# Artificial Intelligence 1

Prof. Dr. Frank Hopfgartner
Dr. Matthias Horbach

Institute for Web Science and Technologies (WeST)
University of Koblenz

# Overview

# Knowledge representation and reasoning

- Definition: *Knowledge representation and reasoning (KR) is the field of artificial intelligence (AI) dedicated to representing information about the world in a form that a computer system can utilize to solve complex tasks such as diagnosing a medical condition.* *[Wikipedia]*
- Goal: Formalisation of beliefs (knowledge) and automatic reasoning
- But we already looked at propositional and first-order logic, isn't that enough?

# Recall: logics

Every logic (=formal system) has the following components:

1. Syntax: What are the possible statements?
   1.1 Signature: What symbols are allowed? ($S = \{\text{Anna}, \text{human}, \text{student}\}$)
   1.2 Grammar: how can symbols be combined in order to obtain complex statements? (student $\Rightarrow$ human)
2. Semantics: Which are the "true" statements? What is the relationship between true statements?
   2.1 Interpretations: Which symbol represents which concrete object? (Anna = "Anna Schmidt")
   2.2 Models: Which statement is true in a given constellation of objects?
   2.3 Reasoning: How can we infer new information?
3. Calculus: How can "reasoning" be implemented?

# Limitations of classical logic 1/4

- Classical logic is monotonic:

$$\text{If } \alpha \vdash \beta \qquad \text{then} \qquad \alpha \wedge \gamma \vdash \beta$$

  for every formula $\gamma$.
- In other words: inferences are never retracted when new information is received
- Classical logic is binary:

$$\alpha \wedge (\alpha \Rightarrow \beta) \vdash \beta$$

  is always true without exception.
- Central concept in knowledge representation are *rules*
- Rules always have exceptions (and there are also exceptions for that)

# Limitations of classical logic 2/4

## Example

Let us model our knowledge about some animals:

- birds are animals
- penguins are birds
- birds usually fly
- penguins do not fly

Naive formalisation:

$$\forall X : (\mathit{bird}(X) \Rightarrow \mathit{animal}(X))$$
$$\forall X : (\mathit{penguin}(X) \Rightarrow \mathit{bird}(X))$$
$$\forall X : (\mathit{bird}(X) \Rightarrow \mathit{flies}(X))$$
$$\forall X : (\mathit{penguin}(X) \Rightarrow \neg \mathit{flies}(X))$$
$$\mathit{penguin}(\mathit{tweety})$$

What is the problem?

# Limitations of classical logic 3/4

## Example

Knowledge base in propositional logic (simpler)

$$\phi = (\mathit{bird} \Rightarrow \mathit{animal}) \wedge (\mathit{penguin} \Rightarrow \mathit{bird}) \wedge (\mathit{bird} \Rightarrow \mathit{flies}) \\ \wedge (\mathit{penguin} \Rightarrow \neg \mathit{flies}) \wedge \mathit{penguin}$$

Observation:

- Knowledge base is inconsistent: $\phi \vdash \mathit{flies}$ and $\phi \vdash \neg \mathit{flies}$
- The "rule" $\mathit{bird} \Rightarrow \mathit{flies}$ is not a classical implication, it is not universally valid
- What now?
- Model exceptions explicitly: $\mathit{bird} \Rightarrow \mathit{flies} \longrightarrow \mathit{bird} \wedge \neg \mathit{penguin} \Rightarrow \mathit{flies}$
- What about emus, dodos, ostriches, ...?

# Limitations of classical logic 4/4

- Explicit enumeration of exceptions is not feasible (hard to maintain, . . . )
- How do we humans do it?
- When encountering a new bird for the first time, we make the *default assumption* that it can fly
- Only when obtaining explicit information that the bird does not fly, we will revise our inference

# Overview

# Overview

- Default logics are logics that allow for *non-monotonic* reasoning

$$\text{Even if } \alpha \vdash \beta \qquad \text{it may be that} \qquad \alpha \wedge \gamma \nvdash \beta$$

- In the following, we consider the prototype of a default logic: Reiter's default logic.

  *R. Reiter. A logic for default reasoning. Artificial Intelligence, 13:81–132, 1980.*

# Some additional notation for first-order logic 1/2

Let $\Sigma = (U, P, F)$ be a first-order signature, $V$ a set of variables, and $\mathcal{L}(\Sigma, V)$ the corresponding first-order language.

## Definition

A variable $X \in V$ is *free* in formula $\phi$ if there is an occurrence of $X$ in $\phi$ which is not quantified. A variable $X \in V$ is *bound* in $\phi$ if it occurs in a sub-formula of the type $\forall X : \phi'$ or $\exists X : \phi'$.
A formula $\phi \in \mathcal{L}(\Sigma, V)$ is *closed* if every appearing variable is bound.

## Example

The following two formulas are closed:

$$\forall X : (a(X) \wedge \forall Y : b(X, Y)) \qquad\qquad r(s, t) \vee \exists Y : a(Y)$$

The following formulas are not closed

$$(\forall X : a(X)) \wedge b(X) \qquad\qquad r(s, Y) \vee \exists X : a(X)$$

# Some additional notation for first-order logic 2/2

A set of first-order formulas is usually seen as equivalent with the conjunction of its elements

$$\{\phi_1, \phi_2, \phi_3\} \vdash \psi \qquad \Longleftrightarrow \qquad \phi_1 \wedge \phi_2 \wedge \phi_3 \vdash \psi$$

### Definition

The *deductive closure* of a set of first-order formulas $\Psi \subseteq \mathcal{L}(\Sigma, V)$ is defined via $Cn(\Psi) = \{\phi \mid \Psi \vdash \phi\}$.

Observe that $\Psi \subseteq Cn(\Psi)$ is always the case.

### Definition

A set of first-order formulas $\Psi \subseteq \mathcal{L}(\Sigma, V)$ is *deductively closed* if $Cn(\Psi) = \Psi$.

# Default rules

### Definition

Let $\phi, \psi_1, \ldots, \psi_n, \chi \in \mathcal{L}(\Sigma, V)$ be closed first-order formulas (or propositional formulas).

Then

$$\delta = \frac{\phi : \psi_1, \ldots, \psi_n}{\chi}$$

is called a *default rule* (or just *default*).

Meaning: If $\phi$ is known and $\psi_1, \ldots, \psi_n$ can be consistently assumed, then conclude $\chi$.

| | |
|---|---|
| $\phi = \mathit{pre}(\delta)$ | (default) precondition |
| $\chi = \mathit{cons}(\delta)$ | (default) conclusion |
| $\{\psi_1, \ldots, \psi_n\} = \mathit{just}(\delta)$ | (default) justifications |

# Default rules - example

- Rules with exceptions:

$$\frac{bird : flies}{flies}$$

- Rules that are typically true:

$$\frac{going\_to\_work : take\_bus}{take\_bus}$$

- Rules that hold unless the opposite can be shown:

$$\frac{accused : innocent}{innocent}$$

# Default schema

A default

$$\delta = \frac{\phi : \psi_1, \dots, \psi_n}{\chi}$$

with *open* formulas $\phi, \psi_1, \dots, \psi_n, \chi$ is interpreted as a schema, i. e., as its set of grounded defaults (over the given universe).

# Default schema - example

The default rule

$$\frac{friend(X, Y) \wedge friend(Y, Z) : friend(X, Z)}{friend(X, Z)}$$

is a shorthand for (given $U = \{tom, bob, sally, tina\}$)

$$\frac{friend(tom, bob) \wedge friend(bob, sally) : friend(tom, sally)}{friend(tom, sally)}$$

$$\frac{friend(tom, bob) \wedge friend(bob, tina) : friend(tom, tina)}{friend(tom, tina)}$$

...

# Default theory (syntax of default logic)

### Definition

A default theory $T$ is a tuple $T = (W, \Delta)$ with

- $W \subseteq \mathcal{L}(\Sigma, V)$ (facts)
- $\Delta$ set of default rules

Main idea of default reasoning (semantics)

- apply defaults on $W$ to extend the knowledge with plausible additional information
- apply defaults as long as no further information can be added

The resulting set of classical formulas is called *extension* and represents a plausible "belief state" of the default theory.

# Main challenges

$$\delta = \frac{\phi : \psi_1, \ldots, \psi_n}{\chi}$$

Meaning: If $\phi$ is known and $\psi_1, \ldots, \psi_n$ can be consistently assumed, then conclude $\chi$.

When is such a default $\delta$ applicable, i. e.

- When is $\phi$ known?
- When can $\psi_1, \ldots, \psi_n$ be consistently assumed?

Approach: use a (at first unknown) extension $E$ to check these conditions:

- $\phi$ is known iff $\phi \in E$;
- $\psi_1, \ldots, \psi_n$ can be consistently assumed iff $\neg\psi_i \notin E$, $1 \leq i \leq n$.

# Semantics of default logic: extensions 1/3

An extension $E \subseteq \mathcal{L}(\Sigma, V)$ is characterised by the following properties:

- $E$ contains all facts: $W \subseteq E$
- $E$ is deductively closed: $Cn(E) = E$
- $E$ is *closed under default application*, i. e. if $\delta = \frac{\phi:\psi_1,\ldots,\psi_n}{\chi} \in \Delta$ is applicable in $E$ then $\chi \in E$ where:

  $\delta$ is applicable in $E$ iff $\phi \in E$ and $\neg\psi_1 \notin E, \ldots, \neg\psi_n \notin E$

# Semantics of default logic: extensions 2/3

More general:

- Let $F$ be a deductively closed set of formulas
- Let $K$ be a set of formulas (the context)

A default $\delta = \frac{\phi:\psi_1,\ldots,\psi_n}{\chi}$ is *applicable* in $F$ wrt. $K$ iff

$$\phi \in F \quad \text{and} \quad \neg\psi_1 \notin K, \ldots, \neg\psi_n \notin K$$

# Semantics of default logic: extensions 3/3

Let $T = (W, \Delta)$ be a default theory and $S$ a set of formulas.

Define $\Lambda_T(S)$ to be the smallest set of formulas with

- $\Lambda_T(S)$ is deductively closed
- $W \subseteq \Lambda_T(S)$
- $\Lambda_T(S)$ is closed under default application wrt. the context $S$, i. e. for all $\delta = \frac{\phi:\psi_1,\ldots,\psi_n}{\chi} \in \Delta$, if $\phi \in \Lambda_T(S)$ and $\neg\psi_1 \notin S, \ldots,$ $\neg\psi_n \notin S$, then $\chi \in \Lambda_T(S)$.

**Definition**

$E$ is an *extension* of $T = (W, \Delta)$ iff $\Lambda_T(E) = E$.

Remark: conceptually, extensions of default theories correspond to models of a classical logic formula.

# Example

$$T = (\{aquatic\_creature\}, \{\frac{aquatic\_creature : fish}{fish}\})$$

$E = Cn(\{aquatic\_creature, fish\})$ is an extension of $T$,

$E' = Cn(\{aquatic\_creature, \neg fish\})$ is *not* an extension of $T$, despite the fact that

- $\{aquatic\_creature\} \subseteq E'$
- $E'$ is deductively closed
- $E'$ is closed under default application

but $\Lambda_T(E') = Cn(\{aquatic\_creature\}) \neq E'$

# More examples

**Example**

$T = (\{p\}, \{\frac{p:q}{q}, \frac{q:r}{r}\})$

- $E = Cn(\{p, q, r\})$

**Example**

$T = (\{p\}, \{\frac{p:q}{q}, \frac{p:\neg q}{\neg q}\})$

- $E_1 = Cn(\{p, q\})$
- $E_2 = Cn(\{p, \neg q\})$

**Example**

$T = (\{p\}, \{\frac{p:r}{s}\})$

- $E = Cn(\{p, s\})$

# A characterisation of extensions

**Theorem (Reiter, 1980)**

*Let $E$ be a set of closed formulas and let $T = (W, \Delta)$ be a default theory. Define a sequence of sets of formulas $E_i$, $i \geq 0$ via*

- $E_0 = W$
- $E_{i+1} = Cn(E_i) \cup \{\chi \mid \frac{\phi:\psi_1,\ldots,\psi_n}{\chi} \in \Delta, \phi \in E_i,$
  $\neg\psi_1 \notin E, \ldots, \neg\psi_n \notin E\}$

*Then $E$ is an extension of $T$ iff*

$$E = \bigcup_{i=0}^{\infty} E_i$$

# Computing extensions

Task: given $T = (W, \Delta)$, enumerate all extensions $E$ of $T$

- We discuss an algorithm to explicitly enumerate all extensions using *default processes* and *process trees*
- Idea: apply defaults successively
  - as long as possible
  - use backtracking if inconsistency occurs

Let $T = (W, \Delta)$ be a fixed default theory.

# Default sequences

$\Pi = (\delta_0, \ldots, \delta_m)$ sequence of defaults from $\Delta$ (finite, no repetitions)

$\Pi[k] = (\delta_0, \ldots, \delta_{k-1})$ sub-sequence of the first $k$ elements

## Definition

Let $\Pi$ be a default sequence, define

$$
\begin{aligned}
In(\Pi) &= Cn(W \cup \{cons(\delta) \mid \delta \in \Pi\}) \\
Out(\Pi) &= \{\neg\psi \mid \psi \in just(\delta), \delta \in \Pi\}
\end{aligned}
$$

- $In(\Pi)$ collects formulas that are concluded by applying defaults from $\Pi$; represents the current belief state after processing $\Pi$
- $Out(\Pi)$ collects formulas that should be not be proven true later

# Default sequences - example

$T = (W, \Delta)$ with

$$W = \{a\} \qquad \Delta = \{\delta_1 = \frac{a : \neg b}{\neg b}, \delta_2 = \frac{a : c}{c}\}$$

Let $\Pi_1 = (\delta_1)$, $\Pi_2 = (\delta_2, \delta_1)$.

$$In(\Pi_1) = Cn(\{a, \neg b\}) \qquad Out(\Pi_1) = \{b\}$$

$$In(\Pi_2) = Cn(\{a, c, \neg b\}) \qquad Out(\Pi_2) = \{\neg c, b\}$$

Note: $In(\Pi)$, $Out(\Pi)$ are independent of the actual order of defaults in $\Pi$

# Default processes and extensions 1/3

**Definition**

$\Pi = (\delta_0, \ldots, \delta_m)$ is a *process* iff every $\delta_k$ is applicable in $In(\Pi[k])$.
In particular, $\delta_0$ must be applicable in $In(()) = Cn(W)$.

A process $\Pi$ is called

- *successful* iff $In(\Pi) \cap Out(\Pi) = \emptyset$;
- *not successful* (or *has failed*) iff $In(\Pi) \cap Out(\Pi) \neq \emptyset$;
- *closed* iff every $\delta \in \Delta$ that is applicable in $In(\Pi)$ appears in $\Pi$.

**Theorem**

*$E$ is an extension of $T$ iff there is a closed and successful process $\Pi$ with $E = In(\Pi)$.*

Proof.

$\Leftarrow$:

Let $\Pi$ be a closed and successful process of $T$ with $E = In(\Pi)$. We have to show $\Lambda_T(E) = E$:

- $\Lambda_T(E) \subseteq E$:
  $E$ is deductively closed and $W \subseteq E$; as $\Pi$ is closed, $E$ is closed wrt. default application. $\Lambda_T(E)$ is defined to be the *smallest* such set, so $\Lambda_T(E) \subseteq E$.
- $E \subseteq \Lambda_T(E)$:
  By induction $In(\Pi[k]) \subseteq \Lambda_T(E)$ for all $k$. It follows $E \subseteq \Lambda_T(E)$.

$\Rightarrow$:

Let $E = \Lambda_T(E)$ be an extension of $T$.
Let $\Delta = \{\delta_0, \ldots, \delta_n\}$ finite, arbitrary.
Construct process $\Pi$ of $T$ with $In(\Pi[k]) \subseteq E$ and $Out(\Pi[k]) \cap E = \emptyset$ as follows:

- $\Pi[0] = ()$;
- Let $\Pi[k]$ be constructed. If every default $\delta \in \Delta$ that is applicable in $In(\Pi[k])$ wrt. $E$ is already in $\Pi[k]$, define $\Pi = \Pi[k]$ and halt. Otherwise select any default $\delta$ that is applicable in $In(\Pi[k])$ wrt. $E$ and define $\Pi[k+1] = (\Pi[k], \delta)$.

In the end, $E = In(\Pi)$. $\square$

# Processes - examples

$T = (W, \Delta)$ with

$$W = \{a\}$$
$$\Delta = \{\delta_1 = \frac{a : \neg b}{\neg b}, \delta_2 = \frac{\top : c}{b}\}$$

- the process $\Pi_1 = (\delta_1)$
  - is successful: $In(\Pi_1) \cap Out(\Pi_1) = Cn(\{a, \neg b\}) \cap \{b\} = \emptyset$
  - is not closed as $\delta_2$ is applicable in $In(\Pi_1)$
- the process $\Pi_2 = (\delta_1, \delta_2)$
  - is not successful:
    $In(\Pi_2) \cap Out(\Pi_2) = Cn(\{a, \neg b, b\}) \cap \{b, \neg c\} = \{b\} \neq \emptyset$
  - is closed
- the process $\Pi_3 = (\delta_2)$
  - is successful: $In(\Pi_3) \cap Out(\Pi_3) = Cn(\{a, b\}) \cap \{\neg c\} = \emptyset$
  - is closed
  - $E = In(\Pi_3) = Cn(\{a, b\})$ is extension of $T$

# Process trees

Process trees give an overview on all possible processes of a default theory $T = (W, \Delta)$:

- every node represents a process $\Pi$ and is annotated with two labels: $In(\Pi)$ and $Out(\Pi)$;
- the root represents the empty process $\Pi = ()$ with $In(()) = Cn(W)$ and $Out(()) = \emptyset$;
- every application of a default induces a branch in the tree
- every leaf represents either
  - a failed process or
  - a closed and successful process

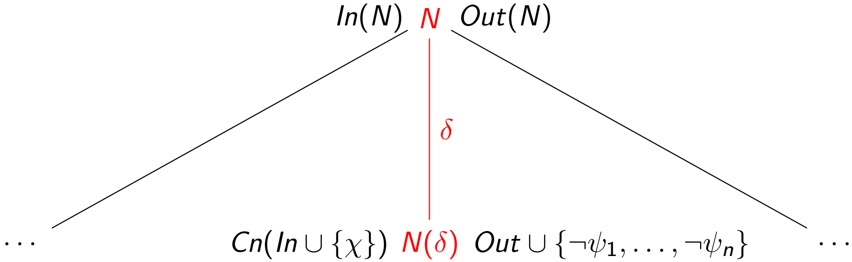
# Construction of process trees

1. Construct a tree with a root $N_0$ and $In(N_0) = Cn(W)$ and $Out(N_0) = \emptyset$
2. As long as there is a leaf node $N$ that is not marked with "failure" or "closed and successful", repeat
   - If $In(N) \cap Out(N) \neq \emptyset$: mark node with "failure"
   - $In(N) \cap Out(N) = \emptyset$
     - for every applicable default $\delta = \frac{\phi:\psi_1,\ldots,\psi_n}{\chi} \in \Delta$ that has not yet been considered in the process, add a child node $N(\delta)$ to $N$ with

$$In(N(\delta)) = Cn(In(N) \cup \{\chi\})$$
$$Out(N(\delta)) = Out(N) \cup \{\neg\psi_1, \ldots, \neg\psi_n\}$$

     - Is there no further applicable default that has not yet been considered, mark $N$ "closed and successful"; $In(N)$ is then an extension

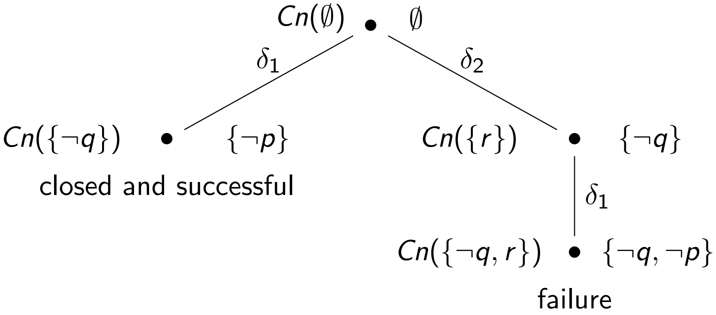
# Process tree




$$\delta = \frac{\varphi : \psi_1, \dots, \psi_n}{\chi}$$

# Process tree - example 1

$$T: \qquad W = \emptyset \qquad \Delta = \{\delta_1 = \frac{\top : p}{\neg q}, \delta_2 = \frac{\top : q}{r}\}$$



# Process tree - example 2

$$T: \quad W_0 = \{penguin \Rightarrow bird, penguin \Rightarrow \neg flies, bird\}$$

$$\Delta = \{\delta_1 = \frac{bird : flies}{flies}\}$$

$Cn(W_0)$ • $\emptyset$

$Cn(W_0 \cup \{flies\})$ • $\{\neg flies\}$

closed and successful

For $W_1 = W_0 \cup \{penguin\}$ note:

$Cn(W_1)$ is the only extension of $T_1 = (W_1, \Delta)$ as $\neg flies \in Cn(W_1)$.

# Properties of default logic 1/2

**Theorem (Minimality of extensions)**

*Let $E, E'$ be extensions of a default theory $T$ with $E \subseteq E'$. Then $E = E'$.*

**Theorem (Uniqueness of extensions)**

*Let $T = (W, \Delta)$ be a default theory a let*

$$W \cup \{\psi_1 \wedge \ldots \psi_n \wedge \chi \mid \frac{\phi : \psi_1, \ldots, \psi_n}{\chi} \in \Delta\}$$

*be classically consistent. Then $T$ has exactly one extension.*

# Properties of default logic 2/2

**Theorem (Inconsistency 1)**

*A default theory $T = (W, \Delta)$ has an inconsistent extension iff $W$ is already inconsistent.*

**Theorem (Inconsistency 2)**

*If $T$ has an inconsistent extension $E$ then $E$ is the only extension of $T$.*

$$T: \qquad W = \emptyset \qquad\qquad \Delta = \{\delta_0 = \frac{\top : a}{a}\}$$

$T$ has exactly one extension $E = Cn(\{a\})$.

Add defaults to $T$:

- $\Delta_1 = \{\delta_0, \delta_1 = \frac{\top : b}{\neg b}\}$. $T_1 = (W, \Delta_1)$ has *no extensions*
- $\Delta_2 = \{\delta_0, \delta_2 = \frac{b : c}{c}\}$. $T_2 = (W, \Delta_2)$ has still exactly one extension $E$
- $\Delta_3 = \{\delta_0, \delta_3 = \frac{\top : \neg a}{\neg a}\}$. $T_3 = (W, \Delta_3)$ has two extensions $E$ and $Cn(\{\neg a\})$.
- $\Delta_4 = \{\delta_0, \delta_4 = \frac{a : b}{b}\}$. $T_4 = (W, \Delta_4)$ has the extension $Cn(\{a, b\})$ which is a superset of $E$.

# Semi-Monotony 2/2

Extending a set of defaults (or the facts) can therefore

- remove extensions
- modify extensions
- create new extensions

Remember: this was the main motivation for default logic.

However, sometimes this behaviour may be to unpredictable.

### Definition

Let $T = (W, \Delta)$ and $T' = (W, \Delta')$ default theories with the same set of facts and defaults $\Delta \subseteq \Delta'$. If every extension of $T$ is contained in some extension of $T'$, then $T'$ is called a *semi-monotone* extension of $T$.

In general, we can also not expect a semi-monotonic behaviour in default logic.

# Normal defaults

### Definition

A default $\delta$ is called *normal* if $\mathit{just}(\delta) = \mathit{cons}(\delta)$, so $\delta$ is of the form

$$\delta = \frac{\phi : \psi}{\psi}$$

### Example

$$\frac{\mathit{bird} : \mathit{flies}}{\mathit{flies}}$$

Using a normal default we can conclude a formula $\psi$ if $\psi$ is consistent with the current beliefs.

# Processes of normal default theories

Let $T = (W, \Delta)$ be a normal default theory (=contains only normal defaults) with a consistent set of facts $W$.

Let $\Pi = (\delta_0, \ldots, \delta_n)$ be a process of $T$ with $\delta_i = \frac{\phi_i : \psi_i}{\psi_i}$.

Remember:

$$In(\Pi) = Cn(W \cup \{\psi_i\}_{i \geq 0})$$
$$Out(\Pi) = \{\neg\psi_i\}_{i \geq 0}$$

Can $\Pi$ be a failure?

Every default $\delta_i$ was applicable, so $\neg\psi_i \notin In(\Pi)$. It follows $In(\Pi) \cap Out(\Pi) = \emptyset$ and therefore:

**Theorem**

*Every process of a normal default theory is successful.*

# Extensions of normal default theories

**Theorem**

*A normal default theory always possesses at least one extension. Every finite process can be extended to a closed and successful process.*

**Theorem**

*Normal default theories are semi-monoton (adding another normal default extends previous extensions or preserves them completely).*

Chapter 4.1: Default logic

# Summary

- Default rules of the form

$$\delta = \frac{\phi : \psi_1, \ldots, \psi_n}{\chi}$$

  represent plausible (but not necessarily generally valid) rules
- Extensions: deductively closed, include facts, closed under default application
- Fix point characterisation of extensions
- Computing extensions with process trees
- Normal default theories and semi-monotony

# Overview

# Default logic and Prolog

- Default logic
  - ... is an expressive formalism for non-monotonic reasoning
  - ... is very formal and complex
  - therefore not very suitable for „practical knowledge representation“
- Recall Prolog
  - Practical programming language
  - Negation-as-failure `not`: similar to non-monotonic reasoning (if something cannot be proven, it is assumed to be false)
  - But there is no „logical negation“ in Prolog
  - Termination is not guaranteed
- Combine advantages of default logic and Prolog: *Answer set programming* (ASP)

# Recall: some notation from first-order logic

- $\Sigma = (U, P, F)$ first-order signature, $V$ set of variables
- In the following we consider only $F = \emptyset$
- A *literal* is an atom $p(t_1, \ldots, t_k)$ or the negation of an atom $\neg p(t_1, \ldots, t_k)$
- A literal $\phi$ is called *ground* if it mentions no variables

# Extended logic programs: syntax 1/2

Let $\Sigma = (U, P, \emptyset)$ be a first-order signature and $V$ a set of variables.

**Definition**

An *extended logic program* $P$ is a (finite) set of rules of the form

$$r: \qquad H \leftarrow A_1, \ldots, A_n, \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m.$$

with literals $H, A_1, \ldots, A_n, B_1, \ldots, B_m$ from $\mathcal{L}(\Sigma, V)$.

- `not` is called *default negation*
- $head(r) = \{H\}$ is called head of the rule $r$
- $pos(r) = \{A_1, \ldots, A_n\}$ positive body literals
- $neg(r) = \{B_1, \ldots, B_m\}$ negative body literals

# Negation-as-failure vs. classical negation

Why do we need two kinds of negation?

Compare

$cross\_tracks \leftarrow \texttt{not}\ train\_is\_coming.$

"We can cross the tracks when we don't know that a train is coming"

$cross\_tracks \leftarrow \neg train\_is\_coming.$

"We can cross the tracks when we know that a train is not coming"

and

$call\_doctor \leftarrow accident, \texttt{not}\ simulating.$

"We should call a doctor when there is an accident and we don't know that the person is simulating his injury"

$call\_doctor \leftarrow accident, \neg simulating.$

"We should call a doctor when there is an accident and we know that the person is not simulating his injury"

# Extended logic programs: syntax 2/2

General rule

$$r: \qquad H \leftarrow A_1, \ldots, A_n, \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m.$$

*Special cases*:

- $n = m = 0$: Rule with empty body (=fact)

$$H \leftarrow . \qquad \text{or simply} \qquad H.$$

- All literals are atoms: normal logic rule
- Empty head literal ($head(r) = \emptyset$): constraint

$$\leftarrow A_1, \ldots, A_n, \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m.$$

# Example

$$
\begin{aligned}
&\mathit{bird}(X) \leftarrow \mathit{penguin}(X).\\
&\mathit{flies}(X) \leftarrow \mathit{bird}(X), \texttt{not}\ \neg\mathit{flies}(X).\\
&\neg\mathit{flies}(X) \leftarrow \mathit{penguin}(X).
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{flies}(X) \leftarrow \mathit{bat}(X).\\
&\leftarrow \mathit{bird}(X), \mathit{bat}(X).
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{penguin}(\mathit{tweety}).\\
&\mathit{bat}(\mathit{batman}).
\end{aligned}
$$

# Grounding of extended logic programs

- An extended logic program with variables is always interpreted as a schema for its instances
- *Grounding* a program means substituting all variables by constants in all combinations

Example

For $U = \{a, b\}$ the program $P = \{p(X) \leftarrow t(X, Y), \texttt{not}\ r(Y).,$ $r(a).\}$ is a shorthand for $\mathit{ground}(P)$:

$$
\begin{aligned}
& p(a) \leftarrow t(a, a), \texttt{not}\ r(a). \\
& p(b) \leftarrow t(b, b), \texttt{not}\ r(b). \\
& p(a) \leftarrow t(a, b), \texttt{not}\ r(b). \\
& p(b) \leftarrow t(b, a), \texttt{not}\ r(a). \\
& r(a).
\end{aligned}
$$

$\rightarrow$ it suffices to consider only propositional literals.

# States 1/2

- Literals $p$ and $\neg p$ are called *complementary*
- For a literal $l$, the complementary literal is denoted $\bar{l}$
- So $\bar{a} = \neg a$ and $\overline{\neg a} = a$ for an atom $a$
- A set of ground literals $S$ is called *consistent* iff it contains no pair of complementary literals

## Definition

A *state* is a consistent set of ground literals.

Conceptually, states of extended logic programs correspond to classical interpretations of classical formulas.

# States 2/2

## Example

Consider the extended logic program $P$:

$$p \leftarrow q, \texttt{not } r.$$
$$q \leftarrow \texttt{not } s.$$
$$s.$$

Some states for $P$ are:

- $Z_1 = \{p, q, \neg r\}$
- $Z_2 = \{\neg p, s\}$
- $Z_3 = \{s, q, p\}$

**Question**: When does a state describe a program "in a meaningful way"?

# Closed states

Let $P$ be an extended logic program *without* default negation.
Let $S$ be a state.

### Definition

$S$ is *closed* under $P$ iff for every rule $r \in P$, if $pos(r) \subseteq S$ then $head(r) \cap S \neq \emptyset$.

- For rules of the form $r : H \leftarrow A_1, \dots, A_n$ this means:

  If $\{A_1, \dots, A_n\} \subseteq S$ then $H \in S$.

- For constraints of the form $r :\leftarrow A_1, \dots, A_n$ this means:

  As $head(r) = \emptyset$ it has to hold $head(r) \cap S = \emptyset$; therefore $\{A_1, \dots, A_n\} \subseteq S$ must not be true.

- For facts $r : H \leftarrow$ this means:

  As $pos(r) = \emptyset$ we always have $pos(r) \subseteq S$; every closed state must contain all facts.

# Example

Consider the extended logic program $P$ (without default negation):

$$p \leftarrow q, r.$$

$$q \leftarrow \neg s.$$

$$\neg s.$$

The following states are not closed:

- $Z_1 = \emptyset$
- $Z_2 = \{\neg s\}$
- $Z_3 = \{r, p\}$

The following states are closed:

- $Z_4 = \{\neg s, q\}$
- $Z_5 = \{\neg s, q, p\}$
- $Z_6 = \{\neg s, q, p, r\}$

**Question**: are $Z_5$ and $Z_6$ meaningful?

# Minimal models

Let $P$ be an extended logic program *without* default negation.
Let $S$ be a state.

**Definition**

$S$ is a *minimal model* of $P$ iff $S$ is closed and for every closed state $S'$ for $P$, $S \subseteq S'$.

**Example**

Consider again $P$:

$$p \leftarrow q, r.$$
$$q \leftarrow \neg s.$$
$$\neg s.$$

Here $Z_4 = \{\neg s, q\}$ is the (only) minimal model of $P$.

# Existence and uniqueness of minimal models 1/2

Let $P$ be an extended logic program *without* default negation.

### Definition

$P$ is called *consistent* iff there is a closed state of $P$.

- $P_1 = \{s., \neg s.\}$ is not consistent (every closed state $S$ would contain both $s$ and $\neg s$; but a set containing complementary literals is not a state)
- $P_2 = \{s., \neg r \leftarrow s., r \leftarrow s.\}$ is not consistent.

### Theorem

*Let $P$ be a consistent extended logic program without default negation. Then $P$ has exactly one minimal model.*

# Existence and uniqueness of minimal models 2/2

## Proof.

We have to show that there is at least one minimal model and at most one minimal model

- $\geq 1$: As $P$ is consistent, there is a closed state $S$. If $S$ is minimal: finished. If not, there is a another closed set $S' \subset S$. As $P$ is finite there is a finite number of states and this sequence must end in a minimal model.
- $\leq 1$: Assume there are two different minimal models $M_1, M_2$. Then $M_1 \not\subseteq M_2$ and $M_2 \not\subseteq M_1$ (otherwise one of them would not be minimal). We now show that $M_3 = M_1 \cap M_2$ is also closed:
  - Let $r \in P$ with $head(r) = H$ (analogous for constraints). If $pos(r) \subseteq M_3$ then $pos(r) \subseteq M_1$ and $pos(r) \subseteq M_2$. As $M_1$ and $M_2$ are closed, $H \in M_1$ and $H \in M_2$. Therefore $H \in M_3$.

  As $M_3$ is closed, neither $M_1$ nor $M_2$ can be minimal (as $M_3 \subset M_1$ and $M_3 \subset M_2$). □

Let $P$ be an extended logic program *without* default negation.

### Definition

For a set $X$ of ground literals define

$$\Lambda_P(X) = \{head(r) \mid r \in P, pos(r) \subseteq X\}$$

### Example

Consider again $P = \{p \leftarrow q, r., q \leftarrow \neg s., \neg s.\}$. Then

- $\Lambda_P(\{r, q\}) = \{p, \neg s\}$
- $\Lambda_P(\{\neg s\}) = \{\neg s, q\}$
- $\Lambda_P(\emptyset) = \{\neg s\}$

Define also $\Lambda_P^1(X) = \Lambda_P(X)$ and $\Lambda_P^{n+1}(X) = \Lambda_P(\Lambda_P^n(X))$.

# Characterisation of minimal models 2/5

**Theorem**

*Let $P$ be a consistent extended logic program without default negation. A state $S$ is closed under $P$ iff $S \supseteq \Lambda_P(S)$.*

**Proof.**

Let $S$ be a state. If $S$ is closed, there are no rule in $P$ that can be applied. Therefore, all head literals $H$ of all rules applicable in $S$ are already in $S$. This is equivalent to $S \supseteq \Lambda_P(S)$. □

# Characterisation of minimal models 3/5

**Theorem**

*Let $P$ be a consistent extended logic program without default negation. Then there is a finite $k \geq 0$ with*

$$\emptyset \subseteq \Lambda_P^1(\emptyset) \subseteq \Lambda_P^2(\emptyset) \subseteq \ldots \subseteq \Lambda_P^k(\emptyset) = \Lambda_P^{k+1}(\emptyset) = \ldots \tag{1}$$

*and $\Lambda_P^k(\emptyset)$ is the minimal model of $P$.*

**Proof.**

We first show (1):

We show this by induction.

- The base case $\emptyset \subseteq \Lambda_P^1(\emptyset)$ is obviously true.

- Assume $\emptyset \subseteq \Lambda_P^1(\emptyset) \subseteq \Lambda_P^2(\emptyset) \subseteq \ldots \subseteq \Lambda_P^i(\emptyset)$. We have to show that $\Lambda_P^i(\emptyset) \subseteq \Lambda_P^{i+1}(\emptyset)$:
  Let $H \in \Lambda_P^i(\emptyset)$. Then there is $l \leq i$ such that $H \in \Lambda_P^l(\emptyset)$ but $H \notin \Lambda_P^{l-1}(\emptyset)$. Let $r : H \leftarrow A_1, \ldots, A_n \in P$ be a rule that caused $H \in \Lambda_P^l(\emptyset)$. Therefore $\{A_1, \ldots, A_n\} \subseteq \Lambda_P^{l-1}(\emptyset)$. As $\Lambda_P^{l-1}(\emptyset) \subseteq \Lambda_P^i(\emptyset)$ we also have $\{A_1, \ldots, A_n\} \in \Lambda_P^i(\emptyset)$. So we can apply $r$ in calculating $\Lambda_P^{i+1}(\emptyset)$ and therefore $H \in \Lambda_P^{i+1}(\emptyset)$. Hence we get $\Lambda_P^i(\emptyset) \subseteq \Lambda_P^{i+1}(\emptyset)$.

It is also clear that this chain must end in a fixed point (the set of literals is finite), i. e., $\Lambda_P^k(\emptyset) = \Lambda_P^{k+1}(\emptyset)$.

We show now that $\Lambda_P^k(\emptyset)$ is a minimal model:

First, $M = \Lambda_P^k(\emptyset)$ is closed (as $M \supseteq \Lambda_P(M)$). Assume there is $M' \subset M$ such that $M'$ is closed. Let $i \geq 0$ be the smallest index with

$$\Lambda_P^i(\emptyset) \subseteq M' \qquad \text{und}$$
$$\Lambda_P^{i+1}(\emptyset) \not\subseteq M'$$

Then there is $H \in \Lambda_P^{i+1}(\emptyset) \setminus M'$ such that there is a $r : H \leftarrow A_1, \dots, A_n \in P$ that was applicable when calculating $\Lambda_P^{i+1}(\emptyset)$, so $\{A_1, \dots, A_n\} \subseteq \Lambda_P^i(\emptyset)$. Therefore $\{A_1, \dots, A_n\} \subseteq M'$ as well and as $H \notin M'$, $M'$ cannot be closed. $\square$

# Gelfond-Lifschitz-Reduct 1/3

...back to the general case of extended logic programs $P$ *with* default negation.

- Idea: Simplify $P$ to a program $P'$ without default negation
- Compute the minimal model $M$ of $P'$ and call $M$ *answer set* of $P$
- More specifically:
  1. „Guess“ a state $S$ that could be an answer set
  2. Simplify $P$ using $S$
  3. Compute the minimal model of the simpler program; if this turns out to be $S$ again then $S$ is an answer set

# Gelfond-Lifschitz-Reduct 2/3

### Definition

Let $P$ be an extended logic program (with default negation) and $S$ a state. The *reduct* $P^S$ of $P$ wrt. $S$ is a logic program defined as

$$P^S = \{H \leftarrow A_1, \ldots, A_n. \mid \\ H \leftarrow A_1, \ldots, A_n, \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m. \in P, \\ \{B_1, \ldots, B_m\} \cap S = \emptyset\}$$

The reduct $P^S$ is constructed from $P$ in two steps:

1. All rules that contain some $\texttt{not}\ B$ with $B \in S$ in their body are removed.
2. For the remaining rules, all negative body literals are removed.

# Gelfond-Lifschitz-Reduct 3/3

Observations:

- $P^S$ looks different, depending on $S$
- $P^S$ is an extended logic program without default negation
- $P^S$ always contains
  - All facts from $P$
  - All rules without default negation

**Remark**: the reduct is also called the Gelfond-Lifschitz-Reduct after

*Michael Gelfond, Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. In New Generation Computing 9:365–385, 1991.*

# Example

Consider the following program $P$

$$
\begin{aligned}
& p \leftarrow \texttt{not}\ r. \\
& r \leftarrow \neg q, \texttt{not}\ b. \\
& \neg q \leftarrow b. \\
& b.
\end{aligned}
$$

and states $S_1$ and $S_2$

$$S_1 = \{r\} \qquad\qquad S_2 = \{b, \neg q, p\}$$

Then

$$
\begin{aligned}
P^{S_1} &= \{r \leftarrow \neg q.\ ,\ \neg q \leftarrow b.\ ,\ b.\} \\
P^{S_2} &= \{p.\ ,\ \neg q \leftarrow b.\ ,\ b.\}
\end{aligned}
$$

# Answer sets

## Definition

Let $P$ be an extended logic program. A state $S$ is an *answer set* of $P$ iff $S$ is the minimal model of $P^S$.

## Example

As before:

$$P = \{p \leftarrow \texttt{not}\ r.\ \ r \leftarrow \neg q, \texttt{not}\ b.\ \ \neg q \leftarrow b.\ \ b.\}$$

$$S_1 = \{r\}$$
$$S_2 = \{b, \neg q, p\}$$

$$P^{S_1} = \{r \leftarrow \neg q.\ ,\ \neg q \leftarrow b.\ ,\ b.\}$$
$$P^{S_2} = \{p.\ ,\ \neg q \leftarrow b.\ ,\ b.\}$$

- Minimal model of $P^{S_1}$ is $\{b, \neg q, r\} \neq S_1$.
- Minimal model of $P^{S_2}$ is $\{b, \neg q, p\} = S_2$, hence $S_2$ is an answer set of $P$.

# Another example 1/3

Consider the extended logic program $P$:

$$p(X) \leftarrow \texttt{not } q(X).$$
$$q(X) \leftarrow r(X), \texttt{not } p(X).$$
$$r(a).$$

Let $U = \{a, b\}$. Grounding $P_g = ground(P)$ of $P$:

$$p(a) \leftarrow \texttt{not } q(a).$$
$$p(b) \leftarrow \texttt{not } q(b).$$
$$q(a) \leftarrow r(a), \texttt{not } p(a).$$
$$q(b) \leftarrow r(b), \texttt{not } p(b).$$
$$r(a).$$

# Another example 2/3

$$
\begin{aligned}
P_g :\ & p(a) \leftarrow \texttt{not}\ q(a).\\
& p(b) \leftarrow \texttt{not}\ q(b).\\
& q(a) \leftarrow r(a), \texttt{not}\ p(a).\\
& q(b) \leftarrow r(b), \texttt{not}\ p(b).\\
& r(a).
\end{aligned}
$$

Assumption: $S_1 = \{r(a), q(a), p(b)\}$ is answer set. Compute reduct:

$$
\begin{aligned}
P_g^{S_1} :\ & p(b).\\
& q(a) \leftarrow r(a).\\
& r(a).
\end{aligned}
$$

Minimal model of $P_g^{S_1}$ is $S_1 \longrightarrow$ answer set.

# Another example 3/3

$$
\begin{aligned}
P_g : & p(a) \leftarrow \texttt{not } q(a). \\
& p(b) \leftarrow \texttt{not } q(b). \\
& q(a) \leftarrow r(a), \texttt{not } p(a). \\
& q(b) \leftarrow r(b), \texttt{not } p(b). \\
& r(a).
\end{aligned}
$$

Assumption: $S_2 = \{r(a), p(a), p(b)\}$ is answer set. Compute reduct:

$$
\begin{aligned}
P_g^{S_2} : & p(a). \\
& p(b). \\
& r(a).
\end{aligned}
$$

Minimal model of $P_g^{S_2}$ is $S_2 \longrightarrow$ answer set.

# Answer sets and default extensions

Let $P$ be an extended logic program.

For every rule $r : H \leftarrow A_1, \ldots, A_n, \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m$
define the default

$$def(r) = \frac{A_1 \wedge \ldots \wedge A_n : \overline{B_1}, \ldots, \overline{B_m}}{H}$$

and $def(P) = (\emptyset, \{def(r) \mid r \in P\})$ as default theory wrt. $P$.

**Theorem (Gelfond, Lifschitz, 1991)**

*If $S$ is answer set of $P$ then $Cn(S)$ is an extension of $def(P)$. If $E$ is an extension of $def(P)$ then there is an answer set $S$ von $P$ with $E = Cn(S)$.*

Chapter 4.2: Answer set programming

# Summary

- extended logic programs contain rules of the form

$$r: \qquad H \leftarrow A_1, \ldots, A_n, \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m.$$

- grounding of first-order rules
- states, closed states
- minimal models of programs without default negation
- Gelfond-Lifschitz-Reduct and answer sets
- Answer sets and default extensions