (variant of \( \mathcal{N}_{\text{PA}} \)).

These sets each define the natural numbers, but in different languages.

**Question:**

If the language bigger is bigger then we can express more. Is \( L_{\text{PA}^v} \) more expressive than \( L_{\text{PA}} \)?
1. Basics about Logic

**Answer:**

No, because one can replace the $1^N$ by a $L_{PA}$-formula: there is a $L_{PA}$-formula $\phi(x)$ so that for each variable assignment $\rho$ the following holds:

$$\mathcal{N}_{PA} \models_\rho \phi(x) \text{ if and only if } \rho(x) = 1^N$$

- Thus we can define a **macro** for $1$.
- Each formula of $L_{PA^'}$ can be transformed into an equivalent formula of $L_{PA}$.

1. Basics about Logic

**Question:**

Is $L_{PA}$ perhaps more expressive than $L_{Pr}$, or can the multiplication be defined somehow?

We will see later that $L_{PA}$ is indeed more expressive:

- the set of sentences valid in $\mathcal{N}_{Pr}$ is **decidable**, whereas
- the set of sentences valid in $\mathcal{N}_{PA}$ is **not even recursively enumerable**.

1. Basics about Logic

**Definition 1.35 (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:

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

As for sentential logic, formulae can be derived from a given theory and they can also (semantically) follow from it.

**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.
1. Basics about Logic 5. First-Order Logic (FOL)

We have already defined a complete and correct calculus for sentential logic $L_{SL}$. Such calculi also exist for first order logic $L_{FOL}$.

Theorem 1.36 (Correct-, Completeness of FOL)

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

$$T |- \varphi \text{ if and only if } T |- \varphi$$

Theorem 1.37 (Compactness of FOL)

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

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

1. Basics about Logic 5. First-Order Logic (FOL)

Quite often the universes of models we are interested in consist exactly of the basic terms $Term_L(\emptyset)$. This leads to

Definition 1.39 (Herbrand model)

A model $A$ is called Herbrand model with respect to a language if the universe of $A$ consists exactly of $Term_L(\emptyset)$ and the function symbols $f^A_i$ are interpreted as follows:

$$f^A_i : \ Term_L(\emptyset) \times \ldots \times Term_L(\emptyset) \rightarrow Term_L(\emptyset);$$

$$(t_1, \ldots, t_k) \rightarrow f^A_i(t_1, \ldots, t_k)$$

We write $T |-_{\text{Herb}} \phi$ if each Herbrand model of $T$ is also a model of $\phi$.

A Herbrand model is uniquely determined by the predicates that are true in it. The whole set $B_L$ of all ground instances of predicates over the Herbrand universe is called Herbrand base.

Question:

Is $T |-_{\text{Herb}} \phi$ not much easier, because we have to consider only Herbrand models? Is it perhaps decidable?

No, truth in Herbrand models is highly undecidable.

The introduced relation $T |- \phi$ says that each model of $T$ is also a model of $\phi$. But because there are many models with very large universes the following question arises: can we restrict to particular models?

Theorem 1.38 (Löwenheim-Skolem)

$T |- \phi$ holds if and only if $\phi$ holds in all countable models of $T$. 
Wumpus world reconsidered in FOL

Question:
How do we axiomatize the Wumpus-world in FOL?

function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
  t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action

Idea:
In order to describe actions or their effects consistently we consider the world as a sequence of situations (snapshots of the world). Therefore we have to extend each predicate by an additional argument.

We use the function symbol

result(action, situation)

as the term for the situation which emerges when the action action is executed in the situation situation.

Actions: Turn_right, Turn_left, Foreward, Shoot, Grab, Release, Climb.

We also need a memory, a predicate

At(person, location, situation)

with person being either Wumpus or Agent and location being the actual position (stored as pair [i,j]).

Important axioms are the so called successor-state axioms, they describe how actions effect situations. The most general form of these axioms is

true afterwards \iff an action made it true or it is already true and no action made it false
Successor State Axiom

Axioms about $At(p,l,s)$:

\[
At(p,l,\text{result}(\text{Forward},s)) \leftrightarrow (l \land \text{location}_\text{ahead}(p,s) \land \neg \text{Wall}(l))
\]

\[
At(p,l,\text{result}(\text{Left},s)) \leftrightarrow (l \land \text{location}_\text{left}(p,s) \land \neg \text{Wall}(l))
\]

\[
At(p,l,\text{result}(\text{Right},s)) \leftrightarrow (l \land \text{location}_\text{right}(p,s) \land \neg \text{Wall}(l))
\]

This lead to the programming language \text{GOLOG}. 

The following theorem is the basic result for applying resolution. It states that \text{FOL} can be reduced to \text{SL}.

\text{Theorem 1.41 (Herbrand)}

\text{Let } T \text{ be universal and } \phi \text{ does not contain quantifiers. Then:}

\[ T \models \exists \psi \text{ if and only if there is } t_1, \ldots, t_n \in \text{Term}_L(\emptyset) \text{ with: } T \models \phi(t_1) \lor \ldots \lor \phi(t_n) \]

Or: Let $M$ be a set of clauses of \text{FOL} (formulae in the form $P_1(t_1) \lor \neg P_2(t_2) \lor \ldots \lor P_n(t_n)$ with $t_i \in \text{Term}_L(X)$). Then:

\text{M is unsatisfiable if and only if there is a finite and unsatisfiable set } M_{\text{inst}} \text{ of ground instances of } M.

\text{Our general goal is to derive an existentially quantified formula from a set of formulae:}

\[ M \leftarrow \exists \psi. \]

\text{To use resolution we must form } M \cup \{\neg \exists \psi\} \text{ and put it into the form of clauses. This set is called input.}

\text{Instead of allowing arbitrary resolvents, we try to restrict the search space.}
**Definition 1.42 (Most general unifier: mgU)**

Given a finite set of equations between terms or equations between literals. Then there is an algorithm which calculates a **most general solution substitution** (i.e., a substitution of the involved variables so that the left sides of all equations are syntactically identical to the right sides) or which returns **fail**. In the first case the **most general solution substitution** is defined (up to renaming of variables): it is called **mgU**, most general unifier.

We outline the mentioned algorithm using an example.

Given: \( f(x, g(h(y), y)) = f(x, g(z, a)) \)

The algorithm successively calculates the following sets of equations:

\[
\begin{align*}
&\{ x = x, \ g(h(y), y) = g(z, a) \} \\
&\{ g(h(y), y) = g(z, a) \} \\
&\{ h(y) = z, \ y = a \} \\
&\{ z = h(y), \ y = a \} \\
&\{ z = h(a), \ y = a \}
\end{align*}
\]

Thus the mgU \( \Theta \) is: \( \Theta = [x/x, y/a, z/h(a)] \).

### A resolution calculus for FOL

The resolution calculus is defined over the language \( L^{res} \subseteq L_{FOL} \) where the set of well-formed formulae \( Fml^{Res}_{L^{res}} \) 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 atoms or their negations. No implications or conjunctions are allowed. These formulae are also called **clauses**.

Such a clause is also written as the set

\[ \{ A, \neg B, C, \ldots, \neg E \}. \]

This means that the set-theoretic union of such sets corresponds again to a clause.

**Note:** a clause now consists of atoms rather than constants (as it was the case for SL).
1. Basics about Logic

Definition 1.43 (Robinson’s resolution for FOL)

The resolution calculus consists of two rules:

\[(\text{Res}) \quad C_1 \cup \{A_1\} \cup \{\neg A_2\} \quad (C_1 \cup C_2) \land \neg (A_1 \lor A_2)\]

where \(C_1 \cup \{A_1\}\) and \(C_2 \cup \{A_2\}\) are assumed to be disjunct wrt the variables, and the factorization rule

\[(\text{Fac}) \quad C_1 \cup \{L_1\} \land \neg (L_1 \lor L_2)\]

Consider for example \(M = \{r(x) \lor \neg p(x), p(a), s(a)\}\) and the question \(M \models \exists x(s(x) \land r(x))\)?

Question:

Why do we need factorization?

Answer (continued):

\[\Box \text{ can never be derived (using resolution only).}\]

Factorization instantly solves the problem, we can deduce both \(s(x)\) and \(\neg s(y)\), and from there the empty clause.

Theorem 1.45 (Resolution is refutation complete)

Robinson’s resolution calculus Robinson\(^{FOL}\)\(_{L_{res}}\) is refutation complete: given an unsatisfiable set, the empty clause can be derived using resolution and factorization.
Example 1.46 (Unlimited Resolution)

Let \( M := \{ r(x) \lor \neg p(x), p(a), s(a) \} \) and \( \square \leftarrow s(x) \land r(x) \) the query.

An unlimited resolution might look like this:

\[
\begin{array}{ll}
\begin{array}{c}
r(x) \lor \neg p(x) \quad p(a) \quad s(a) \quad \neg s(x) \lor \neg r(x)
\end{array} \\
\hline
\begin{array}{c}
r(a) \quad \neg r(a)
\end{array}
\end{array}
\]

Input resolution: in each resolution step one of the two parent clauses must be from the input. In our example:

\[
\begin{array}{c}
\frac{-s(x) \lor \neg r(x) \quad s(a)}{r(a) \lor \neg p(x) \\
\frac{-r(a)}{p(a)}
\end{array}
\]

Linear resolution: in each resolution step one of the two parent clauses must either be from the input or must be a successor of the other parent clause.

Theorem 1.47 (Completeness of resolution variants)

Linear resolution is refutation complete.

Input resolution is correct but not refutation complete.

Idea:

Maybe input resolution is complete for a restricted class of formulae.

Definition 1.48 (Horn clause)

A clause is called Horn clause if it contains at most one positive atom.

A Horn clause is called definite if it contains exactly one positive atom. It has the form

\[
A(t) \leftarrow A_1(t_1), \ldots, A_n(t_n).
\]

A Horn clause without positive atom is called query:

\[
\square \leftarrow A_1(t_1), \ldots, A_n(t_n).
\]

Often, one also calls the body of the above rule query \( Q: \square \leftarrow Q(x_1, \ldots, x_n) \).
Theorem 1.49 (Input resolution for Horn clauses)

Input resolution for Horn clauses is refutation complete.

- From now on, we call a finite set of definite Horn clauses a definite logic program or just definite program.
- Each definite program $P$ defines a language $L_P$, determined by all symbols occurring in $P$. We denote the Herbrand base of this language by $B_P$.

Definition 1.50 (SLD resolution wrt $P$ and query $Q$)

SLD resolution with respect to a definite program $P$ and the query $Q$ is input resolution beginning with the query $\square \leftarrow A_1, \ldots, A_n$.

- Then one $A_i$ is chosen and resolved with a clause of the program.
- A new query emerges, which is treated as before.
- If the empty clause $\square \leftarrow$ can be derived, then SLD resolution was successful and the instantiation of the variables is called computed answer.

Theorem 1.51 (Correctness of SLD resolution)

Let $P$ be a definite program and $Q$ a query. Then each computed answer (for $P$ wrt $Q$) is correct.

Example 1.53 (SLD-Resolution)

Let the definite program $P_{SLD}$ consist of the following three clauses

(1) $p(x, z) \leftarrow q(x, y), p(y, z)$
(2) $p(x, x)$
(3) $q(a, b)$

The query $Q$ we are interested in is given by $p(x, b)$. I.e. we are looking for all substitutions $\Theta$ for $x$ such that $p(x, b)\Theta$ follows from $P$.

Definition 1.52 (Computation rule)

A computation rule $R$ is a function which assigns an atom $A_i \in \{A_1, \ldots, A_n\}$ to each query $\square \leftarrow A_1, \ldots, A_n$. This $A_i$ is the chosen atom against which we will resolve in the next step.

Note:

PROLOG always uses the leftmost atom.
A SLD tree may have three different kinds of branches:

1. **infinite ones**,  
2. branches ending with the empty clause (and leading to an answer) and  
3. **failing branches** (dead ends).

**Theorem 1.54 (Independence of computation rule)**

Let $R$ be a computation rule and $\sigma$ an answer calculated wrt $R$ (i.e. there is a successful SLD resolution). Then there is a successful SLD resolution for each other computation rule $R'$ and the answer $\sigma'$ belonging to $R'$ is a variant of $\sigma$. 
Theorem 1.55 (Completeness of SLD resolution)

Each correct answer substitution is subsumed by a calculated answer substitution. I.e.:

\[ P \models \forall Q \Theta \]

implies

\[ \exists \sigma : Q \sigma = Q \Theta \]

SLD computes an answer \( \tau \) with:

\[ D_{\sigma} : Q_{\tau \sigma} \]

---

Question:

How to find successful branches in a SLD tree?

Definition 1.56 (Search rule)

A search rule is a strategy to search for successful branches in SLD trees.

Note:

PROLOG uses depth-first-search.

A SLD resolution is determined by a computation rule and a search rule.

---

SLD trees for \( P \cup \{ Q \} \) are determined by the computation rule.

PROLOG is incomplete because of two reasons:

- depth-first-search
- incorrect unification (no occur check).

A third reason comes up if we also ask for finite and failed SLD resolutions:

- the computation rule must be fair, i.e. it must be guaranteed that every atom on the list of goals is eventually chosen.
Are definite programs too restricted wrt. expressivity?

No! They are Turing complete: all r.e. sets can be expressed by definite programs. To be more precise: let a language with at least one constant, one unary function symbol and a binary relation symbol be given. We also assume there is a distinguished unary relation symbol $p$. Then the Herbrand universe is infinite and for each r.e. set $M$ of this universe there is a definite program $P$ such that

$$M = \{ p(t) : P \models p(t) \}.$$  

Order of atoms

Example 1.57 (Termination depends on the order)

Consider the following two programs:

1. $\text{reverse}(X|Y,Z) : \neg \text{append}(U,[X],Z), \text{reverse}(Y,U)$
2. $\text{reverse}(X|Y,Z) : \neg \text{reverse}(Y,U), \text{append}(U,[X],Z)$

together with a definition for $\text{append}$

$$\text{append}([],X,X) : \leftarrow$$
$$\text{append}([X|Y],Z,[X|T]) : \neg \text{append}(Y,Z,T)$$

and the query “$Q = \text{reverse}([a|X],[b,c,d,b])$”.

(1) $Q' = \text{append}(U,[a],[b,c,d,b]), \text{reverse}(X,U)$
(2) $Q' = \text{reverse}(X,U), \text{append}(U,[a],[b,c,d,b])$
1. Basics about Logic

Example 1.58 ($T_p$)

Given a definite program $P$ let $T_p$: $2^{B_P} \leftarrow 2^{B_P}; I \leftarrow T_p(I)$

$T_p(I) := \{ A \in B_P : \text{there is an instantiation of a rule in } P \text{ s.t. } A \text{ is the head of this rule and all body-atoms are contained in } I \}$

It turns out that $T_p$ is monotone and continuous: so that

Theorem 1.59 ($T_p$ and $M_p$)

$M_p = T_p \upharpoonright \omega = lfp(T_p)$.

An important feature of SLD-Resolution is its Goal-Orientedness, also called Relevance.

Lemma 1.61 (Goal-Orientedness, Relevance)

Given a program $P$ and a query $Q(x)$, only the call graph below $Q$ (i.e. the relevant part of $P$ wrt. $Q$) is necessary to answer this query.

Theorem 1.60 (Soundness and Completeness of SLD)

The following properties are equivalent for a definite program $P$ and a query $Q$:

- $P \models \forall Q\Theta$, i.e. $\forall Q\Theta$ is true in all models of $P$,
- $M_P \models \forall Q\Theta$,
- SLD computes an answer $\tau$ that subsumes $\Theta$ wrt $Q$ (i.e. $\exists \sigma : Q\tau\sigma = Q\Theta$).

Note that not just some correct answer is computed: the most general one is.

Principle 1.1 (PROLOG Paradigm)

Given a definite program $P$ and a query, the proofs of this query (and the computed answers) represent the solutions of the formalized problem.

It is important that function symbols be used. Given a query, only the relevant part of the program is considered.
1. Basics about Logic

Programming versus knowledge engineering programming

<table>
<thead>
<tr>
<th>choose language</th>
<th>choose logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>write program</td>
<td>define knowledge base</td>
</tr>
<tr>
<td>write compiler</td>
<td>implement calculus</td>
</tr>
<tr>
<td>run program</td>
<td>derive new facts</td>
</tr>
</tbody>
</table>

7. References

Logic programming and negation: A survey.

Knowledge representation with extended logic programs.

Nonmonotonic Reasoning: An Overview., volume 73.

[Dastani et al., 2007] Mehdi Dastani, Jürgen Dix and Peter Novak.
The Second Contest on Multi-Agent Systems based on, Computational Logic.
2. Answer Set Programming (ASP)

- In the previous lecture we have introduced SLD resolution as an efficient procedural mechanism for Horn programs. A nice property was Relevance and the fact, that a unique Herbrand model exists.

- Starting with Slide 142 we discuss some famous important problems in knowledge representation and present several motivating running examples.

- Although Horn programs are Turing complete, they are too restricted as a framework for KR. We motivate to use programs with negation as an appropriate tool to formalize problems. But how should the semantics look like?

- We show (starting with Slide 156) that by assuming a few interesting properties (Red, GPPE, Sub, TAUT, CONTRA), only one canonical semantics survives: answer sets. This leads to the ASP paradigm: representing problems by logic programs with negation under the ASP semantics.

- In contrast to PROLOG, ASP is purely declarative and uses efficient database techniques. The declarativeness of ASP can be shown on the greatest common divisor example.

- A program with negation determines a set of sets: any NP-problem (resp. $\Sigma_2$-problem) can be uniformly represented.

- There exist many efficient implementations of ASP.
The main idea of PROLOG was to describe the given problem in logic and leave its solution to the underlying resolution procedure.

**Example 2.1 (Greatest Common Divisor (1), see also Slide 186)**

Given two integers \( n, m \), how can we compute the greatest common divisor with PROLOG? An answer is to apply Euclid's algorithm and to write down a recursive definition for the predicate

\[
gcd(T, X, Y) : T \text{ is the greatest common divisor of } X, Y.
\]

\[
gcd(X, X, X) :- \#int(X), X > 0.
\]

\[
gcd(T, X, Y) :- X < Y, gcd(T, X, Y1), Y = Y1 + X.
\]

\[
gcd(T, X, Y) :- X > Y, gcd(T, X1, Y), X = X1 + Y.
\]

This is not declarative: It assumes a lot of mathematical insight. Namely an algorithm and the fact that it is correct and complete.

Could we also just describe the properties, without explicitly giving an algorithm?

**Important problems of knowledge representation**

There are three very important representation-problems concerning the axiomatization of a changing world:

**Frame problem:** most actions change only little – we need many actions to describe invariant properties. It would be ideal to axiomatize only what does not change and to add a proposition like “nothing else changes”.

**Ramification problem:** How should we handle implicit consequences of actions? For example \( \text{Grab}(\text{Gold}) \): \text{Gold} can be contaminated. Then \( \text{Grab}(\text{Gold}) \) is not optimal.
Qualification problem: In logic, it is necessary to enumerate all conditions under which an action is executed successfully.

\[
\forall x \ (Bird(x) \land \neg Penguin(x) \land \neg Dead(x) \land
\neg Ostrich(x) \land \neg BrokenWing(x) \land
...)
\rightarrow Flies(x)
\]

It would be ideal to say birds normally fly.

The most natural way is to use “not”
\[
\phi \leftarrow \psi, \ not \ ab
\]
where \(ab\) stands for abnormality. Obviously, this forces us to extend definite programs by negative atoms.

Example 2.2 (Inheritance Hierarchies)

Suppose we know that birds typically fly and penguins are non-flying birds. We also know that Tweety is a bird. Now an agent is hired to build a cage for Tweety.

Should the agent put a roof on the cage?

After all it could be still the case that Tweety is a penguin and therefore can not fly, in which case we would not like to pay for the unnecessary roof. But under normal conditions, it is obvious that one should conclude that Tweety is flying.

A natural axiomatization is given as follows:

\[
P_{\text{Inheritance}}: 
flies(x) \leftarrow bird(x), \ not \ ab(r_1, x)
bird(x) \leftarrow penguin(x)
ab(r_1, x) \leftarrow penguin(x)
make\_top(x) \leftarrow flies(x)
\]

together with some particular facts, like

\(\text{bird(Tweety)}\) and \(\text{penguin(Sam)}\).

For the query “make\_top(Tweety)” we expect the answer “yes” while for “make\_top(Sam)” we expect the answer “no”.

Nonmonotonicity

The last example shows that we need a nonmonotonic logic:

- From \(T\) should follow \(\text{make\_top(Tweety)}\).
- From \(T \cup \{\text{penguin(Tweety)}\}\) should not follow \(\text{make\_top(Tweety)}\).
- Is SLD a nonmonotonic mechanism?
Example 2.3 (The Transitive Closure)

Assume we are given a graph consisting of nodes and edges between some of them. We want to know which nodes are reachable from a given one. A natural formalization of the property “reachable” would be

\[
\text{reachable}(x) \leftarrow \text{edge}(x,y), \text{reachable}(y).
\]

What happens if we are given the following facts

\[
\text{edge}(a,b), \text{edge}(b,a), \text{edge}(c,d)
\]

and reachable(c)?

We expect that neither a nor b are reachable because there is no path from c to either a or b.

Example 2.4 (Van Gelder’s Example)

Assume we are describing a two-players game like checkers. The two players alternately move a stone on a board. The moving player wins when his opponent has no more move to make. This is formalized by

\[
\text{wins}(x) \leftarrow \text{move_from_to}(x,y), \neg \text{wins}(y)
\]

meaning that

- the situation x is won (for the moving player A), if he can lead over (with the help of a regular move, given by the relation \text{move_from_to}(.,.) to a situation y that can never be won for B.

Assume we also have the facts

\[
\text{move_from_to}(a,b), \text{move_from_to}(b,a), \text{move_from_to}(b,c), \text{move_from_to}(c,d).
\]

Our query to this program \(P_{\text{game}}\) is \(?-\text{wins}(b)\).
Negation as finite-failure

- We consider a definite program and a query $Q$.
- If the resulting SLD tree is finite with only failing branches, then the query fails.

We consider again Example 1.53 from Slide 116:

1. $p(x, z) \leftarrow q(x, y), p(y, z)$
2. $p(x, x)$
3. $q(a, b)$

and ask the query $\neg p(b, a)$. What do we get?

- Can we extend SLD to a procedure SLDNF?
- Is SLDNF a nonmonotonic mechanism?
2. Answer Set Programming (ASP) 2. Semantics

Formally, we can associate to any semantics SEM in the sense of Definition 2.6 two entailment relations

- **sceptical**: \( \text{SEM}^{\text{scept}}(P) \) is the set of all atoms or default atoms that are true in all models of SEM(P).
- **credulous**: \( \text{SEM}^{\text{cred}}(P) \) is the set of all atoms or default atoms that are true in at least one model of SEM(P).

When there is only one model, we will omit the outer brackets and write (instead of \( \text{SEM}^{\text{scept}}(P) = \{M\} \))

\( \text{SEM}(P) = M \).

We will also slightly abuse notation and write \( l \in \text{SEM}(P) \) as an abbreviation for \( l \in M \) for all \( M \in \text{SEM}(P) \).

2. Answer Set Programming (ASP) 2. Semantics

Here is the first principle we would like to have (which already distinguishes FOL from LP).

**Principle 2.1 (Reduction)**

Suppose we are given a program \( P \) with possibly default-atoms in its body. If a ground atom \( A \) does not unify with any head of the rules of \( P \), then we can delete in every rule any occurrence of “not \( A \)” without changing the semantics.

Dually, if there is an instance of a rule of the form “\( B \leftarrow \)” then we can delete all rules that contain “not \( B \)” in their bodies.

2. Answer Set Programming (ASP) 2. Semantics

Extending SLD to SLDNF

How should we handle default-atoms?

- If we reach “not \( A \)” as a subgoal, we keep the current SLD-tree in mind and start a new SLD-tree by trying to solve “\( A \)”.
- If this succeeds, then we falsified “not \( A \)”, the current branch is failing and we have to backtrack and consider a different subquery.
- But it can also happen that the SLD-tree for “\( A \)” is finite with only failing branches. Then we say that \( A \) **finitely fails**, we turn back to our original SLD-tree, consider the subgoal “not \( A \)” as successfully solved and go on with the next subgoal in the current list.

2. Answer Set Programming (ASP) 2. Semantics

**Principle 2.2 (A “Naive” SLDNF-Resolution)**

If in the construction of an SLDNF-tree a default-atom not \( L_{ij} \) is selected in the list \( L_i = \{L_{i1}, L_{i2}, \ldots\} \), then we try to prove \( L_{ij} \).

If this fails finitely (it fails because the generated subtree is finite and failing), then we take not \( L_{ij} \) as proved and we go on to prove \( L_{i(j+1)} \).

If \( L_{ij} \) succeeds, then not \( L_{ij} \) fails and we have to backtrack to the list \( L_{i-1} \) of preliminary subgoals (the next rule is applied: **backtracking**).

SLDNF properly handles our Example 2.2.
SLDNF answers quite easily our requirements of a semantics SEM (stated explicitly in Definition 2.6).

Lemma 2.7
SLDNF satisfies Reduction.

Up to now it seems that SLDNF-resolution solves all our problems. It handles our examples correctly, and is defined by a procedural calculus strongly related to SLD.

There are two main problems with SLDNF:
- SLDNF can not handle free variables in negative subgoals (floundering),
- SLDNF is still too weak for Knowledge Representation.

The latter problem is the most important one.

When we consider rules of the form \( p \leftarrow p \), then SLD resolution gets into an infinite loop and no answer to the query \(?- p\) can be obtained. This has often the effect that when we enter into negation-as-failure mode, the SLD-tree to be constructed is infinite, although it is not successful and therefore should be considered as failed.

Principle 2.3 (Elimination of Tautologies)
Suppose a program \( P \) has a rule which contains the same atom in its body as well as in its head (i.e. the head consists of exactly this atom). Then we can eliminate this rule without changing the semantics.

What does SLDNF do on Example 2.3?

reachable\((x) \leftarrow edge(x,y), reachable(y)\),
together with the facts

\( edge(a,b), edge(b,a), edge(c,d) \)

and \( reachable(c) \)?

SLDNF-Resolution does not derive \"not reachable(a)\"!
Partial Evaluation

The query "not reach(a)" leads to the rule "reach(a) ← edge(a,b), reach(b)" and "reach(b)" leads to "reach(b) ← edge(b,a), reach(a)". Both rules are definitions for reach(a) and reach(b). So we can replace the body atoms of reach by their definitions:

\[
\begin{align*}
\text{reach(a)} & \leftarrow \text{edge(a,b)}, \text{reach(b)} \\
\text{reach(b)} & \leftarrow \text{edge(b,a)}, \text{edge(a,b)}, \text{reach(b)}
\end{align*}
\]
that can both be eliminated by applying Principle 2.3. So we end up with a program that does neither contain reach(a) nor reach(b) in one of the heads. Therefore, according to Principle 2.1 both atoms should be considered false.

---

Definition 2.8 (GPPE)

A semantics SEM satisfies GPPE if the following transformation does not change SEM:
Replace a rule \(A \leftarrow B^+ \land \neg B^-\) where \(B^+\) contains an atom \(B\) by the rules:

\[
A \leftarrow \left( B^+ \setminus \{B\} \right) \cup B_i^+ \land \neg \left( B^- \cup B_i^- \right) (i = 1, \ldots, n)
\]

where \(B_i^+ \land \neg B_i^- (i = 1, \ldots, n)\) are all rules with head \(B_i\).

\[
\begin{align*}
B & \leftarrow \neg E \\
B & \leftarrow D, \neg C \\
A & \leftarrow B, \neg C 
\end{align*}
\]

\[
A \leftarrow D, \neg C, \neg C \\
A \leftarrow \neg E, \neg C
\]

---

We reconsider Example 2.5. The semantics of SLDNF corresponds to Clark’s completion \(\text{comp}\).

Example 2.9 (COMP vs. KR (revisited))

\[
\begin{align*}
\text{P}_{KR}: & \quad p \leftarrow p \\
& \quad q \leftarrow \neg p \\
\text{q} & \leftarrow \neg p \\
\text{r} & \leftarrow \neg r \\
\text{comp(P}_{KR}): & \quad p = p \\
& \quad q = \neg p \\
& \quad r = \neg r \\
\end{align*}
\]

\[
\begin{align*}
\text{P}'_{KR}: & \quad p \leftarrow p \\
& \quad q \leftarrow \neg p \\
\text{comp(P}'_{KR}): & \quad p = p \\
& \quad q = \neg p \\
& \quad r = \neg r \\
\end{align*}
\]


Note that any semantics SEM satisfying GPPE and Elimination of Tautologies can be seen as extending SLD by doing some Loop-checking. We will call such semantics KR-semantics in order to distinguish them from the classical LP-semantic which are based on SLDNF or variants of Clark’s completion \(\text{comp}(P)\):

\[
\begin{align*}
\text{KR-Semantics} &= \text{SLDNF + Loop-check.}
\end{align*}
\]
The last principle in this section is related to Subsumption: we can get rid of non-minimal rules by simply deleting them.

**Principle 2.4 (Subsumption)**

In a program $P$ we can delete a rule $A \leftarrow B^+ \land \neg B^-$ whenever there is another rule $A \leftarrow B'^+ \land \neg B'^-$ with $B'^+ \subseteq B^+$ and $B'^- \subseteq B^-$.  

\[
\begin{align*}
A & \leftarrow D, \neg E, \neg F \\
A & \leftarrow D, \neg F
\end{align*}
\]

$\Rightarrow$

\[
A \leftarrow D, \neg F
\]

**Wellfounded Semantics: WFS**

We call a semantics $SEM_1$ weaker than $SEM_2$, if for all programs $P$ and all atoms or default-atoms $l$ the following holds: $SEM_1(P) \models l$ implies $SEM_2(P) \models l$. 

I.e. all (default-) atoms derivable from $SEM_1$ with respect to $P$ are also derivable from $SEM_2$.

**Theorem 2.10 (WFS)**

There exists the weakest semantics satisfying our four principles Elimination of Tautologies, Subsumption, Reduction, and GPPE. This semantics is called wellfounded semantics $WFS$.

What about van Gelder’s Example 2.4

- $wins(x) \leftarrow \text{move}\_\text{from}\_\text{to}(x,y), \neg \text{wins}(y)$
- $\text{move}\_\text{from}\_\text{to}(a,b), \text{move}\_\text{from}\_\text{to}(b,a), \text{move}\_\text{from}\_\text{to}(b,c), \text{move}\_\text{from}\_\text{to}(c,d)$.

Here we have no problems with floundering, but using SLDNF we get an infinite sequence of oscillating SLD-trees (none of which finitely fails).

\[
WFS(P_{game}) = \{ \neg wins(c), wins(b), \neg wins(a) \}\]
2. Answer Set Programming (ASP) 2. Semantics

Contradictions

Some semantics associates to a program $P$ a set of 2-valued models. Such semantics satisfy

**Principle 2.5 (Elimination of Contradictions)**

Suppose a program $P$ has a rule which contains the same atom $A$ and $\not A$ in its body. Then we can eliminate this rule without changing the semantics.

Contradiction: $A \leftarrow C, D, \not C$

Theorem 2.12 (Answer Sets)

There exists the weakest semantics satisfying our five principles Elimination of Tautologies, Subsumption, Reduction, GPPE and Elimination of Contradictions. This semantics assigns to each program $P$ a set of answer sets, also called stable models.

Answer Sets

The underlying idea is that any atom in an intended model should have a definite reason to be true or false.

**Gelfond-Lifschitz transformation:** for a program $P$ and a model $N \subseteq B_P$ we define

$P^N := \{\text{rule}^N : \text{rule} \in P\}$

where \text{rule} := $A \leftarrow B_1, \ldots, B_n, \not C_1, \ldots, \not C_m$ is transformed as follows

$(\text{rule})^N := \begin{cases} A \leftarrow B_1, \ldots, B_n, & \text{if } \forall j : C_j \not\in N, \\ t, & \text{otherwise.} \end{cases}$

Note that $P^N$ is always a definite program. We can therefore compute its least Herbrand model $M_{P^N}$ and check whether it coincides with the model $N$ with which we started:

**Definition 2.13 (Answer Sets)**

$N$ is called an answer set$^d$ of $P$, if $M_{P^N} = N$.

$^d$Note that we only consider Herbrand models.
Relationship between ASP and WFS

They are based on similar principles.
- Stable models $N$ extend WFS: $l \in \text{WFS}(P)$ implies $N \models l$.
- If WFS($P$) is two-valued, then WFS($P$) is the unique stable model.
For Example 2.4 we have two stable models: \{wins$(a)$, wins$(c)$\} and \{wins$(b)$, wins$(c)$\} and therefore
\[
\text{WFS}(P) = \{\text{wins}(c), \text{not wins}(d)\} = \bigcap_{N \text{ a stable model of } P} N.
\]

Example 2.14 (Reasoning by cases)

$P_{\text{splitting}}$: $a \leftarrow \text{not } b$  
$b \leftarrow \text{not } a$  
$p \leftarrow a$  
$p \leftarrow b$

Although neither $a$, nor $b$ can be derived in any semantics based on two-valued models (as ASP for example), the disjunction $a \lor b$, thus also $p$, is true. In this way the example is handled by the SLDNF semantics, too. WFS($P$), however, is empty; if the WFS cannot decide between $a$ or $\text{not } a$, then $a$ is undefined.

Example 2.15 (ASP is not Goal-Oriented)

$P_{\text{rel}(a)}$: $a \leftarrow \text{not } b$  
$b \leftarrow \text{not } a$  
$p \leftarrow \text{not } p$  
$p \leftarrow a$

$P_{\text{rel}(a)}$ is the subprogram of $P$ that consists of all rules that are relevant to answer the query $?- a$. It has two stable models \{a\} and \{b\} — $a$ is not true in all of them. But the program $P$ has the unique stable model \{p,a\}, so $a$ is true in all stable models of $P$. 
2. Answer Set Programming (ASP)  3. Properties

The results above also apply to **disjunctive** programs. Some modifications have to be made (the unique Herbrand model of a definite program has to be replaced by the set of all minimal models of a positive disjunctive program). We just state the following

**Theorem 2.16 (Characterization of ASP)**

Let $SEM$ be a semantics for disjunctive logic programs satisfying **GPPE**, **Elimination of Tautologies**, and **Elimination of Contradictions**. Then: $SEM(P) \subseteq ASP(P)$. Moreover, $ASP$ is the weakest semantics satisfying these properties.

---

**Declarativeness**

**Lemma 2.17 (ASP is Declarative)**

In contrast to PROLOG, ASP programs do neither depend on the (1) order of program clauses, nor on the (2) order of literals within each clause.

---

**Killer-clauses**

For quite some time, the problem of ASP to handle **odd loops** was considered a drawback: Programs with rules of the form $p \leftarrow \neg p$ (and where $p$ can not be derived otherwise) do not possess answer sets.

**Lemma 2.18 (Constraints)**

Suppose the program $P$ does not contain the predicate $p$. Then the answer sets of the program $P \cup \{ p \leftarrow \neg p, q_1, \ldots, q_n, \neg r_1, \ldots, \neg r_m \}$ are exactly those of $P$ except the answer sets that contain $\{ q_1, \ldots, q_n \}$ and do not contain $\{ r_1, \ldots, r_m \}$. Thus adding this clause can be seen as a constraint and can be used for efficient computation.

---

**Example 2.19 (3-colorability)**

Given an undirected graph, assign 3 colors to the nodes, such that no adjacent nodes have the same color. Using the predicates $node(x)$, $edge(x,y)$ we can write the following program:

$$
\text{color}(X,r) \lor \text{color}(X,g) \lor \text{color}(X,b) : - \ node(X) \\
\text{edge}(X,Y), \text{col}(X,C), \text{col}(Y,C)
$$

The first rule can also be written as follows (if no disjunction $\lor$ is available):

$$
\text{color}(X,r) : - \ node(X), \neg \text{color}(X,g), \neg \text{color}(X,b) \\
\text{color}(X,b) : - \ node(X), \neg \text{color}(X,r), \neg \text{color}(X,g) \\
\text{color}(X,g) : - \ node(X), \neg \text{color}(X,r), \neg \text{color}(X,b)
$$
Example 2.20 (Hamiltonian Path)
The input is a directed graph represented by predicates
node( · ) (unary), arc( · , · ) (binary) and a starting node
start( · ) (unary). Output should be a path that starts with
the start node and contains all nodes of the graph (each
exactly once).

\[
inPath(X,Y) \lor outPath(X,Y) \leftarrow arc(X,Y) \\
inPath(X,Y), inPath(X,Y), Y \not\succ Y1 \\
inPath(X,Y), inPath(X,Y), X \not\prec X1 \\
node(X), not\ reached(X) \\
reached(X) \leftarrow start(X) \\
reached(X) \leftarrow reached(Y), inPath(Y,X)
\]

The first rule guesses all possibilities, the remaining rules
check whether the guess was correct and a hamiltonian
path was found.

Example 2.21 (Greatest Common Divisor (2), see also
Slide 142)
Given two integers \( n, m \), how can we compute the greatest
common divisor?

\[
\text{divisor}(T,N) \leftarrow \#int(T), \#int(N), \#int(M), N = T \times M. \\
\text{cd}(T,N1,N2) \leftarrow \text{divisor}(T,N1), \text{divisor}(T,N2). \\
\text{cd}(T,N1,N2) \leftarrow \text{cd}(T,N1,N2), \text{cd}(T1,N1,N2), T < T1. \\
\text{gcd}(T,N1,N2) \leftarrow \text{gcd}(T,N1,N2), \text{not}\ larger\_cd(T,N1,N2).
\]

Definition 2.22 (Search Problem)
A search problem \( S \) is a pair \( \langle \text{Inst}, \{\text{Sol}_I : I \in \text{Inst}\} \rangle \) where
\( \text{Inst} \) is an (infinite) set of finite objects, called
instances, and
for each \( I \), \( \text{Sol}_I \) is a finite set (called set of solutions).

An algorithm \( \mathcal{A} \) solves a search problem \( S \), if the
following holds: for each \( I \in \text{Inst} \)
\[
\begin{align*}
&\{ \mathcal{A} \text{ returns } \text{"No"}, \text{ if } \text{Sol}_I = \emptyset; \\
&\mathcal{A} \text{ returns any } S \in \text{Sol}_I, \text{ otherwise.} \}
\end{align*}
\]

Definition 2.23 (NP-Search)
A search problem is in the class \( \text{NP-Search} \), if there is a
nondeterministic Turing machine that runs in polynomial
time and satisfies
for every instance \( I \in \text{Inst} \), exactly \( \text{Sol}_I \) is written on the
tape (when accepting computations for \( I \) terminate).

NP is usually defined as the class of decision problems.
Obviously, all decision problems are contained in
\( \text{NP-Search} \).
Separating Data from the Problem

We would like to separate the search problem itself from the representation of its instances (see Slide 31).

**Definition 2.24 (Uniform Representation)**

A search problem $S$ can be represented uniformly in ASP, if

1. there is a finite program $P$,
2. for each instance $I \in \text{Inst}$ a finite set $M_I$ of ground atoms (and this set can be efficiently encoded)
3. such that $\text{Sol}_I = \text{ASP}(P \cup M_I)$ or the solutions can be efficiently reconstructed from the answer sets of $P \cup M_I$.

**Principle 2.6 (ASP Paradigm)**

The set of all answer sets of a program represents the solution of a problem.

nondisjunctive: The class of problems that can be uniformly represented in ASP is NP-search.

disjunctive: The class of problems that can be uniformly represented in ASP is $\Sigma_2$-search.

Note that there are no function symbols in ASP.

Very important properties of ASP:

**Variables**: although there are no function symbols, variables are allowed (but the grounding is finite),

**Predicates**: also predicates are allowed and facilitate concise formalizations,

**Modularity**: global models should be composed of local components,

**Semantics**: there should be an intuitive methodology to formalise problems.
Many links can be obtained from

  http://wasp.unime.it
  (FP 6: IST-FET 2001-37004)

### Definition 2.25 (AnsProlog\(^{\text{not}}\), AnsProlog\(^{\top}\), AnsProlog\(^{\top,\text{not}}\))

The language AnsProlog\(^{\text{not}}\) consists of rules of the form

\[ A \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n \]

where \( A, B_1, \ldots, B_m, C_1, \ldots, C_n \) are positive atoms which may contain free variables, like \( p(X,Y,c) \). When \( A \) is absent (resp. identical to \( \bot \)): then we call the language AnsProlog\(^{\bot}\).

The language AnsProlog\(^{\top}\) consist of rules of the form

\[ A_1 \lor A_2 \lor \ldots \lor A_n \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n \]

where \( A_1, A_2, C_i \) are positive atoms, which may contain free variables, like \( p(X,Y,c) \). Similar to the above, we define AnsProlog\(^{\top,\bot}\).

Finally the language AnsProlog\(^{\top,\text{not}}\) (resp. AnsProlog\(^{\top,\text{not}},\bot\)) consists of rules where we allow both disjunctions in the head as well as negations in the body.

### Some implementations

- **SModels:** http://www.tcs.hut.fi/Software/smodels/
- **DLV:** http://www.dbai.tuwien.ac.at/proj/dlv/
- **GnT:** http://www.tcs.hut.fi/Software/gnt/
- **Cmodels (1, 2):** http://www.cs.utexas.edu/users/tag/cmodels/
- **ASSAT:** http://assat.cs.ust.hk/

- **aspps:** http://www.cs.engr.uky.edu/ai/aspps/
- **NoMore:** http://www.cs.uni-potsdam.de/linke/nomore/
- **ccalc:** http://www.cs.utexas.edu/users/tag/cc/
- **XASP:** distributed with XSB v2.6
  http://xsb.sourceforge.net
Disjunction: DLV is designed for full AnsProlog \( \lor, \neg, \bot \), while smodels is designed for AnsProlog \( \neg, \bot \). smodels has only primitive functionality for \( \lor \).

Grounding: Both systems compute intelligent groundings, trying to avoid unnecessary instances.

Relational DB: DLV can be seen as an extension to SQL3 and thus has functionality for answering SQL3 queries.

Queries: DLV allows brave and cautious reasoning: queries can be specified and tested for truth in in at least one or in all answer sets.

Allowedness: In smodels each variable in a rule must occur in a positive domain predicate on the right hand side of this rule. A domain predicate is one with the following property: each path in the dependency graph of the program starting with this predicate does not go through a negative cycle. This property is also called strongly range restricted. The idea is that domain predicates can be efficiently computed (no recursion through negation). In DLV this is more relaxed: each variable must occur in a positive predicate on the right hand side.

Special Constraints: smodels allows weight and cardinality constraints, while DLV allows weak constraints.

Arithmetic: smodels allows rules of the form
\[
p(T + 1) \leftarrow p(T).
\]
In DLV this must be written as
\[
p(T') \leftarrow p(T), T' = T + 1.
\]

Classical Negation: In our definition of an answer set (Definition 2.13) and also in the definition of AnsProlog \( \lor, \neg, \bot \), we did not allow atoms that are classically negated. In fact, in several formalisations we used predicates of the form \( \neg \_ \_predicate \) which, intuitively represented the negation of the predicate \( \_ \_predicate \). We did this mainly to avoid any confusion with classical negation.
Cardinality Constraints: smodels allows cardinality constraints to ensure that an answer set contains at least and at most a certain number of prespecified atoms.

1 \{a, b, not c\} 2

This means that we are looking for answer sets which contain at least one but at most two of the atoms a, b, not c.

Formalizing Sudoku

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).

The theory

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

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. because of the Principle 2.1 that holds for ASP.

More constructs in smodels

1 \{ p(X, Y, A) : val(A) \} 1

\[ :- row(X), col(Y) \]

this makes sure that in all entries of the grid, exactly one number (\( \text{val}() \)) is contained.

1 \{ p(X, Y, A) : row(X) : col(Y) : eq(div(X, 3), div(R, 3)) : eq(div(Y, 3), div(C, 3)) \} 1

\[ :- val(A), row(R), col(C) \]

this rule ensures that in each of the 9 squares each number from 1 to 9 occurs only once.
Sudoku formalization

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

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)
    : \text{eq}(\text{div}(X,3),\text{div}(R,3)) : \text{eq}(\text{div}(Y,3),\text{div}(C,3)) \} 1
    :- \text{row}(R), \text{col}(C), \text{val}(A).\]

References

[Brass and Dix, 1999] Stefan Brass and Jürgen Dix.
Semantics of (disjunctive) logic programs based on partial evaluation.

Knowledge representation with extended logic programs.

A logic programming approach to knowledge-state planning, ii: The dlvK system.

Stable models and an alternative logic programming paradigm.

[Niemelä, 1999] Ilkka Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
2. Answer Set Programming (ASP)


5. References

3. Modal Logic

Chapter 3. Modal Logic

- Modal Logic
  - 3.1 Reasoning about Knowledge
  - 3.2 Kripke Semantics
  - 3.3 Reasoning about Muddy Children
  - 3.4 Axioms for Modal Logics
  - 3.5 References

The first two lectures showed how classical logic and a nonmonotonic version of logic programming can be used to describe/specify problems and their (desired) solutions.

Rephrasing slightly the above: we saw how (nonmonotonic) logic can be used to describe what we know about the world. The main intuition is that what we know is described by a closed set of classical formulae (in other words, by a theory).

Logic programs are used to represent these theories in a compact way. The actual knowledge state represented by program $P$ is, by definition, the set of answer sets of $P$.

We have also seen how these descriptions can be set to work, i.e., used to compute actual models that satisfy our needs.

In this lecture, we present another logic-based tradition of modeling the world (and reasoning about it), namely modal logic. We focus especially on epistemic logic, i.e., the modal logic of knowledge.
3. Modal Logic

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

- □p: p is necessarily true
- ◊p: p is possibly true

Independently of the precise definition:

◊p ↔ ¬□¬p

Definition 3.1 (Modal Logic with n modalities)

The language $L_{\text{modal}}$ of modal logic with n modal operators □$_1$, ..., □$_n$ is the smallest set containing the propositional constants of $L$, and with formulae $\phi, \psi$ also the formulae □$_i$φ, ¬φ, φ ∧ ψ. We treat ∨, →, ↔, ◊ as macros (defined as usual).

Note that the □ operators can be nested:

(□$_1$□$_2$□$_1$p) ∨ □$_3$¬p

Modal logic can be translated to classical logic;

- ... but it looks horribly UGLY then;
- ... and in most cases it’s not automatizable any more.

Remember Frege’s notation of predicate logic:

\[
\begin{align*}
A - n < b \\
\gamma \beta (0 \gamma + \Gamma = b_p) \\
b < B \\
n > 0 \\
A \geq b \\
\gamma \beta (0 + \Gamma = b_p) \\
b < B
\end{align*}
\]

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
3. Modal Logic

1. Reasoning about Knowledge

3.1 Reasoning about Knowledge

Example 3.2 (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 3.3 (Partition Model)

An $n$-agent partition model over language $\mathcal{L}$ is a tuple

$$\langle \mathcal{W}, I_1, \ldots, I_n \rangle,$$

where

- $\mathcal{W}$: a set of possible worlds;

- $w \in \mathcal{W}$: an $\mathcal{L}$-structure (each $\mathcal{L}$ sentence $\varphi$ is either true or false in $w$);

- $I_i$: each $I_i$ is a partition of $\mathcal{W}$:

  $$I_i = \{W_{i1}, W_{i2}, \ldots, W_{ik}\}$$
  with $W_{ij} \cap W_{ik} = \emptyset$ for $j \neq k$ and $\bigcup_{1 \leq j \leq k} W_{ij} = \mathcal{W}$.

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

- \( I_i(w) \): all worlds in partition \( I_i \) containing world \( w \)

\[
I_i(w) := \{ w' : w \in W_i \text{ and } w' \in W_j \}
\]

Example: Robots and Carriage

We introduce new operators \( K_i \):

- \( K_i \phi \): “agent \( i \) knows that \( \phi \) holds”

How can we formalise the notion of one agent knowing something? And reason about what agents know and draw conclusions?
Definition 3.4 (Semantics for partition models)

Let $\mathcal{A} = (\mathcal{W}, I_1, \ldots, I_n)$ be an $n$-agent partition model over $\mathcal{L}$.

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

A slight generalisation of partition models leads to Kripke semantics.

How can one look at a partition model? It is like a set of equivalence classes: for agent $i$, all worlds in one partition are equivalent.

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 $w_1$. 

Example: Robots and Carriage

pos$_1$ \to K$_1$pos$_1$
pos$_2$ \to \neg K$_1$pos$_2$
pos$_2$ \to K$_2$K$_1$\neg pos$_1$
pos$_2$ \to \neg K$_1$K$_2$K$_1$\neg pos$_1$
3. Modal Logic

2. Kripke Semantics

**Definition 3.5 (Kripke Structure)**

A Kripke structure \((\mathcal{W}, R)\) is a set of possible worlds \(\mathcal{W}\) plus a binary relation \(R \subseteq \mathcal{W} \times \mathcal{W}\) (the accessibility relation).

Note that:
- Elements of \(\mathcal{W}\) are now abstract entities: we assume that they do not carry any internal structure!
- Still, we would like to see them as classical (e.g., propositional) models. How can that be achieved?
- Let \(\Pi\) be the set of all propositional symbols. A propositional model can be represented as a list of propositions \(\pi(\mathcal{w}) \subseteq \Pi\) that hold in world \(\mathcal{w}\).

**Definition 3.6 (Kripke Model / Possible World Model)**

The truth of formulae is evaluated with respect to a Kripke model (possible world model):

\[
(\mathcal{W}, R, \pi),
\]

that is, a Kripke structure plus a valuation of propositions \(\pi: \mathcal{W} \rightarrow \mathcal{P}(\Pi)\).

**Definition 3.7 (Kripke Semantics of Modal Logic)**

Given a Kripke model \(M = (\mathcal{W}, R, \pi)\), and a world \(\mathcal{w} \in \mathcal{W}\), we define the satisfaction relation \(M, \mathcal{w} \models \phi\) as follows:
- \(M, \mathcal{w} \models p\) iff \(p \in \pi(\mathcal{w})\)
- \(M, \mathcal{w} \models \phi \land \psi\) iff \(M, \mathcal{w} \models \phi\) and \(M, \mathcal{w} \models \psi\)
- \(M, \mathcal{w} \models \neg \phi\) iff not \(M, \mathcal{w} \models \phi\)
- \(M, \mathcal{w} \models \square \phi\) iff for every \(\mathcal{w}' \in \mathcal{W}\) with \(\mathcal{w}R\mathcal{w}'\) we have that \(M, \mathcal{w}' \models \phi\).

~~What if we want multiple modalities \(\square_1, \ldots, \square_k\)~?

Then, we need multiple accessibility relations \(R_1, \ldots, R_k \subseteq \mathcal{W} \times \mathcal{W}\), one per modality.
Definition 3.8 (Kripke Semantics of Multi-modal Logic)

Given a Kripke model $M = \langle W, R_1, \ldots, R_k, \pi \rangle$, and a world $w \in W$, we define:

- $M, w \models \square_i \varphi$ iff for every $w' \in W$ with $wR_iw'$ we have that $M, w' \models \varphi$.

- For knowledge modalities $K_i$, we assume that the corresponding relation $R_i$ is an equivalence.

- Note: $K_i$ is a “modal box” operator!

Example: Robots and Carriage

3.3 Reasoning about Muddy Children
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 3.9 (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

**Necessitation** \[ \frac{\varphi}{\Box \varphi} \]

**Theorem 3.10 (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.
3. Modal Logic

4. Axioms for Modal Logics

- Note that we have not assumed any properties of the accessibility relation $R$: it is just any binary relation.

- Assuming that $R$ is an equivalence relation, which additional statements (axioms) are true in all Kripke models?

3. Modal Logic

4. Axioms for Modal Logics

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 $x, y, z$.

3. Modal Logic

4. Axioms for Modal Logics

Lemma 3.11

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

3. Modal Logic

4. Axioms forModal Logics

**Definition 3.12 (Extending K by Axioms D, T, 4, 5)**

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

- $K$ \((K_i \phi \land K_i(\phi \to \psi)) \to K_i \psi\) [logical omniscience]
- $D$ \(\neg K_i(\phi \land \neg \phi)\) [consistency]
- $T$ \(K_i \phi \to \phi\) [truth]
- $4$ \(K_i \phi \to K_i K_i \phi\) [positive introspection]
- $5$ \(\neg K_i \phi \to K_i \neg K_i \phi\) [negative introspection]

The system consisting of $KT45$ is also called $S5$. 
Theorem 3.13 (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 structures the accessibility relation of which satisfies $X$.

Corollary 3.14 (Sound-, completeness of KT45 (S5))
System KT45 is sound and complete with respect to Kripke structures with equivalence accessibility relations.

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 $\Box_i$) 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.

3.5 References
3. Modal Logic

Deontic logic.
In D. M. Gabbay and F. Guenther (Eds.), Handbook of

Reasoning about Knowledge.
MIT Press: Cambridge, MA.

FIPA home page.
http://www.fipa.org/.

3. Modal Logic

Semantical analysis of modal logic I. Normal propositional
calculi.
Zeitschrift fur math. Logik und Grundlagen der Mathematik 9,
67–96.

Semantical considerations on modal logic.

Semantical analysis of modal logic II. Non-normal modal
propositional calculi.
In Addison, Henkin, and Tarski (Eds.), The theory of models,
Amsterdam, North-Holland, pp. 206–220.

5. References

FIPA ACL message structure specification.
Approved for standard, 2002-12-06.
http://www.fipa.org/specs/fipa00061/.

[HiDBoHoMe98] Hindriks, K. V., F. S. de Boer, H. Hoek, and J.-J. C.
Formal semantics of the core of AGENT-0.
In ECAI’98 Workshop on Practical Reasoning and Rationality, pp.
20–29.
In the previous lecture, we presented the basic idea of modal logic, and showed that it can be used to reason about agents’ knowledge in a natural way.

Now, we show that modal logic is a generic framework that can be used to reason about other dimensions of systems, too.

In particular, we show how one can use modal logic to reason about time, action, and strategic abilities of agents.

First, we present an overview of Propositional Dynamic Logic, which can be used for reasoning about actions and their outcomes.

Then, we switch to temporal logic, where actions are not mentioned explicitly, but instead one can use them to reason about evolution of systems over a longer (possibly infinite) timeline. We present two most popular variants of temporal logic: linear time logic LTL and branching time logic CTL.

Finally, we briefly present the logic of ATL which extends branching-time temporal logic with the game theoretical notion of a strategy.

Modal logic is a generic framework.

Various modal logics:
- knowledge $\leadsto$ epistemic logic,
- beliefs $\leadsto$ doxastic logic,
- obligations $\leadsto$ deontic logic,
- actions $\leadsto$ dynamic logic,
- time $\leadsto$ temporal logic,
- ability $\leadsto$ strategic logic,
- and combinations of the above

Until now:
- Several operators $K_i$, each defines an epistemic/doxastic relation on worlds.
- Description of static systems: no possibility of change

But:
- computational systems are dynamic!
**4. Action and Time**

**1. Dynamic Logic**

### 4.1 Dynamic Logic

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

2**nd idea:** We need statements about the outcome of actions:

- $[\alpha] \varphi$: “after every execution of $\alpha$, $\varphi$ holds,
- $\langle \alpha \rangle \varphi$: “after some executions of $\alpha$, $\varphi$ holds.

As usual, $\langle \alpha \rangle \varphi \equiv [\alpha] - \varphi$.

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

$[\alpha; \beta] \varphi$

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

---

**Definition 4.1 (Labelled Transition System)**

A labelled transition system is a pair

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

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

$$ \alpha \in Q \times Q.$$

**Definition 4.2 (Dynamic Logic: Models)**

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

1. Dynamic Logic

**Definition 4.3 (Semantics of DL)**

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

**Definition 4.4 (Composite labels)**

The set of labels forms a Kleene algebra \( \langle L, ;, \cup, * \rangle \). In addition, 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:

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

\[\text{if } \varphi \text{ then } a \text{ else } b \quad (\varphi; a) \cup (\neg \varphi; b)\]

\[\text{while } \varphi \text{ do } a \quad (\varphi; a)^* ; \neg \varphi\]

c\text{a\text{R} } \rightarrow \left[(\text{nofuel}; \text{fuel}) \cup (\text{fuelOK}; \text{load})\right]\text{c\text{a\text{R}}}
4. Action and Time

2. Temporal Logic

4.2 Temporal Logic

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>◯φ</td>
<td>φ is true in the next moment in time</td>
</tr>
<tr>
<td>□φ</td>
<td>φ is true in all future moments</td>
</tr>
<tr>
<td>◊φ</td>
<td>φ is true in some future moment</td>
</tr>
<tr>
<td>φ U ψ</td>
<td>φ is true until the moment when ψ becomes true</td>
</tr>
</tbody>
</table>

send(msg, rcvr) → ◊receive(msg, rcvr)

Temporal logic was originally developed in order to represent tense in natural language.

Within Computer Science, it has a significant role in the formal specification and verification of concurrent and distributed systems.

Much of this popularity has been achieved as a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.

- safety properties
- liveness properties
- fairness properties
4. Action and Time

2. Temporal Logic

**Safety:**

“something bad will not happen”

“something good will always hold”

Typical examples:

- □¬bankrupt
- □¬(fuelOK ∨ ◇fuelOK)
  and so on . . .

Usually: □¬....

**Liveness:**

“something good will happen”

Typical examples:

- ◇rich
- rocketLondon → ◇rocketParis
  and so on . . .

Usually: ◇....

Combinations of safety and liveness possible:

- ◇□rich
- □◇rich  →  fairness

**Strong fairness:**

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

Typical examples:

- □(attempt → ◇success)
- ◇attempt → □◇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 4.5 (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
- $\lambda[i...j]$: all time moments between $i$ and $j$
- $\lambda[i...\infty]$: all timepoints from $i$ on
3. Linear Time Logic

Important: computational vs. behavioral structure

System

Computational str.

Behavioral str.

Jürgen Dix and Wojtek Jamroga  
ESSLLI (Dublin 2007)  281/456

Definition 4.6 (Semantics of LTL)

\[
\begin{align*}
\lambda & : \models p \quad \text{iff } p \text{ is true at moment } \lambda[0]; \\
\lambda & : \models \Diamond \varphi \quad \text{iff } \lambda[i..\infty] \models \varphi; \\
\lambda & : \models \Box \varphi \quad \text{iff } \lambda[i..\infty] \models \varphi \text{ for all } i \geq 0; \\
\lambda & : \models \varphi \mathbin{\mathcal{U}} \psi \quad \text{iff } \lambda[i..\infty] \models \psi \text{ for some } i \geq 0, \text{ and } \\
& \quad \lambda[j..\infty] \models \varphi \text{ for all } 0 \leq j \leq i.
\end{align*}
\]

Note that:

\[
\begin{align*}
\Box \varphi & \equiv \neg \Diamond \neg \varphi \\
\Diamond \varphi & \equiv \neg \Box \neg \varphi \\
\Diamond \varphi & \equiv \top \mathbin{\mathcal{U}} \varphi
\end{align*}
\]

Jürgen Dix and Wojtek Jamroga  
ESSLLI (Dublin 2007)  284/456
4. Action and Time

3. Linear Time Logic

4. Computation Tree Logic

4.4 Computation Tree Logic

Branching Time: CTL

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

**Path quantifiers:** A (for all paths), E (there is a path);

**Temporal operators:** ⊗ (nexttime), ◊ (sometime), □ (always) and 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.

---

**Definition 4.7 (CTL models: transition systems)**

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, transition relation is just a modal accessibility relation.

---

**Important: computational vs. behavioral structure**

**Computational str.**

**Behavioral str.**

CTL models are defined as computational structures!

**Definition 4.8 (Paths in a model)**

A path \( \lambda \) is a sequence of states.

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.
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 4.9 (Semantics of CTL*: state formulae)

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

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

Definition 4.10 (Semantics of CTL*: path formulae)

Exactly like LTL!

$M, \lambda \models 0\varphi$ iff $M, \lambda[1\ldots\infty] \models \varphi$;

$M, \lambda \models \varphi U \psi$ iff $M, \lambda[i\ldots\infty] \models \psi$ for some $i \geq 0$, and $M, \lambda[j\ldots\infty] \models \varphi$ for all $0 \leq j < i$. 

$M, \lambda \models E\diamond \varphi$ iff $M, \lambda \models \varphi$ for some $\lambda$.

$M, \lambda \models A\Box (\varphi \lor \psi)$ iff $M, \lambda \models \varphi$ and $M, \lambda \models \psi$ for all $\lambda$. 

$M, \lambda \models roL \rightarrow E\diamond roP$ 

$M, \lambda \models A\Box (roP \rightarrow nofuel)$
4. Action and Time

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

4. Action and Time

ATL: What Agents Can Achieve

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

\[ \langle A \rangle \Phi: \text{coalition } A \text{ has a collective strategy to enforce } \Phi \]

4. Action and Time

ATL: What Agents Can Achieve

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

\[ \langle A \rangle \Phi: \text{coalition } A \text{ has a collective strategy to enforce } \Phi \]

4. Action and Time

4.5 ATL

- \[ \langle jamesbond \rangle \diamond \text{win}: \]
  “James Bond has an infallible plan to eventually win”

- \[ \langle jamesbond, bondsgirl \rangle \text{fun } \cup \text{shot}: \]
  “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;
ATL Models: Concurrent Game Structures

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

Definition 4.11 (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
- $d : \text{Agt} \times Q \to \mathcal{P}(\text{Act})$ 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

Example: Robots and Carriage

Definition 4.12 (Strategy)
A strategy is a conditional plan. We represent strategies by functions $s_a : Q \to \text{Act}$. 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.
**Definition 4.13 (Semantics of ATL)**

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


We have seen in the first lecture that although logic itself is static, it can be made dynamic with the situation calculus: terms denote situations.

Now we introduce the machinery of belief revision. Again, we take an abstract point of view and investigate which properties should (or not) be satisfied for a revision operator \( \ast \): the AGM approach.

While belief revision is suitable for changes in a static world, it is not for describing dynamically changing worlds: Updating is based on different principles.

Both belief revision and updating are not well suited for nonmonotonic logics. Therefore we present a method for updates of logic programs and a language to express them.

### The AGM approach

**AGM**: Alchourron/Gärdenfors/Makinson.

*Seminal paper in 1985: [Alchourron et al., 1985].*

**Given**: A (propositional or first-order) theory \( K \), and some new information \( \phi \).

**Wanted**: A revision operator \( \ast \) that revises a set of beliefs \( K \) in the light of new information \( \phi \).

**Problem**: If \( K \models \phi \), then we have no problems:

\[
K \ast \phi = Cn(K).
\]

If \( \phi \) is consistent with \( K \), then we can simply define \( K \ast \phi = Cn(K \cup \{ \phi \}) \).

**What to do when \( K \models \neg \phi \)?**

---

**Example 5.1 (Swans: white or black?)**

\( \alpha \): All European swans are white.

\( \beta \): The bird caught in the trap is a swan.

\( \gamma \): The bird caught in the trap comes from Sweden.

\( \delta \): Sweden is part of Europe.

\( \epsilon \): The bird caught in the trap is white.

\( \epsilon \) can be derived from the rest.

Now we discover that the bird caught in the trap is black!

This means we want to revise \( \{ \alpha, \beta, \gamma, \delta \} \) by \( \neg \epsilon \).
5. Knowledge in Flux

1. Revising or Updating Beliefs?

Shall we give up $\alpha$ but keep some of its consequences?

- All European swans except the one caught in the trap are white.
- All European swans except some of the swedish are white.

Some assumptions

**Closure:** Belief sets $K$ are logically closed:
$$K = Cn(K).$$

**Expansion:** Add a sentence $\phi$ to $K$.
$$K + \phi := \{ \psi : K \cup \{ \phi \} \models \psi \}.$$

**Contraction:** Remove a sentence $\phi$ from $K$.
$$K \vdash \phi: \text{difficult! Principle of economy: no belief should be given up unnecessarily.}$$

**Revision:** Revise $K$ by $\phi$. We assume that for each $K$ and $\phi$ there is a unique $K\vdash\phi$.

Axioms about $\ast$

1. $Cn(K \ast \phi) = K \ast \phi$
2. $\phi \in K \ast \phi$
3. $K \ast \phi \subseteq K + \phi$
4. If $-\phi \notin K$ then $K + \phi \subseteq K \ast \phi$.
5. $K \ast \phi$ is inconsistent if and only if $-\phi$ is a tautology.
6. If $\models \phi \iff \psi$ then $K \ast \phi = K \ast \psi$.
7. $K \ast (\phi \land \psi) \subseteq (K \ast \phi) + \psi$
8. If $-\psi \notin K \ast \phi$, then
$$\models (K \ast \phi) + \psi \subseteq K \ast (\phi \land \psi).$$

Axioms about $\vdash$

1. $Cn(K \vdash \phi) = K \vdash \phi$
2. $K \vdash \phi \subseteq K$
3. If $\phi \notin K$ then $K \vdash \phi = K$.
4. If $\not\models \phi$ then $\phi \not\in K \vdash \phi$.
5. If $\phi \in K$ then $K \subseteq (K \vdash \phi) + \phi$.
6. If $\models \phi \iff \psi$ then $K \vdash \phi = K \vdash \psi$.
7. $(K \vdash \phi) \cap (K \vdash \psi) \subseteq K \vdash (\phi \land \psi)$.
8. If $\phi \not\in K \vdash (\phi \land \psi)$, then
$$K \vdash (\phi \land \psi) \subseteq K \vdash \psi.$$
Interdefinability of $\star$ and $\vdash$:

**Harper identity:** Define $\vdash$ using $\star$:  
$$K \vdash \phi = (K \star \neg \phi) \cap K$$

**Levi identity:** Define $\star$ using $\vdash$:  
$$K \star \phi = (K \vdash \neg \phi) + \phi$$

---

**Problems with AGM (1)**

**Belief set vs base:** Up to now: $K = Cn(K)$. But beliefs are usually given by a **finite** set only (a **base**). So there are **basic** beliefs which, should be more persistent than **derived** beliefs.

Katsuno/Mendelzon consider the notion of a **base**, represented by one **single formula** and use another operator $\circ$ to distinguish it from $\star$:  
$$\psi \circ \mu \rightarrow \phi \text{ iff } \phi \in Cn(\{\psi\}) \star \mu$$

---

**Are our choices correct?**

**Theorem 5.2 (From $\star$ to $\vdash$ and back)**

If a **contraction function** $\vdash$ satisfies $K \vdash 1 - K \vdash 4$ and $K \vdash 6$, then the revision function $\star$ defined by the Levi identity satisfies $K \star 1 - K \star 6$. If in addition $K \vdash 7$ or $K \vdash 8$ is satisfied, then so are $K \star 7$ or $K \star 8$.

If a **revision function** $\star$ satisfies $K \star 1 - K \star 6$, then the contraction function $\vdash$ defined by the Harper identity satisfies $K \vdash 1 - K \vdash 6$. If in addition $K \vdash 7$ or $K \vdash 8$ is satisfied, then so are $K \vdash 7$ or $K \vdash 8$.

---

**Lemma 5.3 (Katsuno and Mendelzon, 1992)**

AGM postulates $(K \star 1) - (K \star 8)$ formulated for **finite sets (bases)** are equivalent to the following

1. $\psi \circ \mu \rightarrow \mu$,
2. If $\psi \land \mu$ is satisfiable then $\psi \circ \mu \leftrightarrow \psi \land \mu$.
3. If $\mu$ is satisfiable then $\psi \circ \mu$ is also satisfiable.
4. If $\psi \leftrightarrow \psi'$ and $\mu \leftrightarrow \mu'$ then $\psi \circ \mu \leftrightarrow \psi' \land \mu'$.
5. $(\psi \circ \mu) \land \phi \rightarrow (\psi \circ (\mu \land \phi))$
6. If $(\psi \circ \mu) \land \phi$ is satisfiable, then  
$$\psi \circ (\mu \land \phi) \rightarrow (\psi \circ \mu) \land \phi.$$
5. Knowledge in Flux

1. Revising or Updating Beliefs?

Problems with AGM (2)

Revision vs update: Distinguish new information about a static world (revision) from new information on changes brought about by an agent (updating).

Revision vs updates

We distinguish between revising a static world and updating a dynamically changing world: Updating does not satisfy $K \star 4$.

However, updating satisfies the following:

$$(\alpha \land \alpha') \boxplus \phi = (\alpha \oplus \phi) \land (\alpha' \oplus \phi)$$

which is not valid in the AGM approach.

Example 5.4 (Updating [Winslett, 1988])

1. There is a book, a table and a magazine.
2. Either the book or the magazine is on the table, but not both.

Let $\beta$: the book is on the table, and $\mu$: the magazine is on the table. Then a robot is ordered to put the book on the table.

According to $K \star 4$ (if $\neg \phi \notin K$ then $K \cup \phi \subseteq K \star \phi$), we should end up in the state $Cn(\{\beta, \neg \mu\})$.

But why conclude that the magazine is not on the table?

This is called a dynamically changing world.

Problems with AGM (3)

Informational Economy: This tells us that we should keep as many beliefs as possible. But if we also insist on one single revised theory, this might be too strong an assumption. Hans Rott proved the following. Given a theory $K$ and a formula $\alpha$ we call a theory $\Gamma$ with (1) $\Gamma = Cn(\Gamma) \neq \text{Fml}$ and (2) $\alpha \in \Gamma$ a candidate revision of $K$ by $\alpha$.

Lemma 5.5 (Rott)

Let $\neg \alpha \in K$ and $K_1$, $K_2$ be two candidate revisions. Then neither $K_1 \backslash K \subseteq K_2 \backslash K$ nor $K_1 \backslash K \supseteq K_2 \backslash K$.

Therefore we can’t prefer one to the other.
Problems with AGM (4)

Violation of $K \ast 3$: Suppose we interpret the revision of $K$ by $\alpha \lor \beta$ as **all we know about the statements** $\alpha, \beta$ is that $\alpha \lor \beta$. Then, if $K = \{ \alpha \}$ (before the revision, we knew that $\alpha$), then

$$K \ast (\alpha \lor \beta) = Cn(\{ \alpha \lor \beta \}).$$

Therefore, under such an interpretation, even the following postulate (implied by $K \ast 3$) is violated:

If $\gamma \in K$ then $K \ast \gamma = K$.

Iteration: AGM did not consider any iteration of the revision process. This, however, introduces many new problems. **Should more recent information be handled differently from older one?** What about revisions of the type:

$$(K \vdash \alpha) \ast \alpha?$$

While it seems obvious at first sight, that $K \vdash \alpha \ast \alpha = K$, under the above interpretation of **all we know is $\alpha$**, this axiom is certainly not fulfilled.

Example 5.6 (Hansson)

Let $K = Cn(\{ S, D \})$, where $S$ means **Cleopatra had (at least) a son** and $D$ stands for **Cleopatra had (at least) a daughter**. Then you learn that **Cleopatra had no children at all**: $Cn(\{ S, D \}) \vdash S \lor D$. Finally you are told that **she had at least one child**, but it is not known whether it was a boy or a girl (or several):

$$(Cn(\{ S, D \}) \vdash S \lor D) \ast S \lor D.$$ 

Certainly, you should end up with $Cn(\{ S \lor D \})$, and not with the original $Cn(\{ S, D \})$. 

5. Knowledge in Flux 1. Revising or Updating Beliefs?

5. Knowledge in Flux 2. Updates of Logic Programs
5. Knowledge in Flux  2. Updates of Logic Programs

A logic program represents a static world. By adding new facts, some dynamic change can be incorporated, but only to a limited extent.

What, if new knowledge emerges, that forces us to change the underlying program? This can be seen as an update of the program.

Why not simply applying the theory of belief revision or updates (last section)? The update theory does not work well for nonmonotonic semantics!

In this subsection we discuss the work of [Alferes et al., 2002], [Alferes and Pereira, 2002], [Leite et al., 2001], [Alferes et al., 2004].

They have developed a language to formulate knowledge updates, LUPS.

They have also developed a theory of dynamic logic programming: given a sequence of logic programs \( P_1 \), what is the semantics of the program \( P_1 \oplus P_2 \oplus \ldots \oplus P_n \), the update of \( P_1 \) by the successive updates \( P_2, \ldots, P_n \)?

The semantics of LUPS is reduced to the semantics of \( P_1 \oplus P_2 \oplus \ldots \oplus P_n \). This latter construction is out of the scope of this lecture. We concentrate on formulating logic program updates.

Example 5.7 ([Alferes et al., 2002])

Consider the logic program

\[
\begin{align*}
\text{free} & \leftarrow \text{not jail} \\
\text{jail} & \leftarrow \text{abortion}
\end{align*}
\]

We update this program with the new information \( U_1: \text{abortion} \leftarrow . \) After a while, we learn that \( U_2: \text{not jail} \leftarrow \text{abortion} . \)

Classical update-approaches update specific models. They consider the model \( \{\text{free, abortion} \} \) as a suitable update (after learning that \( U_1 \)). However, the model \( \{\text{abortion, jail} \} \) is a better choice.

After update \( U_2, \text{jail} \) should become false and \( \text{free} \) again true. Again, update approaches do not get these results.
LUPS- an update language

**Definition 5.8 (Knowledge state)**

A **knowledge state** $KS$ is a set of rules (a logic program). An atom holds in $s$ if it is true in all stable models (answer sets) of $KS$.

The idea is that successive updates $U_1, U_2, \ldots, U_n$ are applied to the initial knowledge state $KS_0$ and result in the final state $KS_n = KS_0[U_1] \ldots [U_n]$.

**Updates $U_i$**

Each set $U_i$ consists of (finitely many) update commands of the following form:

1. **assert** $L \leftarrow L_1, \ldots, L_n$ when $L_{k+1}, \ldots, L_m$.
2. **assert event** $L \leftarrow L_1, \ldots, L_n$ when $L_{k+1}, \ldots, L_m$.
3. **always** $L \leftarrow L_1, \ldots, L_n$ when $L_{k+1}, \ldots, L_m$.
4. **always event** $L \leftarrow L_1, \ldots, L_n$ when $L_{k+1}, \ldots, L_m$.
5. **retract** $L \leftarrow L_1, \ldots, L_n$ when $L_{k+1}, \ldots, L_m$.
6. **cancel** $L \leftarrow L_1, \ldots, L_n$ when $L_{k+1}, \ldots, L_m$.

**Example 5.7 in this terminology:**

$KS_0 := \{ free \leftarrow \text{not jail}, \ jail \leftarrow \text{abortion} \}$

$KS = KS_0[\text{assert abortion}][\text{assert not jail} \leftarrow \text{abortion}]$
Definition 5.9 (LUPS)

An update program in LUPS is a finite sequence of update commands of the form mentioned on the previous slide.

Example 5.10 (Parallel updates [Alferes et al., 2002])

A suitcase with two latches opens only when both latches are up. A toggling action is available (and can be applied to each latch). This can be represented by:

- \textit{always} open \leftarrow up(l_1), up(l_2)
- \textit{always} up(L) \textit{when} not up(L), toggle(L)
- \textit{always} not up(L) \textit{when} up(L), toggle(L)

Suppose in the initial situation \(l_1\) is down and \(l_2\) is up, the suitcase is closed. Then there are two toggling actions (one for each latch). And then a toggling action only for \(l_2\).

\[ U_1 = \{\text{assert not up}(l_1), \text{assert up}(l_2), \text{assert not open}\} \]
\[ U_2 = \{\text{assert event toggle}(l_1), \text{assert event toggle}(l_2)\} \]
\[ U_3 = \{\text{assert event toggle}(l_2)\} \]
5. Knowledge in Flux

3. References


The world is dynamic, and agents’ knowledge/beliefs should evolve accordingly (see previous chapter). We showed how modal logic can be used to model knowledge and time separately. In this chapter, we show how these two dimensions can be combined to obtained a better model of an agent.

We present CTLK, which is a straightforward combination of CTL and standard epistemic logic. We use the Muddy Children puzzle to argue that even such a simple idea is a powerful tool for reasoning about evolution of knowledge.

Then, we talk about interpreted systems, a well known formalism that have been successfully applied to modeling synchrony and asynchrony, perfect recall, message passing systems, knowledge bases, distributed systems etc.

Finally, we briefly mention two propositional variants of the BDI framework of beliefs, desires and intentions.
6. Combining Knowledge and Time  1. CTLK

- Simple idea: straightforward combination of temporal and epistemic logic.
- Language includes both kinds of operators
- Models include both kinds of modal relations
- Semantics: union of semantic clauses

Example:

\[
\text{CTLK} = \text{CTL} + \text{Knowledge}
\]

6. Combining Knowledge and Time  1. CTLK

Robots and Carriage revisited

\[
\neg E(\bigvee_i K_1 \text{pos}_i \land \bigvee_i K_2 \text{pos}_i \land \bigvee_i K_3 \text{pos}_i)
\]

\[
E(\bigodot \bigvee_i K_1 \text{pos}_i \land \bigodot \bigvee_i K_2 \text{pos}_i \land \bigodot \bigvee_i K_3 \text{pos}_i)
\]

Note: the latter is a CTLK\(^*\) property!

6. Combining Knowledge and Time  2. Interpreted Systems

Muddy Children revisited

\begin{align*}
\text{mud}_i & \rightarrow E(\bigodot K_i \text{mud}_i) \\
\text{mud}_i & \rightarrow A(\neg K_i \neg \text{mud}_i) \\
\neg \text{mud}_i & \rightarrow A(\neg K_i \neg \text{mud}_i \land \neg K_i \text{mud}_i) \\
\text{mud}_i & \rightarrow K_i E(\bigodot K_i \text{mud}_i)
\end{align*}
6. Combining Knowledge and Time    2. Interpreted Systems

- More grounded notion of epistemic state
- Global states are tuples of local states
- $Q_i$: set of local states of agent $i$
- Global states: $Q \subseteq Q_1 \times \cdots \times Q_k \times Q_{\text{env}}$
- Epistemic relations are based on local states:
  \[ \langle q_1, \ldots, q_k \rangle \sim_i \langle q'_1, \ldots, q'_k \rangle \iff q_i = q'_i \]
- Temporal dimension: runs (paths)

Interpreted systems have been applied to modeling of synchrony and asynchrony, perfect recall, message passing systems, knowledge bases, distributed systems etc.

**Definition 6.1 (System)**

A **system** is a set of runs.

Note: a set of runs can as well be seen as a branching-time tree!

**Definition 6.2 (Interpreted system)**

An **interpreted system** $I$ is a set of runs $\mathcal{R}$ plus valuation of propositions: $\pi : Q \rightarrow \mathcal{P}(\Pi)$.

6. Combining Knowledge and Time    2. Interpreted Systems

- Reasoning about dynamics of knowledge:
  - LTL+Knowledge.
  - Formulae evaluated wrt time points $\langle r, m \rangle$: a run $r$ plus a time moment $m$.
  - That is, $W = \mathcal{R} \times \mathbb{N}$.
  - Epistemic equivalence between points:
    \[ \langle r, m \rangle \sim_i \langle r', m' \rangle \iff r_m \sim_i r'_m. \]
  - Knowledge interpreted as before:
    $I, r, m \models K_i \varphi$ iff $I, r', m' \models \varphi$ for every $\langle r', m' \rangle$ such that $\langle r, m \rangle \sim_i \langle r', m' \rangle$.

**Interpretation of LTL operators:**

- $I, r, m \models \Box \varphi$ iff $I, r, m + 1 \models \varphi$,
- $I, r, m \models \varphi \mathcal{U} \psi$ iff $I, r, m' \models \psi$ for some $m' > m$ and $I, r, m'' \models \varphi$ for all $m''$ such that $m \leq m'' < m'$.

**What about path quantifiers?**

- $I, r, m \models \mathcal{E} \varphi$ iff there is $r'$ such that $r'[0 \ldots m] = r[0 \ldots m]$ and $I, r', m \models \varphi$
6.3 BDI

**BDI = Beliefs, Desires, and 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**
---|---
\( \text{Bel}_i \varphi \) | agent \( i \) believes \( \varphi \)
\( \text{Goal}_i \varphi \) | agent \( i \) has goal of \( \varphi \)
\( \Diamond \alpha \) | action \( \alpha \) will happen next
\( \text{Done} \alpha \) | action \( \alpha \) has just happened

Additionally:
- Action constructors “;” and “?”, as in DL;
- Derived operators: \( \Diamond \alpha \) (sometime \( \alpha \)), \( \Box \alpha \) (always \( \alpha \)), (Later \( \varphi \)): strict sometime, (Before \( \varphi, \psi \)): \( \varphi \) holds before \( \psi \).

---

**Examples:**
\( \text{Goal}_\text{citizen} \square \text{safe}_\text{citizen} \)
\( \text{Goal}_\text{police} \text{Bel}_\text{citizen} \square \text{safe}_\text{citizen} \)
BDI according to Rao and Georgeff:
- Mental primitives: beliefs, desires and intentions
- Time: CTL
- Sophisticated semantic structure

Example: Card Play

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).
6. Combining Knowledge and Time

4. What’s the Use?

6.4 What’s the Use?

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.

**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.

*Beware!*

Not all modal dimensions are independent!

Example: abilities and knowledge
6.5 References


7. Modalities in Action

- In this section, we discuss one particular class of applications of modal logics, namely verification through model checking.
- We demonstrate a simple algorithm for model checking CTLK, discuss complexity of the problem for various temporal logics, and present how the MCMAS model-checker can be used to verify temporal-epistemic properties of agents communicating via the alternating bit protocol.

7. Modalities in Action

- 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 $M, q$
- Verification example: we want to make sure that the cargo can be always moved to the other location.

$$\Box (\text{caL} \rightarrow E\Diamond \text{caP})$$

$$\land \text{caP} \rightarrow E\Diamond \text{caL}$$

$$\land \text{caR} \rightarrow (E\Diamond \text{caL} \lor E\Diamond \text{caP})$$

7. Modalities in Action

- Two perspectives to model checking MAS:
  - Verification
    - Model represents the view of an objective observer
    - Formula: specification to be met
  - Planning
    - Models represent the subjective view of an agent
    - Formula: goal to be achieved
7. Modalities in Action

1. Verification

Example: Modified Rocket Domain, \( E\Box \text{caL} \)

Nice result: model checking CTLK is tractable!

**Theorem (Clarke, Emerson & Sistla 1986)**

CTL model checking is \( P\text{-complete} \), and can be done in time linear in the size of the model and the length of the formula.

Extending the result to CTLK is straightforward.

So... Let’s model-check!

Not as easy as it seems...
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 of Model Checking Temporal and Strategic Logics

<table>
<thead>
<tr>
<th></th>
<th>$m, l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTL</td>
<td>P-complete</td>
</tr>
<tr>
<td>LTL</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>CTL*</td>
<td>PSPACE-complete</td>
</tr>
</tbody>
</table>

$k$ attributes . . . at least $2^k$ states

**Symbolic model checking!**
Explicit vs. Symbolic Model Checking

<table>
<thead>
<tr>
<th></th>
<th>$m, l$</th>
<th>$n_{local}, l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTL</td>
<td>P-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>LTL</td>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>CTL*</td>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
</tbody>
</table>

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.

⇝ demo (MCMAS)

7.3 References
7. Modalities in Action

Automatic verification of finite-state concurrent systems using
 temporal logic specifications.
ACM Transactions on Programming Languages and Systems,

[GiunchigliaTraverso00] Fausto Giunchiglia and Paolo Traverso.
Planning as model checking.
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,

Jürgen Dix and Wojtek Jamroga

8. Where the Frameworks Meet

Chapter 8. Where the Frameworks Meet

Where the Frameworks Meet
8.1 Modalities in LP
8.2 LP in Modal Structures
8.3 There and Back Again
8.4 References

This section is more speculative than the others. Starting with Slide 398, we discuss briefly some LP approaches that are enhanced with suitable modal flavor.

From Slide 423 on, we consider a few approaches that do contain some LP flavor.

Finally, in the last section we try to come up with some comparison between modal approaches on one hand and LP approaches on the other. We formulate two conjectures about the representation of Kripke models as sets of logic programs.

Jürgen Dix and Wojtek Jamroga
8. Where the Frameworks Meet

1. Modalities in LP

8.1 Modalities in LP

Temporal Logic Programming

- Temporal formulae are “injected” into logic programs
- Many variants exist
- Explicit reasoning about properties that evolve over time
- Enforcing execution flow

Example: a TEMPLOG program that generates the sequence of Fibonacci numbers:

\[
\text{fib}(0). \\
\text{fib}(1). \\
\square (\text{fib}(X) \leftarrow \text{fib}(Y), \text{fib}(Z), X \text{ is } Y+Z).
\]

Temporal Programming: METATEM

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.
8. Where the Frameworks Meet

1. Modalities in LP

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

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 401/456

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.

8. Where the Frameworks Meet

1. Modalities in LP

Temporal operators:

\[ \phi \land \psi \] \( \phi \) holds until \( \psi \) \quad \text{primitive}
\[ \phi \rightarrow \psi \] \( \phi \) since \( \psi \) \quad \text{primitive}
\[ \Box \phi \] \( \phi \) is true in the next state
\[ \Box_T \phi \] \( \phi \) since \( \Box_T \phi \)
\[ \Diamond \phi \] \( \phi \) will be true in some future state
\[ [\top] \phi \] \( \phi \) will be true in some past state
\[ [\top_T] \phi \]
\[ \Box_R \phi \] \( \phi \) will be true in all future states
\[ [\top_R] \phi \]
\[ \Diamond_R \phi \] \( \phi \) was true in all past states
\[ [\top_R] \phi \]

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 402/456

Definition 8.1 (FML model)

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

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

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

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

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 403/456

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 404/456
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.

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

**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.

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**
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 agent 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.
8. Where the Frameworks Meet  
1. Modalities in LP

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):

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

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

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

Concurrent METATEM program for the seller agent

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 \( \text{seller} \) identifier, is able to recognize a \( \text{contractProposal} \) 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 \( \text{accept}, \text{refuse}, \text{contractProposal}, \text{ship} \).

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

\[
\forall \text{Buyer, Merchandise, Req_Amnt, Price} \\exists [\text{contractProposal}(\text{Buyer, seller, Merchandise, Req_Amnt, Price}) \\land \text{storing}(\text{Merchandise, Old_Amount}) \\land \text{Old_Amount} \geq \text{Req_Amnt} \\
\land \text{maxprice}(\text{Merchandise, Max}) \land \text{Price} \geq \text{Max}] \\
\Rightarrow [\text{ship}(\text{Buyer, Merchandise, Req_Amnt, Price}) \land \\
\text{accept}(\text{seller, Buyer, Merchandise, Req_Amnt, Price})] \\
\]

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

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.

Situation Calculus

- Another way of extending logic programs with modal elements
- Predicates parameterized with situations
- situation ≈ possible world
- Moreover, execution of an action transforms the current situation into a new one (⇝ dynamic logic)
8. Where the Frameworks Meet

1. Modalities in LP

\[
\text{Percept}([a, \text{Breeze}, g, u, c], s) \rightarrow \text{Breeze}(s) \\
\text{At}(\text{Agent}, l, s) \land \text{Breeze}(s) \rightarrow \text{Breezy}(l, s) \\
\text{At}(p, l, \text{do}(\text{Forward}, s)) \leftrightarrow \text{location\_ahead}(p, l, s) \land \neg \text{Wall}(l, s) \\
\text{Holding}(\text{Gold}, \text{do}(\text{Grab}, s)) \leftrightarrow \text{At\_Gold}(s) \lor \text{Holding}(\text{Gold}, s)
\]

Situation calculus can be seen as a **translation of dynamic logic to FOL**.

8. Where the Frameworks Meet

2. LP in Modal Structures

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 Programming: AGENT-0

AGENT-0 programming language 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),
- \(\text{turn}(I, \text{left})^t\) (instantaneous actions).

No distinction between actions and facts.

Beliefs: \(B_I^{\text{now}} \text{go\_swimming}(I, \text{CLZ})^t\): 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)
- \(OBL_I^t \text{go\_swimming}(I, \text{CLZ})^{t+1}\),
  (at time \(t\), I am obligated to \(y\) to go swimming in CLZ at time \(t + 1\)).

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

A decision is an obligation to oneself: \(\text{DEC}_a^t \varphi := OBL_{(a,a)}^t \varphi\).
### 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:

\[ \text{CAN}^\text{now}_{i} \text{in_lecture} (\text{you})^t \]

(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

\[ \text{COMMITmsgcond mntlcond} \ (\text{agent} \ t \ \text{action}) \]

“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).”

---

**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);

---

**Cycle**

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

At each time step

1. gather incoming messages and update the mental state accordingly,
2. execute commitments (using capabilities).
More recent agent-oriented programming languages follow similar ideas!

- 2APL, 2APL → BDI programming
- Impact
- Jason
- Jack
- ......
From Modal Logic to Logic Programming

In more general terms: agent programming uses logic programs as compact representations of agents’ motivational attitudes.

Logic program = epistemic state
Update = transition

Together with update procedures, logic programs can be used to represent temporal-epistemic models.

Advantages:
- Compact representations
- Can be compact even when the number of possible worlds is huge or infinite

Example: Counting Ducks

```
inhool(tweety).
duckscnt(0,[],0,[]).
duckscnt(N,Duckslist,NewN,[Duck|Duckslist]) ←
  duckscnt(_,_,N,Duckslist), inpool(Duck),
  not in(Duck,Duckslist), NewN is N+1.
ducks(N) ← duckscnt(_,_,N,[]),
  N1 is N+1, not duckscnt(_,_,N1,[]).
¬ducks(N) ← not ducks(N).
```

Additional assumptions:
- Perception: inpool(_), ¬inpool(_)
- At most one percept per step!
- Relevant: ducks(_)

...
8. Where the Frameworks Meet

3. There and Back Again

Representation:

\[
\text{inpool(tweety).}
\text{ducksnt(0,0,0,0).}
\text{ducksnt(N,Duckslist,NewN,[Duck|Duckslist])} \leftarrow
\text{ducksnt(_,_,N,Duckslist), inpool(Duck), not in(Duck,Duckslist), NewN is N+1.}
\text{ducks(N)} \leftarrow \text{ducksnt(_,_,N,_)},
\text{N1 is N+1, not ducksnt(_,_,N1,_)}. 
\text{\neg ducks(N)} \leftarrow \text{not ducks(N)}. 
\]

Perception: \text{inpool(_), \neg inpool(_), warm, warm \leftarrow \neg warm}

At most one percept per step

Relevant: \text{ducks(_), warm}

\[
\text{Conjecture 8.1}
\]

Every finite Kripke model can be represented by a multi-LP system (i.e., a finite collection of logic programs + update procedure).

What about infinite models?
From Logic Programming to Modal Logic

The reverse transformation is also meaningful. A logic programming system can be simulated by a Kripke model.

Conjecture 8.2
Every collection of logic programs \( \langle P_1, \ldots, P_k \rangle \) can be simulated as a pointed Kripke model \( \langle M, w \rangle \) with \( k \) accessibility relations.

Advantage:
- We get semantics of epistemic formulae
- \( \langle P_1, \ldots, P_k \rangle \models K_i Bel_i \phi \) iff \( \langle M, w \rangle \models K_i Bel_i \phi \)
- Note that, for propositional \( \phi \):
  \( \langle P_1, \ldots, P_k \rangle \models Bel_i \phi \) iff \( m \models \phi \) for every answer set \( m \in \text{ASP}(P_i) \)

Transition = update
But: do the agents really know anything about the future?
Suggestion: try the “worlds-within worlds” approach (cf. Rao/Georgeff’s BDI logic)

Exercise
Try to simulate a multi-LP system with a neighborhood model.
METATEM: A framework for programming in temporal logic.
In J. W. de Bakker, W. P. de Roever, and G. Rozenberg (Eds.),
Springer-Verlag.
LNCS 430.

METATEM at work: Modelling reactive systems using executable temporal logic.

A normal form for first-order temporal formulae.
Springer-Verlag.
LNCS 607.

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.
8. Where the Frameworks Meet

4. References


