1. Basics about Logic

Chapter 1. Basics about Logic

Basics about Logic
1.1 Why Logic?
1.2 Sentential Logic
1.3 Examples
1.4 Calculi
1.5 First-Order Logic (FOL)
1.6 Logic Programming
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 declarativeness is lost and PROLOG is not (yet) the answer: emphasis is put on computing answer substitutions.

---

1. Why Logic?

Why logic at all?

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

verification: check specification against implementation,
executable specification,
planning as model checking
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).

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

The language $L_{SL}$ of propositional (or sentential) logic consists of

- $\square$ and $\top$: the constants $\text{falsum}$ and $\text{verum}$,
- $p, q, r, x_1, x_2, \ldots x_n, \ldots$: a countable set $\mathcal{AT}$ of SL-constants,
- $\neg, \land, \lor, \to$: 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 $v(\square) = \text{false}$, $v(\top) = \text{true}$.

Each valuation $v$ can be uniquely extended to a function $\bar{v}: Fml_L \to \{\text{true}, \text{false}\}$ so that:

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

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

Thus each valuation \( \bar{v} \) uniquely defines a \( \bar{v} \). We call \( \bar{v} \) \( L \)-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 \( L \)-formulae into \{true, false\} is called semantics.

Definition 1.3 (Validity of a Formula, Tautology)

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

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

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

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

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

A formula \( \varphi \) follows from \( T \) if for all models \( v \) of \( T \) (i.e. \( v \models T \)) also \( v \models \varphi \) holds. We write: \( T \models \varphi \). We call
\[ Cn_L(T) = \text{def} \{ \varphi \in Fml_L : T \models \varphi \}, \]
or simply \( Cn(T) \), the semantic consequence operator.

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

The semantic consequence operator has the following properties:

1. \( T \)-expansion: \( T \subseteq Cn(T) \),
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

Definition 1.7 (MOD(T), Cn(\text{\textit{U}}))

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

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

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

\[
\text{Cn}(\text{\textit{U}}) = \{ \phi \in \text{Fml}_L : \forall v \in \text{\textit{U}} : \bar{v}(\phi) = \text{true} \}.
\]

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

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

1.3 Examples

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

\( T \) is called \textit{complete} if for each formula \( \phi \in \text{Fml} \): \( T \models \phi \) or \( T \models \neg \phi \) 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 \textit{consistent} if there is a valuation (model) \( v \) with \( \bar{v}(\phi) = \text{true} \) for all \( \phi \in T \).

Lemma 1.10 (Ex Falso Quodlibet)

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

Wumpus World

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:

- \( -S_{1,1} \) \( \rightarrow \) \(( -W_{1,1} \land -W_{1,2} \land -W_{2,1}) \)
- \( -S_{2,1} \) \( \rightarrow \) \(( -W_{1,1} \land -W_{2,1} \land -W_{2,2} \land -W_{3,1}) \)
- \( -S_{1,2} \) \( \rightarrow \) \(( -W_{1,1} \land -W_{1,2} \land -W_{2,2} \land -W_{1,3}) \)
- \( S_{1,2} \) \( \rightarrow \) \(( W_{1,3} \lor W_{1,2} \lor W_{2,2} \lor W_{1,1}) \)
Knowledge after the 3rd move:

\(-S_{1,1} \land \neg S_{2,1} \land S_{1,2} \land \neg B_{1,1} \land \neg B_{2,1} \land \neg B_{1,2}\)

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

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

---

**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:

- \(A_{1,1} \land East \land W_{2,1} \rightarrow \neg Forward\)
- \(A_{1,1} \land East \land Grube_{2,1} \rightarrow \neg Forward\)
- \(A_{i,j} \land Gl_{i,j} \rightarrow TakeGold\)

**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

\(KB \models \neg \text{action or } KB \models \neg \text{action}\)

**But it can happen that neither one nor the other is deducible.**
Can they be solved with sentential logic?

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

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

Solution

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

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

---

1. Basics about Logic

Sudoku

Since some time, Sudoku puzzles are becoming quite famous.

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

Table 1: A simple Sudoku ($S_1$)

We introduce the following language $\mathcal{L}_{\text{Sudoku}}$:

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

This completes the language, the syntax.

How many symbols are these?
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 \( 1 \leq i, j \leq 9 \)).

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

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

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

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

e tc.

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

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

---

How does \( T_S \) look like?

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

Does that mean, that each number 1, …, 9 occurs exactly once in the first square?

No! We have to say, that each number occurs only once:

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

How many formulae are these?

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

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

The same has to be done for all 9 squares.
What is still missing:

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

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

Analogously for all other rows, eg.

Ninth Row: $T_{Row\ 9}$

- eins$_{9,1}$ ∨ eins$_{9,2}$ ∨ ... ∨ eins$_{9,9}$
- zwei$_{9,1}$ ∨ zwei$_{9,2}$ ∨ ... ∨ zwei$_{9,9}$
- drei$_{9,1}$ ∨ drei$_{9,2}$ ∨ ... ∨ drei$_{9,9}$
- vier$_{9,1}$ ∨ vier$_{9,2}$ ∨ ... ∨ vier$_{9,9}$
- fuenf$_{9,1}$ ∨ fuenf$_{9,2}$ ∨ ... ∨ fuenf$_{9,9}$
- sechs$_{9,1}$ ∨ sechs$_{9,2}$ ∨ ... ∨ sechs$_{9,9}$
- sieben$_{9,1}$ ∨ sieben$_{9,2}$ ∨ ... ∨ sieben$_{9,9}$
- acht$_{9,1}$ ∨ acht$_{9,2}$ ∨ ... ∨ acht$_{9,9}$
- neun$_{9,1}$ ∨ neun$_{9,2}$ ∨ ... ∨ neun$_{9,9}$

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

First Column: $T_{Column\ 1}$

- eins$_{1,1}$ ∨ eins$_{1,2}$ ∨ ... ∨ eins$_{1,9}$
- zwei$_{1,1}$ ∨ zwei$_{1,2}$ ∨ ... ∨ zwei$_{1,9}$
- drei$_{1,1}$ ∨ drei$_{1,2}$ ∨ ... ∨ drei$_{1,9}$
- vier$_{1,1}$ ∨ vier$_{1,2}$ ∨ ... ∨ vier$_{1,9}$
- fuenf$_{1,1}$ ∨ fuenf$_{1,2}$ ∨ ... ∨ fuenf$_{1,9}$
- sechs$_{1,1}$ ∨ sechs$_{1,2}$ ∨ ... ∨ sechs$_{1,9}$
- sieben$_{1,1}$ ∨ sieben$_{1,2}$ ∨ ... ∨ sieben$_{1,9}$
- acht$_{1,1}$ ∨ acht$_{1,2}$ ∨ ... ∨ acht$_{1,9}$
- neun$_{1,1}$ ∨ neun$_{1,2}$ ∨ ... ∨ neun$_{1,9}$

Analogously for all other columns.

Is that sufficient? What if a column contains several 1’s?
All put together:

\[ T_{\text{Sudoku}} = T_1 \cup T'_1 \cup \cdots \cup T_9 \cup T'_9 \]
\[ T_{\text{Row 1}} \cup \cdots \cup T_{\text{Row 9}} \]
\[ T_{\text{Column 1}} \cup \cdots \cup T_{\text{Column 9}} \]

Here is a more difficult one.

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

Table 3: A difficult Sudoku \( S_{\text{difficult}} \)
We now define a particular instance of our general notion.

**Definition 1.12 (Calculus for Sentential Logic SL)**

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

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

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

We have now introduced two important notions:

- **Syntactic derivability** \( \vdash \): the notion that certain formulae can be derived from other formulae using a certain calculus,
- **Semantic validity** \( \models \): the notion that certain formulae follow from other formulae based on the semantic notion of a model.
Definition 1.14 (Correct-, Completeness for a calculus)

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

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

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

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

$$\Phi \models \phi \text{ implies } \Phi \vdash \phi.$$
A resolution calculus for SL

The resolution calculus is defined over the language $L^{res} \subseteq L_{SL}$ where the set of well-formed formulae $Fml^{res}_{L}$ consists of all disjunctions of the following form

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

i.e. the disjuncts are only constants or their negations. No implications or conjunctions are allowed. These formulae are also called clauses.

□ is also a clause: the empty disjunction.

We define the following inference rule on $Fml^{res}_{L}$:

**Definition 1.17 (SL resolution)**

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

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

If $C_1 = C_2 = \varnothing$, then $C_1 \lor C_2 = \square$.

If we use the set-notation for clauses, we can formulate the inference rule as follows:

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

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

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

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

Set-notation of clauses

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

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

Thus the set-theoretic union of such sets corresponds again to a clause: $\{A, \neg B\} \cup \{A, \neg C\}$ represents $A \lor \neg B \lor \neg C$. Note that the empty set $\varnothing$ is identified with $\square$. 
Definition 1.19 (Resolution Calculus for SL)

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

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 \( T \vdash \phi \) is equivalent to \( T \cup \{\neg \phi\} \) is unsatisfiable

or rather to
\( T \cup \{\neg \phi\} \vdash \Box \)

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

Theorem 1.20 (Completeness of Resolution Refutation)

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

We also say that resolution is refutation complete.

Definition 1.21 (First order logic \( L_{FOL} \))

The language \( L_{FOL} \) 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 \( k \in \mathbb{N}_0 \): \( P_1^k, P_2^k, \ldots, P_n^k \): 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 \( L_{SL} \)). We suppose that \( \Box \) and \( \top \) are available.
- for each \( k \in \mathbb{N}_0 \): \( f_1^k, f_2^k, \ldots, f_n^k \): 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 $\mathcal{L} \subseteq \mathcal{L}_{FOL}$ (analogously to definition 1.1 on page 2). The used set of predicate and function symbols is also called **signature** $\Sigma$.

### Term

$\mathcal{L}$-terms $t$ are defined as follows:

1. each variable is a $\mathcal{L}$-term.
2. if $f^k$ is a $k$-dimensional function symbol from $\mathcal{L}$ and $t_1, \ldots, t_k$ are $\mathcal{L}$-terms, then $f^k(t_1, \ldots, t_k)$ is a $\mathcal{L}$-Term.

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

### Formula

$\mathcal{L}$-formulae $\phi$ are also defined inductively:

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

### Atomic Formulae

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

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

If $\phi$ is a $\mathcal{L}$-formula and is part of an other $\mathcal{L}$-formula $\psi$ then $\phi$ is called **sub-formula** of $\psi$. 
1. Basics about Logic

5. First-Order Logic (FOL)

An illustrating example

Example 1.22 (From semigroups to rings)

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

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

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 1.23 (**\( L \)-structure \( \mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}}) \)**)

A **\( L \)-structure** or a **\( L \)-interpretation** is a pair \( \mathcal{A} =_{\text{def}} (U_{\mathcal{A}}, I_{\mathcal{A}}) \) 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 \( L \) a \( k \)-dimensional predicate over \( U_{\mathcal{A}} \)
- assigns to each \( k \)-dimensional function symbol \( f^k \) in \( L \) 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 \( L \).

---

Definition (continued)

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

\[
I_{\mathcal{A}}(P) = P^{\mathcal{A}}, \quad I_{\mathcal{A}}(f) = f^{\mathcal{A}}.
\]

\( \varphi \) be a \( L_1 \)-formula and \( \mathcal{A} =_{\text{def}} (U_{\mathcal{A}}, I_{\mathcal{A}}) \) a \( L \)-structure. \( \mathcal{A} \) is called **matching with** \( \varphi \) if \( I_{\mathcal{A}} \) is defined for all predicate and function symbols which appear in \( \varphi \), i.e. if \( L_1 \subseteq L \).

---

Definition 1.24 (**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 1.25 (Semantics of FOL, Model $\mathcal{A}$)

Let $\varphi$ be a formula, $\mathcal{A}$ a structure matching with $\varphi$ and $\rho$ a variable assignment over $\mathcal{A}$. For each term $t$, which can be built from components of $\varphi$, we define inductively the value of $t$ in the structure $\mathcal{A}$, we call $\mathcal{A}(t)$.

1. for a variable $x$ is $\mathcal{A}(x) = \text{def } \rho(x)$.
2. if $t$ has the form $t = f^k(t_1, \ldots, t_k)$, with $t_1, \ldots, t_k$ being terms and $f^k$ a $k$-dimensional function symbol, then $\mathcal{A}(t) = \text{def } f^k(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k))$.

Definition (continued)

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

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

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

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 $x^{\mathcal{A'}}$:

$x^{\mathcal{A'}} = \text{def } d$ (whether $I_\mathcal{A}$ is defined for $x$ or not).
1. Basics about Logic

Definition 1.26 (Tautology)

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

2. A $L$-formula $\varphi$ is called $L$-**tautology**, if for all matching $L$-structures $\mathcal{A}$ the following holds: $\mathcal{A} \models \varphi$.

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

Lemma 1.28 (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**: $Cn(Cn(T)) = Cn(T)$.

Definition 1.27 (Consequence set $Cn(T)$)

A formula $\varphi$ **follows semantically from** $T$, if for all structures $\mathcal{A}$ with $\mathcal{A} \models T$ also $\mathcal{A} \models \varphi$ holds. We write: $T \models \varphi$.

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

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

Lemma 1.29 ($\varphi \notin Cn(T)$)

$\varphi \notin Cn(T)$ if and only if there is a structure $\mathcal{A}$ with $\mathcal{A} \models T$ and $\mathcal{A} \models \neg \varphi$.

In other words: $\varphi \notin Cn(T)$ if and only if there is a **counterexample**: a model of $T$ in which $\varphi$ is **not** true.
**Definition 1.30 (MOD(T), Cn(:UI))**

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

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

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

$$\text{Cn}(\mathcal{U}) = \text{def} \{ \phi \in \text{Fml}_L : \forall \mathcal{A} \in \mathcal{U} : \mathcal{A} \models \phi \}.$$  

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

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

**Jürgen Dix and Wojtek Jamroga**

**ESSLLI (Dublin 2007)** 81/335

---

**Question:**  
If the language bigger is bigger then we can express more. Is $L_{PA'}$ more expressive than $L_{PA}$?

---

**Example 1.33 (Natural numbers in different languages)**

- $N_{PA} = (\mathbb{N}, 0, +, \cdot, =, <)$ ("Presburger Arithmetik"),
- $N_{PA} = (\mathbb{N}, 0, +, \cdot, =, <)$ ("Peano Arithmetik"),
- $N_{PA'} = (\mathbb{N}, 0, 1, +, \cdot, =, <)$ (variant of $N_{PA}$).

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

---

**Answer:**

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

$$N_{PA'} \models \rho \phi(x) \text{ if and only if } \rho(x) = 1 \mathcal{N}.$$  

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

---

1. Basics about Logic

Definition 1.34 (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:
$\Phi \vdash \phi$ implies $\Phi \models \phi$.

**Completeness:** The calculus is **complete** with respect to the semantics, if the following holds:
$\Phi \models \phi$ implies $\Phi \vdash \phi$.

---

1. Basics about Logic

Theorem 1.35 (Correct-, Completeness of FOL)

A formula follows semantically from a theory $T$ if and only if it can be derived:
$T \vdash \phi$ if and only if $T \models \phi$.

Theorem 1.36 (Compactness of FOL)

A formula follows from a theory $T$ if and only if it follows from a finite subset of $T$:
$Cn(T) = \bigcup \{Cn(T') : T' \subseteq T, T' \text{ finite}\}$.
The introduced relation $T \models \phi$ 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 1.37 (Löwenheim-Skolem)**

$T \models \phi$ holds if and only if $\phi$ holds in all countable models of $T$.

**Definition 1.38 (Herbrand model)**

A model $\mathcal{A}$ is called Herbrand model with respect to a language if the universe of $\mathcal{A}$ consists exactly of $\text{Term}_L(\emptyset)$ and the function symbols $f^i_k$ are interpreted as follows:

$$f^i_k: \text{Term}_L(\emptyset) \times \ldots \times \text{Term}_L(\emptyset) \to \text{Term}_L(\emptyset); \quad (t_1, \ldots, t_k) \mapsto f^i_k(t_1, \ldots, t_k)$$

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

**Theorem 1.39 (Reduction to Herbrand models)**

If $T$ is universal and $\phi$ existential, then the following holds:

$T \models \phi$ if and only if $T \models_{\text{Herb}} \phi$

**Question:**

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

No, truth in Herbrand models is highly undecidable.

---

**Wumpus world reconsidered in FOL**

**Question:**

How do we axiomatize the Wumpus-world in FOL?

```markdown
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
```
1. Basics about Logic

We also need a **memory**, a predicate

\[ \text{At}(\text{person}, \text{location}, \text{situation}) \]

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

\[ \text{true afterwards} \iff \text{an action made it true} \]
\[ \text{or it is already true and} \]
\[ \text{no action made it false} \]

## Successor State Axiom

**Axioms about** \( \text{At}(p, l, s) \):

\[ \text{At}(p,l, \text{result}(\text{Forward}, s)) \iff (l \dashv \text{location\_ahead}(p,s) \land \neg \text{Wall}(l)) \]

\[ \text{At}(p,l,s) \rightarrow \text{Location\_ahead}(p,s) \dashedrightarrow \text{Location\_toward}(l, \text{Orient}(p,s)) \]

\[ \text{Wall}([x,y]) \iff (x = 0 \lor x = 5 \lor y = 0 \lor y = 5) \]
The following theorem is the basic result for applying resolution. It states that FOL can be reduced to SL.

**Theorem 1.40 (Herbrand)**

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

Or: Let \( M \) be a set of clauses of 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:

\( M \) is unsatisfiable if and only if

there is a finite and unsatisfiable set \( M_{\text{inst}} \) of basic instances of \( M \).

---

**Definition 1.41 (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.
1. Basics about Logic

- \( p(x,a) = q(y,b) \),
- \( p(g(a), f(x)) = p(g(y), z) \). Basic substitutions are:
  \[ [y/a, x/a, z/f(a)], [y/a, x/f(a), z/f(f(a))], \ldots \]
  The mgU is: \( [y/a, z/f(x)] \).

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 \( Fml_{L^{res}} \) 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 there 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, that a clause now consists of atoms rather than constants, as it was the case of the resolution calculus for SL.

6. Logic Programming

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 is: \( [x/x, y/a, z/h(a)] \).

Definition 1.42 (Robinson’s resolution for FOL)

The resolution calculus consists of two rules:

\[
\begin{align*}
(\text{Res}) & \quad C_1 \cup \{ A_1 \} \quad C_2 \cup \{ \neg A_2 \} \\
& \quad (C_1 \cup C_2) \quad \text{mgU}(A_1, A_2)
\end{align*}
\]

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

\[
\begin{align*}
(\text{Fac}) & \quad C_1 \cup \{ L_1, L_2 \} \\
& \quad (C_1 \cup \{ L_1 \}) \quad \text{mgU}(L_1, L_2)
\end{align*}
\]

Consider for example \( M = \{ r(x) \lor \neg p(x), p(a), s(a) \} \) and the question \( M \models \exists x (s(x) \land r(x)) ? \)?
Definition 1.43 (Resolution Calculus for FOL)

We define the resolution calculus Robinson_{\text{FOL}}^{\text{res}} = \langle \emptyset, \{\text{Res}, \text{Fac}\} \rangle as follows. The underlying language is \( L_{\text{res}} \subseteq L_{\text{FOL}} \) defined on Slide 103 together with the set of well-formed formulae \( Fm_{\text{Res}}^{\text{res}} \).

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

Question:

Is this calculus correct and complete?

Answer (continued):

\[ \square \] 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 1.44 (Resolution is refutation complete)

Robinsons resolution calculus Robinson_{\text{FOL}}^{\text{res}} is refutation complete: given an unsatisfiable set, the empty clause can be derived using resolution and factorization.

Question:
Why do we need factorization?

Answer:
Consider

\[ M = \{ s(x_1) \lor s(x_2), \neg s(y_1) \lor \neg s(y_2) \} \]

Resolving both clauses gives

\[ \{ s(x_1) \} \cup \{ \neg s(y_1) \} \]

or variants of it.
Resolving this new clause with one in \( M \) only leads to variants of the respective clause in \( M \).

Example 1.45 (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:

\[
\begin{array}{ccc}
  r(x) \lor \neg p(x) & p(a) & s(a) \\
  \hline
  r(a) & \neg s(x) \lor \neg r(x) & \neg r(a)
\end{array}
\]
1. Basics about Logic

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

\[
\begin{align*}
-s(x) \lor -r(x) & \quad s(a) \\
- & \quad r(x) \lor -p(x) \\
- & \quad r(a) \\
- & \quad -p(a) \\
\hline
\end{align*}
\]

**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 1.46 (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 1.47 (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**:

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

---

**Theorem 1.48 (Input resolution for Horn clauses)**

Input resolution for Horn clauses is refutation complete.

**Definition 1.49 (SLD resolution wrt P and query Q)**

SLD resolution with respect to a 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 1.50 (Correctness of SLD resolution)

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

Question:
Is SLD completely instantiated?

Definition 1.51 (Computation rule)

A computation rule $R$ is a function which assigns an atom $A_i \in \{A_1, \ldots, A_n\}$ to each query $\Box \leftarrow A_1, \ldots, A_n$. 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:

- infinite ones,
- branches ending with the empty clause (and leading to an answer) and
- failing branches (dead ends).
Theorem 1.52 (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 1.53 (Completeness of SLD resolution)
Each correct answer substitution is subsumed by a calculated answer substitution. I.e.: $P \models \forall Q \Theta$ implies $\exists \sigma : Q\sigma = Q\Theta$

Question: How to find successful branches in a SLD tree?

Definition 1.54 (Search rule)
A search rule is a strategy to search for successful branches in SLD trees.

Note:
PROLOG uses depth-first-search.

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. there must be a guarantee that every atom on the list of goals is eventually chosen.

### Principle 1.1 (PROLOG Paradigm)
*Given a program \( P \) and a query, the proofs of this query (and the computed answers) represent the solution of the formalized problem.*

### Order of atoms

#### Example 1.55 (Termination depends on the order within a rule)

Consider the following two programs:

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

Together with a definition for \( \text{append} \):

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

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

Obviously, (1) and (2) are equivalent, but asking the query \( Q \) wrt (1) yields immediately a “fail”. Asking the query \( Q \) wrt (2) yields a non-terminating computation.
Theorem 1.58 (Soundness and Completeness of SLD)

The following properties are equivalent:

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

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

Programming versus knowledge engineering

<table>
<thead>
<tr>
<th>programming</th>
<th>knowledge engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>choose language</td>
<td>choose logic</td>
</tr>
<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>
2. Answer Set Programming

Chapter 2. Answer Set Programming

Answer Set Programming
2.1 Motivating Examples
2.2 Semantics
2.3 Properties
2.4 ASP engines
2.5 References

1. Basics about Logic

Logic programming and negation: A survey.

Knowledge representation with extended logic programs.
In D. Gabbay and F. Guenthner, editors, Handbook of
Philosophical Logic, volume 12, chapter 1, pages 1–85. Reidel
Shortened version also appeared in Dix, Pereira, Przymusinski
(Eds.), Logic Programming and Knowledge Representation,

Nonmonotonic Reasoning: An Overview., volume 73.

Nonmonotonic reasoning: Towards efficient calculi and
implementations.
In John Alan Robinson and Andrei Voronkov, editors, Handbook
of Automated Reasoning (in 2 volumes), pages 1241–1354.

2. Answer Set Programming

- 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 135 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 146) 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 with 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.

---

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

\[ \text{Grab(Gold): Gold can be contaminated. Then Grab(Gold) is not optimal.} \]

Qualification problem: In logic, it is necessary to enumerate all conditions under which an action is executed successfully.

\[
\forall x \ (\text{Bird}(x) \land \neg \text{Penguin}(x) \\land \neg \text{Dead}(x) \\land \neg \text{Ostrich}(x) \\land \neg \text{BrokenWing}(x) \\land \ldots)
\]

\[
\rightarrow \text{Flies}(x)
\]

It would be ideal to only store “birds normally fly”.

The most natural way is to use “not”

\[
\phi \leftarrow \psi, \ \text{not} \ ab
\]

where $ab$ stands for abnormality. Obviously, this forces us to extend definite programs by negative atoms.
Example 2.1 (Inheritance Hierachies)
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 should be obvious that one should conclude that Tweety is flying.

Jürgen Dix and Wojtek Jamroga

Example 2.2 (Greatest Common Divisor (1))
Given two integers \( n, m \), how can we compute the greatest common divisor?
An answer is to apply Euclid’s algorithm and to write down a recursive definition for the predicate \( \text{gcd}(X, Y, Z) \): \( Z \) is the greatest common divisor of \( X, Y \).

\[
\begin{align*}
\text{gcd}(X, X, X) & : = \\#\text{int}(X), X > 1. \\
\text{gcd}(T, X, Y) & : = X < Y, \text{gcd}(T, X, Y1), Y = Y1 + X. \\
\text{gcd}(T, X, Y) & : = X > Y, \text{gcd}(T, X1, Y), X = X1 + Y.
\end{align*}
\]

This is not declarative: It assumes a lot of mathematical insight. Namely an algorithm that is correct and complete. Could we also just describe the properties, without explicitly giving an algorithm?

Jürgen Dix and Wojtek Jamroga

Example 2.3 (Greatest Common Divisor (2))
Given two integers \( n, m \), how can we compute the greatest common divisor?

\[
\begin{align*}
\text{divisor}(T, N) & : = \\#\text{int}(T), \#\text{int}(N), \#\text{int}(M), N = T \cdot M. \\
\text{gcd}(T, N1, N2) & : = \text{divisor}(T, N1), \text{divisor}(T, N2). \\
\text{larger_cd}(T, N1, N2) & : = T < T1. \\
\text{gcd}(T, N1, N2) & : = \text{gcd}(T, N1, N2), \text{not larger_cd}(T, N1, N2).
\end{align*}
\]

Jürgen Dix and Wojtek Jamroga
Example 2.4 (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 \( \text{reachable}(c) \)? Of course, we expect that neither \( a \) nor \( b \) are reachable because there is no path from \( c \) to either \( a \) or \( b \).

Example 2.5 (COMP vs. NMR)
\[
P_{KR}: \quad \begin{align*}
p & \leftarrow p \\
q & \leftarrow \text{not } p \\
r & \leftarrow \text{not } r
\end{align*}
\]

\[
P'_{KR}: \quad \begin{align*}
p & \leftarrow p \\
q & \leftarrow \text{not } p \\
r & \leftarrow \text{not } r
\end{align*}
\]


Example 2.6 (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. We can formalize that by

\[ \text{wins}(x) \leftarrow \text{move_from_to}(x,y), \text{not } \text{wins}(y) \]

meaning that

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

Assume we also have the facts

\[
\begin{align*}
\text{move_from_to}(a,b) \\
\text{move_from_to}(b,a) \\
\text{move_from_to}(b,c)
\end{align*}
\]

Our query to this program \( P_{game} \) is \( \text{?- wins}(b) \).
2.2 Semantics

Instead of giving a definition out of Pandora’s box, we try to start with a general notion and list properties that a semantics should satisfy.

**Definition 2.7 (SEM)**

A semantics SEM is a mapping from the class of all programs into the powerset of the set of all 3-valued structures. SEM assigns to every program \( P \) a set of 3-valued models of \( P \):

\[ \text{SEM}(P) \subseteq \text{MOD}_{3\text{-val}}(P). \]

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

Principle 2.1 (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 2.1.

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,
- SLDNF is still too weak for Knowledge Representation.

The latter problem is the most important one.

SLDNF answers quite easily our requirements of a semantics SEM (stated explicitly in Definition 2.7).

Principle 2.2 (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.

Lemma 2.8

SLDNF satisfies Reduction.

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 he 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.
2. Answer Set Programming
2. Semantics

What does SLDNF do on Example 2.4?

SLDNF-Resolution does not derive "not reachable(a)".

Partial Evaluation

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

reachable(a) ← edge(a,b), edge(b,a), reachable(a)
reachable(b) ← edge(b,a), edge(a,b), reachable(b)

that can both be eliminated by applying Principle 2.3. So we end up with a program that does neither contain reachable(a) nor reachable(b) in one of the heads. Therefore, according to Principle 2.2 both atoms should be considered false.

Definition 2.9 (GPPE)

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

$A ← (B^+ \setminus \{B\}) \cup B^+_i \land not (B^- \cup B^-_i) (i = 1, \ldots, n)$

where $B ← B^+_i \land not B^-_i (i = 1, \ldots, n)$ are all rules with head $B$.

We reconsider Example 2.5. The semantics of SLDNF corresponds to Clark’s completion $comp$.

Example 2.10 (SLDNF vs. KR (revisited))

<table>
<thead>
<tr>
<th>$P_{KR}$</th>
<th>$P'_{KR}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p ← p$</td>
<td>$p ← p$</td>
</tr>
<tr>
<td>$q ← not p$</td>
<td>$q ← not p$</td>
</tr>
<tr>
<td>$r ← not r$</td>
<td>$r ← not r$</td>
</tr>
</tbody>
</table>

$comp(P_{KR}) : p ← p$ $comp(P'_{KR}) : p ← p$

$q ← ¬p$ $q ← ¬p$

$r ← ¬r$ $r ← ¬r$

?-q: No (SLDNF). Yes (KR).
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 $comp(P)$:


### 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 2.11 (WFS)

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

### Theorem 2.12 (Confluent Calculus for WFS)

*The calculus consisting of these four transformations is confluent, i.e. whenever we arrive at an irreducible program, it is uniquely determined. The order of the transformations does not matter. For finite propositional programs, it is also strongly terminating for fair sequences: any program $P$ is therefore associated a unique normalform $res(P)$. The wellfounded semantics of $P$ can be read off from $res(P)$ as follows*

$$WFS(P) = \{ A : A \leftarrow \in res(P) \} \cup \{ notA : A \text{ is in no head of } res(P) \}$$
What about van Gelder’s Example 2.6
Here we have no problems with floundering, but using SLDNF we get an infinite sequence of oscillating SLD-trees (none of which finitely fails).

$$WFS(P_{game}) = \{\text{not wins}(c), \text{wins}(b), \text{not wins}(a)\}$$

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: $A \leftarrow C, D, \text{not } C$

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

$$(\text{rule})^N := \begin{cases} A \leftarrow B_1, \ldots, B_n, & \text{if } \forall j : C_j \notin N, \\ t, & \text{otherwise.} \end{cases}$$
Note that $P^N$ is always a definite program. We can therefore compute its least Herbrand model $M_{PN}$ and check whether it coincides with the model $N$ with which we started:

**Definition 2.14 (Answer Sets)**

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

### 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 2.6 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 2.15 (Reasoning by cases)

\[ P_{\text{splitting}} : \]
\[
\begin{align*}
    a &\leftarrow \text{not } b \\
    b &\leftarrow \text{not } a \\
    p &\leftarrow a \\
    p &\leftarrow b
\end{align*}
\]

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 2.16 (ASP is not Goal-Oriented)

\[ P_{\text{rel}}(a) : \]
\[
\begin{align*}
    a &\leftarrow \text{not } b \\
    b &\leftarrow \text{not } a \\
    p &\leftarrow \text{not } p \\
    p &\leftarrow a
\end{align*}
\]

$P_{\text{rel}}(a)$ 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

2.3 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 2.17 (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 2.18 (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 2.19 (Constraints)**

Suppose the program $P$ does not contain the predicate $p$. Then the answer sets of the program $P \cup \{ p \leftarrow q_1, \ldots, q_n, r_1, \ldots, r_m \}$ 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 2.20 (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\(xy\) we can write the following program:

\[
\begin{align*}
\text{color}(X,r) \lor \text{color}(X,g) \lor \text{color}(X,b) & : - \ \text{node}(X) \\
: - \ \text{edge}(X,Y), \text{col}(X,C), \text{col}(Y,C)
\end{align*}
\]

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

\[
\begin{align*}
\text{color}(X,r) & : - \ \text{node}(X), \neg \text{color}(X,g), \neg \text{color}(X,b) \\
\text{color}(X,b) & : - \ \text{node}(X), \neg \text{color}(X,r), \neg \text{color}(X,g) \\
\text{color}(X,g) & : - \ \text{node}(X), \neg \text{color}(X,r), \neg \text{color}(X,b)
\end{align*}
\]

Definition 2.22 (NP-Search)

A search problem \(S\) is a pair \(\langle\text{Inst}, \{\text{Sol}_I \mid I \in \text{Inst}\}\rangle\) where

1. \(\text{Inst}\) is an (infinite) set of finite objects, called instances, and
2. 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 \(I \in \text{Inst}\)

\[
\{ \begin{align*}
\mathcal{A} \text{ returns } \text{"No"} , & \text{ if } \text{Sol}_I = \emptyset; \\
\mathcal{A} \text{ returns any } S \in \text{Sol}_I , & \text{ otherwise.}
\end{align*} \]

Example 2.21 (Hamiltonian Path)

Definition 2.23 (Search Problem)

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 173/335
2. Answer Set Programming

Separating Data from the Problem

We would like to separate the search problem itself from the representation of its instances (see Slide 30).

Definition 2.24 (Uniform Representation)

A search problem \( S \) can be represented uniformly in \( \text{ASP} \), 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.

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

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

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 \( \perp \)): then we call the language AnsProlog\( \perp \).

The language AnsProlog\( \perp \) 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 \( A_1, B_1, C_1 \) are positive atoms, which may contain free variables, like \( p(X, Y, c) \). Similar to the above, we define AnsProlog\( \perp, \perp \).

Finally the language AnsProlog\( \perp, \perp \) (resp. AnsProlog\( \perp, \perp, \perp \)) 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
2. Answer Set Programming

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 2.14) 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.

\[
1 \{a, b, \text{not } c\} \leq 2
\]

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

Formalizing Sudoku

smodels uses the following constructs:

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

More constructs in smodels

- The theory

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

means that the whole grid is filled with 5’s and only with 5’s: eg. \(\neg p(X,Y,1)\) is true for all \(X, Y\), as well as \(\neg p(X,Y,2)\) etc. because of the Principle 2.2 that holds for ASP.

\[
1 \{ p(X,Y,A) : \text{val}(A) \} \leq 1
\]

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

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

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

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

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


Modal Logic

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

Modal logic is an extension of classical logic by new connectives $\Box$ and $\Diamond$: necessity and possibility.

- $\Box p$: $p$ is necessarily true
- $\Diamond p$: $p$ is possibly true

Independently of the precise definition:

$$\Diamond p \leftrightarrow \neg \Box \neg p$$

Definition 3.1 (Modal Logic with $n$ modalities)

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

Note that the $\Box$ operators can be nested:

$$(\Box_1 \Box_2 \Box_1 p) \lor \Box_3 \neg p$$

Modal logic can be translated to classical logic;

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

- MSPASS is a theorem prover implementing many modal logics,
- also description logics, relational calculus, etc
- built upon SPASS, a resolution prover for first-order logic with equality,
- check out http://www.cs.man.ac.uk/~schmidt/mspass/.

Example 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 \( L \) is a tuple 
\[ \langle W, I_1, \ldots, I_n \rangle, \]

where

\( W \): a set of possible worlds;

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

\( I_i \): each \( I_i \) is a partition of \( W \):

\( I_i = \{ W_{i_1}, W_{i_2}, \ldots, W_{i_r} \} \) with \( W_{i_j} \cap W_{i_k} = \emptyset \) for \( j \neq k \) and \( \bigcup_{1 \leq j \leq r} W_{i_j} = W \).

The worlds in \( W \) 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 \( w \)

\[ I_i(w) := \{ w' : w \in W_{i_j} \text{ and } w' \in W_{i_j} \} \]
How can we formalise the notion of one agent knowing something? And reason about what agents know and draw conclusions?

- We introduce new operators $K_i$
- $K_i \varphi$: “agent $i$ knows that $\varphi$ holds”

Definition 3.4 (Semantics for partition models)

Let $\mathcal{A} = (W, I_1, \ldots, I_n)$ 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 $\mathcal{A}, w' \models \varphi$.

Example: Robots and Carriage

pos\(_1 \rightarrow K_1 pos_1\)
pos\(_2 \rightarrow \neg K_1 pos_2\)
pos\(_2 \rightarrow K_1 \neg pos_1\)
pos\(_2 \rightarrow K_2 K_1 \neg pos_1\)
pos\(_2 \rightarrow \neg K_1 K_2 K_1 \neg pos_1\)
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 $w_1$.

**Definition 3.5 (Kripke Structure)**

A Kripke structure $\langle W, R \rangle$ is a set of possible worlds $W$ plus a binary relation $R \subseteq W \times W$ (the accessibility relation).

Note that:
- Elements of $W$ are now abstract entities: we assume no internal structure to them!
- Still, we would usually like to see them as classical (e.g., propositional) models. How can it be done?
- Let $\Pi$ be the set of all propositional symbols. A propositional model can be represented with a list of propositions $\pi(w) \subseteq \Pi$ that hold in world $w$.

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

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

$$\langle W, R, \pi \rangle,$$

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

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

Given a Kripke model $M = \langle W, R, \pi \rangle$, and a world $w \in W$, we define the satisfaction relation $\models$ as follows:

- $M, w \models p$ iff $p \in \pi(w)$
- $M, w \models \varphi \land \psi$ iff $M, w \models \varphi$ and $M, w \models \psi$
- $M, w \models \neg \varphi$ iff not $M, w \models \varphi$
- $M, w \models \Box \varphi$ iff for every $w' \in W$ with $wRw'$ we have that $M, w' \models \varphi$. 
What if we want multiple modalities $\Box_1, \ldots, \Box_k$?

Then, we need multiple accessibility relations $R_1, \ldots, R_k \subseteq W \times W$, one per modality.

Example: Robots and Carriage

3. Reasoning about Muddy Children
3. Modal Logic

3.4 Axioms for Modal Logics

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

**Definition 3.9 (System K)**

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

\[ \Box \phi \land \Box (\phi \rightarrow \psi) \rightarrow \Box \psi \]

and the inference rule

\[ \frac{\Box \phi}{\Box \Box \phi} \]

Jürgen Dix and Wojtek Jamroga
3. Modal Logic

**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

**Definition 3.11 (Extending K by Axioms D, T, 4, 5)**

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

- **K** ($K_i\varphi \land K_i(\varphi \rightarrow \psi) \rightarrow K_i\psi$) [logical omniscience]
- **D** $\neg K_i(\varphi \land \neg \varphi)$ [consistency]
- **T** $K_i\varphi \rightarrow \varphi$ [truth]
- **4** $K_i\varphi \rightarrow K_iK_i\varphi$ [positive introspection]
- **5** $\neg K_i\varphi \rightarrow K_i\neg K_i\varphi$ [negative introspection]

The system consisting of $KT45$ is also called $S5$.

### Notes

- 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, what additional statements (axioms) are true in all Kripke models?

**Theorem 3.12 (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 3.13 (Sound-, completeness of KT45 (S5))**

System $KT45$ is sound and complete with respect to Kripke structures with equivalence accessibility relations.
3.5 References


3. Modal Logic


5. References


4. Logics of Action and Time

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.

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 237/335

4. Logics of Action and Time

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!

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 238/335

4. Logics of Action and Time

1. Dynamic Logic

4.1 Dynamic Logic

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

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

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

Jürgen Dix and Wojtek Jamroga

ESSLLI (Dublin 2007) 239/335
3rd 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

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

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

\[\alpha \rightarrow Q \times Q.\]

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

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

**Definition 4.3 (Semantics of DL)**

\[M, s \models [\alpha]\varphi\] iff for every \(t\) such that \(s \xrightarrow{\alpha} t\), we have \(M, t \models \varphi\).
4. Logics of 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>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigcirc \varphi$</td>
<td>$\varphi$ is true in the next moment in time</td>
</tr>
<tr>
<td>$\Box \varphi$</td>
<td>$\varphi$ is true in all future moments</td>
</tr>
<tr>
<td>$\Diamond \varphi$</td>
<td>$\varphi$ is true in some future moment</td>
</tr>
<tr>
<td>$\varphi U \psi$</td>
<td>$\varphi$ is true until the moment when $\psi$ becomes true</td>
</tr>
</tbody>
</table>

$send(msg, rcvr) \rightarrow \Diamond receive(msg, rcvr)$

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

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

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

- Safety properties
- Liveness properties
- Fairness properties
4. Logics of Action and Time

2. Temporal Logic

Safety:

“something bad will not happen”
“something good will always hold”

Typical examples:

\[ \neg \square \text{bankrupt} \]
\[ \square (\text{fuelOK} \lor \bigcirc \text{fuelOK}) \]

and so on . . .

Usually: \( \square \neg \ldots \)

Liveness:

“something good will happen”

Typical examples:

\[ \Diamond \text{rich} \]
\[ \text{rocketLondon} \rightarrow \Diamond \text{rocketParis} \]

and so on . . .

Usually: \( \Diamond \ldots \)

Combinations of safety and liveness possible:

\[ \Diamond \square \text{rich} \]
\[ \square \Diamond \text{rich} \rightarrow \text{fairness} \]

Strong fairness:

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

Typical examples:

\[ \square (\text{attempt} \rightarrow \Diamond \text{success}) \]
\[ \square \Diamond \text{attempt} \rightarrow \square \Diamond \text{success} \]
4. Logics of Action and Time
3. Linear Time Logic

4.3 Linear Time Logic

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 4.4 (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
Important: computational vs. behavioral structure

Computational str.

Behavioral str.

LTL models are defined as behavioral structures!

Definition 4.5 (Semantics of LTL)

\( \lambda \models p \) iff \( p \) is true at moment \( \lambda[0] \);

\( \lambda \models \Diamond \varphi \) iff \( \lambda[1..\infty] \models \varphi \);

\( \lambda \models \varphi \) iff \( \lambda[i..\infty] \models \varphi \) for some \( i \geq 0 \);

\( \lambda \models \Box \varphi \) iff \( \lambda[i..\infty] \models \varphi \) for all \( i \geq 0 \);

\( \lambda \models \varphi \U \psi \) iff \( \lambda[i..\infty] \models \psi \) for some \( i \geq 0 \), and \( \lambda[j..\infty] \models \varphi \) for all \( 0 \leq j \leq i \).

Note that:

\( \Box \varphi \equiv \neg \Diamond \neg \varphi \)

\( \Diamond \varphi \equiv \neg \Box \neg \varphi \)

\( \Diamond \varphi \equiv \top \U \varphi \)
4. Logics of Action and Time

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.

---

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

**Temporal operators**: $\bigcirc$ (nexttime), $\lozenge$ (sometime), $\Box$ (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 4.6 (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.

CTL models are defined as computational structures!

**Definition 4.7 (Paths in a model)**

A path \( \lambda \) is an infinite sequence of states that can be effected by subsequent transitions. A path must be full, i.e. either infinite, or ending in a state with no outgoing transition.

Usually, we assume that the transition relation is serial (time flows forever). Then, all paths are infinite.
Example: Rocket and Cargo

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

**Definition 4.8 (Semantics of CTL*: state formulae)**

\[
M, q \models E \varphi \quad \text{iff there is a path } \lambda, \text{ starting from } q, \text{ such that } M, \lambda \models \varphi; \\
M, q \models A \varphi \quad \text{iff for all paths } \lambda, \text{ starting from } q, \text{ we have } M, \lambda \models \varphi.
\]

**Definition 4.9 (Semantics of CTL*: path formulae)**

Exactly like LTL!

\[
M, \lambda \models \Diamond \varphi \quad \text{iff } M, \lambda[1, \infty] \models \varphi; \\
M, \lambda \models \varphi \lor \psi \quad \text{iff } M, \lambda[i, \infty] \models \psi \text{ for some } i \geq 0, \text{ and } M, \lambda[j, \infty] \models \varphi \text{ for all } 0 \leq j \leq i.
\]
Exercise:

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

Practical Importance of Temporal and Dynamic Logics:

Automatic verification in principle possible (model checking).
Can be used for automated planning. Executable specifications can be used for programming.

Note:

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

4.5 References


Combining Knowledge and Time

5. Combining Knowledge and Time

5.1 CTLK

5.2 Interpreted Systems

5.3 BDI

5.4 What’s the Use?

5.5 References

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

How about combining them?

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:

\[ \text{CTLK} = \text{CTL} + \text{Knowledge} \]
Muddy Children revisited

\[
\text{mud}_i \rightarrow E \square K_i \text{mud}_i \\
\text{mud}_i \rightarrow A \square \neg K_i \neg \text{mud}_i \\
\neg \text{mud}_i \rightarrow A \square (\neg K_i \neg \text{mud}_i \land \neg K_i \text{mud}_i) \\
\text{mud}_i \rightarrow K_i E \square K_i \text{mud}_i
\]

Robots and Carriage revisited

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

Note: the latter is a CTLK* property!

5. Combining Knowledge and Time  

2. Interpreted Systems

5.2 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 5.1 (System)
A system is a set of runs.

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

Definition 5.2 (Interpreted system)
An interpreted system $I$ is a set of runs $R$ plus valuation of propositions: $\pi : Q \rightarrow P(\Pi)$.

Reasoning about dynamics of knowledge: LTL+Knowledge
Formule evaluated wrt time points $<r,m>$: a run $r$ plus a time moment $m$
That is, $W = R \times \mathbb{N}$
Epistemic equivalence between points:
$<r,m> \sim_i <r',m'>$ iff $r_m \sim_i r'_m$

Knowledge interpreted as before:
$I, r, m \models K_i \phi$ iff $I, r_1, m_1 \models \phi$ for every $<r',m'>$ such that $<r,m> \sim_i <r',m'>$

Interpretation of LTL operators:
$I, r, m \models \bigcirc \phi$ iff $I, r, m + 1 \models \phi$,
$I, r, m \models \phi \mathcal{U} \psi$ iff $I, r, m' \models \psi$ for some $m' > m$
and $I, r, m'' \models \phi$ for all $m''$ such that $m \leq m'' < m'$.

What about path quantifiers?
$I, r, m \models \mathcal{E} \phi$ iff there is $r'$ such that $r'[0...m] = r[0...m]$ and $I, r', m \models \phi$
5. Combining Knowledge and Time

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), (\text{Later} \varphi): strict sometime, (\text{Before} \varphi, \psi): \varphi holds before \psi.

Examples:
- \text{Goal}_{\text{citizen}} \Diamond_{\text{safe}}_{\text{citizen}}
- \text{Goal}_{\text{police}} \text{Bel}_{\text{citizen}} \Diamond_{\text{safe}}_{\text{citizen}}

BDI according to Rao and Georgeff:
- Mental primitives: beliefs, desires and intentions
- Time: CTL
- Sophisticated semantic structure
5. Combining Knowledge and Time
3. BDI

Example: Card Play

\[
\begin{align*}
&\text{Winning states:} \\
&\text{AK1, QK2, AQ2, KQ2} \\
&\text{Bel}_A \text{E} \text{win}: \text{Agent A believes that there is a way to win in one step} \\
&\text{Des}_A \text{A} \text{E} \text{win}: \text{the agent desires that every path leads to a victory, so he does not have to worry about his decisions} \\
&\text{However, he does not believe it is possible:} \\
&\neg \text{Bel}_A \text{E} \text{win}
\end{align*}
\]

Of course, it is possible to extend BDI:

- **Horizontally**: with other modal dimensions (e.g., BOID);
- **Vertically**: to a language of higher order (e.g., LORA).
What do we use these frameworks for?

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

Verification & Exploration
- Reasoning about concrete systems;
- Correctness testing.

What do we use these frameworks for?

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

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

Beware!
Not all modal dimensions are independent!

Example: abilities and knowledge

5.5 References
5. Combining Knowledge and Time

5. References


6. Knowledge in Flux

Chapter 6. Knowledge in Flux

Knowledge in Flux
6.1 Revising or Updating Beliefs?
6.2 Updates of Logic Programs
6.3 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 ∗: 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.
6. Knowledge in Flux
1. Revising or Updating Beliefs?

6.1 Revising or Updating Beliefs?

The AGM approach

AGM: Alchourron/Gärdenfors/Makinson. Seminal paper in 1985: [Alchourrón et al., 1985].

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

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

Problem: If \( K \vdash \phi \), then we have no problems: \( K * \phi = \text{Cn}(K) \).

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

What to do when \( K \vdash \neg \phi \)?

Example 6.1 (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!

That means we want to revise \( \{\alpha, \beta, \gamma, \delta\} \) by \( \neg \epsilon \).

Shall we give up \( \alpha \) but keep some of its consequences:

- All European swans except the 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 $K$.

- **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 $K \star \phi$.

---

**Axioms about $\perp$**

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

---

**Interdefinability of $\star$ and $\perp$**

- **Harper identity:** Define $\perp$ using $\star$:
  $$K \perp \phi = (K \star \neg \phi) \cap K$$
- **Levi identity:** Define $\star$ using $\perp$:
  $$K \star \phi = (K \perp \neg \phi) + \phi$$
Are our choices correct?

**Theorem 6.2 (From \( \cdot \) to \( \vdash \) and back)**

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

If a revision function \( \cdot \) satisfies \( K \cdot 1-K \cdot 6 \), 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 \( K \vdash 8 \).

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 \( \cdot \):

\[
\psi \circ \mu \rightarrow \phi \text{ iff } \phi \in Cn(\{\psi\}) \star \mu
\]

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).
Example 6.4 (Updating [Winslett, 1988])

1. There is a book, a table and a magazine.
2. $\beta$: the book is on the table.
3. $\mu$: the magazine is on the table.
4. Not both are on the table.

Then a robot is ordered to put the book on the table. According to $K \star 4$, we should end up in the state $Cn(\{\beta, -\mu\})$. But why conclude that the magazine is not on the table?

This is a dynamically changing world.

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:

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

which is not valid in the AGM approach.

6.2 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 $P_i$, 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 6.5 ([Alferes et al., 2002])

Consider the logic program

- $\textit{free} \leftarrow \textit{not jail}$
- $\textit{jail} \leftarrow \textit{abortion}$

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

---

Classical update-approaches update specific models.

They consider the model $\{\textit{free}, \textit{abortion}\}$ as a suitable update (after learning that $U_1$).

However, the model $\{\textit{abortion}, \textit{jail}\}$ is a better choice.

After update $U_2$, $\textit{jail}$ should become false and $\textit{free}$ again true. Again, update approaches do not get these results.

---

LUPS- an update language

Definition 6.6 (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 6.5 in this terminology:

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

$KS = KS_0[\text{assert abortion}][\text{assert not jail} \leftarrow \text{abortion}]$

---

**Definition 6.7 (LUPS)**

An *update program* in LUPS is a finite sequence of update commands of the form mentioned on the previous slide.
Example 6.8 (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

\[
\begin{align*}
\text{always open} & \leftarrow \text{up}(l_1).\text{ip}(l_2) \\
\text{always up}(L) & \quad \text{when not up}(L).\text{toggle}(L) \\
\text{always not up}(L) & \quad \text{when up}(L).\text{toggle}(L)
\end{align*}
\]

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 \(l_2\).

\[
\begin{align*}
U_1 & = \{\text{assert not up}(l_1), \text{assert up}(l_2), \text{assert not open}\} \\
U_2 & = \{\text{assert event toggle}(l_1), \text{assert event toggle}(l_2)\} \\
U_3 & = \{\text{assert event toggle}(l_2)\}
\end{align*}
\]

6.3 References


6. Knowledge in Flux

3. References

Lupsa language for updating logic programs.

Semantics for dynamic logic programming: A principle-based approach.

Propositional knowledge base revision and minimal change.

Multi-dimensional dynamic knowledge representation.

A framework for comparison of update semantics.