AI-KI-B

Resolution Calculus and Prolog

Ute Schmid & Diedrich Wolter

Practice: Johannes Rabold & student assistants

Cognitive Systems and Smart Environments
Applied Computer Science, University of Bamberg

last change: 13. Juni 2020

Reasoning in First Order Logic

- Logic as a formal language for automated reasoning
- Propositional logic ('Ausagenlogik'): Atomic formula are propositions which evaluate to true or false, proof of truth of a formula by truth table based on the semantics of junctors
- First order logic ('Pädikatenlogik erster Stufe'): More expressive, allows for predicates over terms, only semi-decidable
- One of the most influential calculi for first order logic: resolution calculus
  - Resolution was introduced by Robinson (1965) as a mechanic way (a calculus) to perform logical proofs.
  - Logical formula must be rewritten into **clause form**, using **equivalence rules**.
  - To perform a **resolution step** on a pair of clauses, literals must be **unified**.

Outline

- Semantic Equivalence
- Clausal Form
- Substitution and Unification
- Proofs by Resolution
- Prolog and SLD-Resolution
- Applications of Resolution
- Reasoning and Inference

Semantic Equivalence

- Two formulas F and G are called **equivalent**, if for each interpretation of G and F holds that G is valid iff F is valid. We write $F \equiv G$.
- Theorem: Let be $F \equiv G$. Let H be a formula where F appears as a sub-formula. Let H' be a formula derived from H by replacing F by G. Then it holds $H \equiv H'$.
- Equivalences can be used to rewrite formulas.

'iff' is an abbreviation for 'if and only if' (genau dann wenn)

$(F \land F) \equiv F$, $(F \lor F) \equiv F$	(idempotency)
$(F \land G) \equiv (G \land F)$, $(F \lor G) \equiv (G \lor F)$	(commutativity)
$((F \land G) \land H) \equiv (F \land (G \land H))$, $((F \lor G) \lor H) \equiv (F \lor (G \lor H))$	(associativity)
$(F \land (F \lor G)) \equiv F$, $(F \lor (F \land G)) \equiv F$	(absorption)
$(F \land (G \lor H)) \equiv ((F \land G) \lor (F \land H))$, $(F \lor (G \land H)) \equiv ((F \lor G) \land (F \lor H))$	(distributivity)
$\neg\neg F \equiv F$	(double negation)
$\neg(F \land G) \equiv (\neg F \lor \neg G)$, $\neg(F \lor G) \equiv (\neg F \land \neg G)$	(de Morgan)
$(F \to G) \equiv (\neg F \lor G)$	(remove implication)

$F \lor \neg F \equiv \text{true}$	(tautology)
$F \land \neg F \equiv \text{false}$	(contradiction)

Remark: This is the *tertium non datur* principle of classical logic.

$\neg\forall x\ F \equiv \exists x\ \neg F$, $\neg\exists x\ F \equiv \forall x\ \neg F$

$(F \vee G) \equiv F$, if F tautology;
$(F \wedge G) \equiv G$, if F tautology
$(F \vee G) \equiv G$, if F contradiction;
$(F \wedge G) \equiv F$, if F contradiction

If x is not free in G it holds:
$(\forall x\ F \wedge G) \equiv \forall x\ (F \wedge G)$, $(\forall x\ F \vee G) \equiv \forall x\ (F \vee G)$,
$(\exists x\ F \wedge G) \equiv \exists x\ (F \wedge G)$, $(\exists x\ F \vee G) \equiv \exists x\ (F \vee G)$

$(\forall x\ F \wedge \forall x\ G) \equiv \forall x\ (F \wedge G)$, $(\exists x\ F \vee \exists x\ G) \equiv \exists x\ (F \vee G)$

$\forall x \forall y F \equiv \forall y \forall x F$, $\exists x \exists y F \equiv \exists y \exists x F$

- **Conjunctive Normal Form (CNF)**:
  Conjunction of disjunctions of literals

$$\wedge_{i=1}^{n}(\vee_{j=1}^{m} L_{ij})$$

- **Clause Form**:
  Set of disjunctions of literals (can be generated from CNF)

Rewriting of formulas to clause form:

8 steps, illustrated with example

$$\forall x[B(x) \rightarrow (\exists y[O(x,y) \wedge \neg P(y)] \wedge \neg \exists y[O(x,y) \wedge O(y,x)] \wedge \forall y[\neg B(y) \rightarrow \neg E(x,y)])]$$

(0) Original Formula

$\forall x[B(x) \rightarrow (\exists y[O(x,y) \wedge \neg P(y)] \wedge \neg\exists y[O(x,y) \wedge O(y,x)] \wedge \forall y[\neg B(y) \rightarrow \neg E(x,y)])]$

(1) Remove Implications

$\forall x[\neg B(x) \vee (\exists y[O(x,y) \wedge \neg P(y)] \wedge \neg\exists y[O(x,y) \wedge O(y,x)] \wedge \forall y[\neg(\neg B(y)) \vee \neg E(x,y)]\ )\ ]$

(2) Reduce scopes of negation

$\forall x[\neg B(x) \vee (\exists y[O(x,y) \wedge \neg P(y)] \wedge \forall y[\neg O(x,y) \vee \neg O(y,x)] \wedge \forall y[B(y) \vee \neg E(x,y)]\ )\ ]$

(3) Skolemization (remove existential quantifiers)

Replace existentially quantified variables by constant/function symbols.

$$\exists x\ p(x) \text{ becomes } p(C)$$

("There exists a human who is a student." is satisfiable if there exists a constant in the universe $\mathcal{U}$ for which the sentence is true.

"Human C is a student." is satisfiable if the constant symbol C can be interpreted such that relation p is true.)

Skolemization cont.

If an existentially quantified variable is in the scope of a universally quantified variable, it is replaced by a function symbol dependent of this variable:

$$\forall x\ \exists y\ p(x) \wedge q(x, \mathbf{y}) \text{ becomes } \forall x\ p(x) \wedge q(x, \mathbf{f(x)})$$

("For all x holds, x is a positive integer and there exists a y which is greater than x." is satisfiable if for each x exists an y such that the relation "greater than" holds. E.g., $f(x)$ can be interpreted as successor-function.)

Skolemization is **no equivalence transformation**. *A formula and its Skolemization are only equivalent with respect to satisfiability!*
The skolemized formula has a model iff the original formula has a model.

$$\forall x[\neg B(x) \vee ((O(x, f(x)) \wedge \neg P(f(x))) \wedge \forall y[\neg O(x, y) \vee \neg O(y, x)] \wedge \forall y[B(y) \vee \neg E(x, y)]\ )\ ]$$

(4) Standardize variables ("bounded renaming")

A variable bound by a quantifier is a "dummy" and can be renamed. Provide that each variable of universal quantifier has a different name. (Problematic case: free variables)

$$\forall \mathbf{x}[\neg B(x) \vee ((O(x, f(x)) \wedge \neg P(f(x))) \wedge \forall \mathbf{y}[\neg O(x, y) \vee \neg O(y, x)] \wedge \forall \mathbf{z}[B(z) \vee \neg E(x, z)]\ )\ ]$$

(5) Prenex-form

Move universal quantifiers to front of the formula.

$$\forall x \forall y \forall z[\neg B(x) \vee ((O(x, f(x)) \wedge \neg P(f(x))) \wedge (\neg O(x, y) \vee \neg O(y, x)) \wedge (B(z) \vee \neg E(x, z))\ )\ ]$$

(6) CNF

(Repeatedly apply the distributive laws)

$$\forall x \forall y \forall z[(\neg B(x) \vee O(x, f(x))) \wedge (\neg B(x) \vee \neg P(f(x))) \wedge (\neg B(x) \vee \neg O(x, y) \vee \neg O(y, x)) \\ \wedge (\neg B(x) \vee B(z) \vee \neg E(x, z))\ ]$$

(7) Eliminate Conjunctions

If necessary, rename variable such that each disjunction has a different set of variables.

The truth of a conjunction entails that all its parts are true.

$$\forall \mathbf{x}[\neg B(x) \vee O(x, f(x))],\ \forall \mathbf{w}[\neg B(w) \vee \neg P(f(w))],\ \forall \mathbf{u}\ \forall \mathbf{y}[\neg B(u) \vee \neg O(u, y) \vee \neg O(y, u)],\ \forall \mathbf{v}\ \forall \mathbf{z}[\neg B(v) \vee B(z) \vee \neg E(v, z)]$$

(8) Eliminate Universal Quantifiers

Clauses are implicitly universally quantified.

$M =$

$\{\neg B(x) \vee O(x, f(x)), \neg B(w) \vee \neg P(f(w)), \neg B(u) \vee \neg O(u, y) \vee \neg O(y, u), \neg B(v) \vee B(z) \vee \neg E(v, z)\}$

- A **substitution** is a set

$$\theta = \{v_1 \leftarrow t_1, \ldots v_n \leftarrow t_n\}$$

  of replacements of **variables** v_i by **terms** t_i.
- If θ is a substitution and E an expression, $E' = E\theta$ is called **instance** of E.
  E' was derived from E by applying θ to E.

Example

- $E = p(x) \vee (\neg q(x, y) \wedge p(f(x)))$
- $\theta = \{x \leftarrow C\}$
- $E\theta = p(C) \vee (\neg q(C, y) \wedge p(f(C)))$

- Special case: alphabetic substitution (variable renaming).

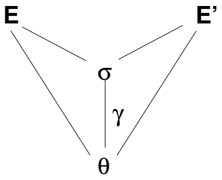
Composition of Substitutions

- Let be
  $\theta = \{u_1 \leftarrow t_1, \ldots u_n \leftarrow t_n, v_1 \leftarrow s_1, \ldots v_k \leftarrow s_k\}$ and
  $\sigma = \{v_1 \leftarrow r_1, \ldots v_k \leftarrow r_k, w_1 \leftarrow q_1, \ldots w_m \leftarrow q_m\}$.

- The composition is defined as
  $\theta\sigma =_{Def} \{u_1 \leftarrow t_1\sigma, \ldots u_n \leftarrow t_n\sigma, v_1 \leftarrow s_1\sigma, \ldots v_k \leftarrow s_k\sigma, w_1 \leftarrow q_1, \ldots w_m \leftarrow q_m\}$

- Composition of substitutions is not commutative!

- Let be $\{E_1 \ldots E_n\}$ a set of expressions. A substitution θ is a **unificator** of $E_1 \ldots E_n$, if $E_1\theta = E_2\theta \ldots = E_n\theta$.
- A unificator θ is called **most general unifier** (mgu), if for each other unificator σ for $E_1 \ldots E_n$ there exists a substitution γ with $\sigma = \theta\gamma$.
- Theorem: If a unificator exists, then also an mgu exists.



There are different unification algorithms, e.g. one proposed by Robinson.

(1)	$\{P(x), P(A)\}$	$\theta = \{x \leftarrow A\}$
(2)	$\{P(f(x), y, g(y)), P(f(x), z, g(x))\}$	$\theta = \{y \leftarrow x, z \leftarrow x\}$
(3)	$\{P(f(x, g(A, y)), g(A, y)), P(f(x, z), z)\}$	$\theta = \{z \leftarrow g(A, y)\}$
(4)	$\{P(x, f(y), B), P(x, f(B), B)\}$	$\sigma = \{x \leftarrow A, y \leftarrow B\}$
		$\theta = \{y \leftarrow B\}$

In (4) holds:

θ is more general than σ: $\sigma = \theta\gamma$, with $\gamma = \{x \leftarrow A\}$

θ is mgu for $\{P(x, f(y), B), P(x, f(B), B)\}$

For a given set of formula S:

1. Let be $\theta = \{\}$
2. While $|S| > 1$ DO
    1. Calculate the disagreement set D of S
    2. If D contains a variable x and a term t in which x does not occur
    Then $\theta = \theta\{x \leftarrow t\}$ and $S = S\theta$
    Else stop (S not unifiable)
3. Return θ as mgu of S

- Since S is the set of all formula, it has size one if all formula are identical (unified by θ).
- For calculation of disagreement set see practice

- A formula G is called **logical consequence** (or entailment) of a set of formula $F = \{F_1 \ldots F_n\}$, if each model of F is also a model of G.
  Note:
  We write $\mathcal{A} \models G$ to denote the "*model relation*" and also $F \models G$ to denote the "*entailment relation*".
- **The following propositions are equivalent**:
  1. G is a logical consequence of F.
  2. $(\wedge_{i=1}^{n} F_i) \rightarrow G$ is valid (tautology).
  3. $(\wedge_{i=1}^{n} F_i) \wedge \neg G$ is not satisfiable (a contradiction).

  This relation between logical consequences and syntactical expressions can be exploited for syntactical proofs. We write $F \vdash G$ if formula G can be **derived** from the set of formulas F.

- The resolution calculus consists of a single rule (and does not possess any axioms).
- Resolution is defined for **clauses** (each formula is a disjunction of positive and negative literals).
- All formulas must hold: conjunction of clauses.
- **Proof by contradiction**, exploiting the equivalence given above. If

$$(\bigwedge_{i=1}^{n} F_i) \land \neg G$$

is not satisfiable, then "false" (the empty clause) can be derived:

$$[(\bigwedge_{i=1}^{n} F_i) \land \neg G] \vdash \Box$$

Resolution rule in propositional logic:

$$(P \vee P_1 \vee \dots P_n) \wedge (\neg P \vee Q_1 \vee \dots Q_m) \vdash (P_1 \vee \dots P_n \vee Q_1 \vee \dots Q_m)$$

Resolution rule for clauses:

$$[(L \vee C_1) \wedge (\neg L \vee C_2)]\sigma \vdash [C_1 \vee C_2]\sigma$$

(σ is a substitution of variables such that L is identical in both parts of the conjunction)

The general idea is to cut out a literal which appears positive in one disjunction and negative in the other.

Example

Theory:

All humans are mortal.	$F_1 = \text{Human} \rightarrow \text{Mortal}$
Socrates is a human.	$F_2 = \text{Human}$

Query:

Socrates is mortal: $G = \text{Mortal}$

To prove $F_1 \wedge F_2 \wedge \neg G \vdash \Box$, we need the following resolution steps:

1. $\text{Human} \rightarrow \text{Mortal} \equiv \neg\text{Human} \vee \text{Mortal}$
2. $\text{Human} \wedge [\neg\text{Human} \vee \text{Mortal}] \wedge \neg\text{Mortal}$
3. $\vdash \text{Mortal} \wedge \neg\text{Mortal}$
4. $\vdash \Box$.

Example

Theory:

All humans are mortal. $F_1 = \forall x \text{ Human(x)} \rightarrow \text{Mortal(x)}$
Socrates is a human. $F_2 = \text{Human(S)}$

Query:

Socrates is mortal: $G = \text{Mortal(S)}$

To prove $F_1 \wedge F_2 \wedge \neg G \vdash \Box$, we need the following resolution steps:

1. $\forall x \text{ Human(x)} \rightarrow \text{Mortal(x)} \equiv \forall x \neg\text{Human(x)} \vee \text{Mortal(x)}$
   (substitute S for universally quantified variable x)
2. $[\text{Human(S)} \wedge [\neg\text{Human(x)} \vee \text{Mortal(x)}] \wedge \neg\text{Mortal(S)}]\{x \rightarrow S\}$
3. $\vdash \text{Mortal(S)} \wedge \neg\text{Mortal(S)}$
4. $\vdash \Box$.

A clause

$$C = \bigvee_{i=1}^{n} L_i$$

can be written as set

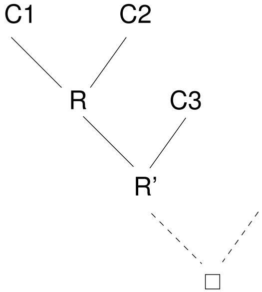
$$C = \{L_1, \dots L_n\}.$$

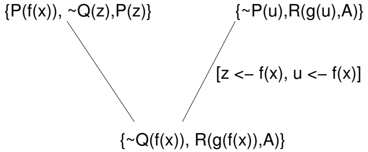
Let be C_1, C_2 and R clauses. R is called **resolvent** of C_1 and C_2 if:

- There are alphabetical substitutions σ_1 and σ_2 such that $C_1\sigma_1$ and $C_2\sigma_2$ have no common variables.
- There exists a set of literals $L_1, \dots L_m \in C_1\sigma_1 (m \geq 1)$ and $L'_1, \dots L'_n \in C_2\sigma_2 (n \geq 1)$ such that $L = \{\neg L_1, \neg L_2, \dots \neg L_m, L'_1, L'_2, \dots L'_n\}$ are unifiable with θ as mgu of L.
- R has the form:

$$R = ((C_1\sigma_1 \setminus \{L_1, \dots L_m\}) \cup (C_2\sigma_2 \setminus \{L'_1, \dots L'_n\}))\theta.$$

Derivation of a clause by application of the resolution rule can be described by a **refutation tree**:





$C_1 = \{P(f(x)), \neg Q(z), P(z)\}$
$C_2 = \{\neg P(x), R(g(x), A)\}$

$\sigma_1 = \{\}$, $\sigma_2 = \{x \leftarrow u\}$

$L = \{P(f(x)), P(z), \neg\neg P(x)\} = \{P(f(x)), P(z), P(u)\}$

$\theta = \{z \leftarrow f(x), u \leftarrow f(x)\}$

$$R = [(\{P(f(x)), \neg Q(z), P(z)\} \setminus \{P(f(x)), P(z)\}) \cup (\{\neg P(u), R(g(u), A)\} \setminus \{P(u)\})]\theta = \{\neg Q(f(x)), R(g(f(x)), A)\}$$

- To prove that formula G (assertion) logically follows from a set of formula (axioms) $F_1 \dots F_n$:

Resolution Proof Strategy

1. Include the negated assumption in the set of axioms.
2. Try to derive a contradiction (empty clause).

- Theorem:
  A set of clauses is not satisfiable, if the empty clause ($\square$) can be derived with a resolution proof.
- Contradiction:
  $C_1 = A, C_2 = \neg A$, stands for $(A \wedge \neg A)$ and $(A \wedge \neg A) \vdash \square$

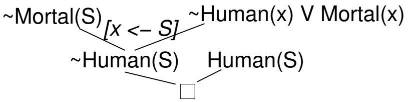
- Axiom: “All humans are mortal” Fact: “Socrates is human”
  (Both are non-logical: their truth is presupposed)
- Assertion

  “Socrates is mortal.”
- Formalization:

$$F_1 : \quad \forall x : \text{Human}(x) \rightarrow \text{Mortal}(x)$$
$$F_2 : \quad \text{Human}(S)$$
$$F_3 : \quad \neg\text{Mortal}(S) \quad \text{(negation of assertion)}$$

- Clause form:

$$F_1' : \quad \neg\text{Human}(x) \lor \text{Mortal}(x)$$
$$F_2' : \quad \text{Human(S)}$$
$$F_3' : \quad \neg\text{Mortal}(S)$$



Soundness and Completeness of Res.

- A calculus is **sound**, if only such conclusions can be derived which also hold in the model.
- A calculus is **complete**, if all conclusions can be derived which hold in the model.
- The resolution calculus is sound and refutation complete.

  Refutation completeness means, that if a set of formula (clauses) is unsatisfiable, then resolution will find a contradiction. Resolution cannot be used to generate all logical consequences of a set of formula, but it can establish that a given formula is entailed by the set. Hence, it can be used to find all answers to a given question, using the "negated assumption" method.

Remarks

- The proof ideas will given for resolution for propositional logic (or ground clauses) only.
- For FOL, additionally, a lifting lemma is necessary and the proofs rely on Herbrand structures.
- We cover elementary concepts of logic only.
- For more details, see
  - Ghallab, Nau, & Traverso, Appendix B and chapter 12
  - Uwe Schöning, Logik für Informatiker, 5. Auflage, Spektrum, 2000.
  - Volker Sperschneider & Grigorios Antoniou, Logic – A foundation for computer science, Addison-Wesley, 1991.

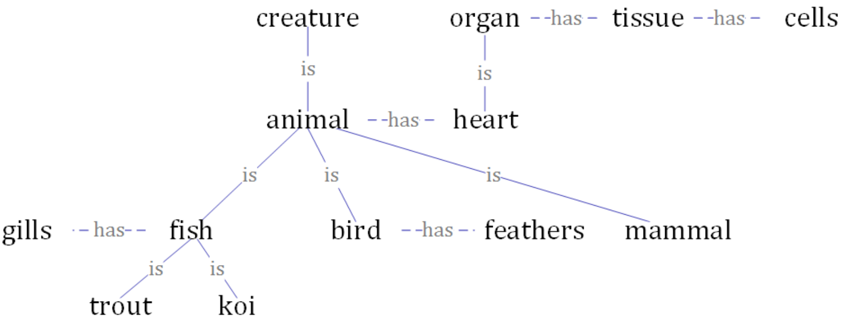
Theorem: A set of clauses F is not satisfiable iff the empty clause $\square$ can be derived from F by resolution.

- **Soundness**:
  (Proof by contradiction)
  Assume that $\square$ can be derived from F. If that is the case, two clauses $C_1 = \{L\}$ and $C_2 = \{\neg L\}$ must be contained in F. Because there exists no model for $L \wedge \neg L$, F is not satisfiable.
- **Refutation completeness**:
  (Proof by induction over the number n of atomar formulas in F)
  Assume that F is a set of formula which is not satisfiable. Because of the compactness theorem, it is enough to consider the case that a finite non-satisfiable subset of formula exists in F.
  To show: $\square$ is derived from F. (see e.g., Schöning)

- In general, there are many possibilities, to find two clauses, which are resolvable. Of the many alternatives, there are possibly only a few which help to derive the empty clause $\hookrightarrow$ combinatorial explosion!
- For feasible algorithms: use a resolution **strategy**
- E.g., exploit **subsumption** to keep the knowledge space, and therefore the search space, small.
  Remove all sentences which are subsumed (more special than) an existing sentence.
  If $P(x)$ is in the knowledge base, sentences as $P(A)$ or $P(A) \vee Q(B)$ can be removed.
- Well known efficient strategy: **SLD-Resolution** (*linear resolution with selection function for definite clauses*) (e.g. used in Prolog)

- **linear**: Use a sequence of clauses ($C_0 \ldots C_n$) starting with the negated assertion C_0 and ending with the empty clause C_n. Each C_i is generated as resolvent from C_{i-1} and a clause from the original set of axioms.
- **Selection function** (for the next literal which will be resolved) e.g. top-down-left-to-right in PROLOG; makes the strategy **incomplete**! ("user" must order clauses in a suitable way)
- **definite Horn clauses:** A Horn clause contains maximally one positive literal; a definite Horn clause contains exactly one positive literal (Prolog rule)

Example Prolog Program – Semantic Net



- Has a trout gills?
- Has a fish cells?

Example Prolog Program – Semantic Net

```
/* Example of a hierarchical semantic network in PROLOG */
/* ***************************************************** */

/* explicit isa and has links  */
/* facts                       */
isa(animal,creature).
isa(fish,animal).
isa(trout,fish).
isa(heart,organ).
hasprop(animal,heart).
hasprop(organ,tissue).
hasprop(tissue,cells).

/* -------------------------------------------------- */
/* Reasoning Rules                                     */

is(A,B) :- isa(A,B).                    /* Transitivity of is */
is(A,B) :- isa(A,C), is(C,B).

has(A,X) :- hasprop(A,X).               /* Transitivity of has */
has(X,Z) :- hasprop(X,Y), has(Y,Z).

has(A,X) :- isa(A,B), has(B,X).         /* Inheritance of has wrt is */
has(A,X) :- hasprop(A,Y), isa(Y,X).     /* Generalizing has wrt is */
```

	PROLOG	Logic	
Fact	`isa(fish,animal).` `isa(trout,fish).`	isa(Fish,Animal) isa(Trout,Fish)	Positive Literal
Rule	`is(X,Y) :-` `isa(X,Y).`	$\text{is}(x,y) \vee \neg\text{isa}(x,y)$	Definite Clause
	`is(X,Z) :-` `isa(X,Y), is(Y,Z).`	$\text{is}(x,z) \vee \neg\text{isa}(x,y)$ $\vee\neg\text{is}(y,z)$	
Query	`is(trout,animal).` `is(fish,X).`	$\neg\text{is}(\text{Trout}, \text{Animal})$ $\neg\text{is}(\text{Fish}, x)$	Assertion

- $:-$ denotes the "reversed" implication arrow.

```
is(X,Z) :- isa(X,Y), is(Y,Z).
```

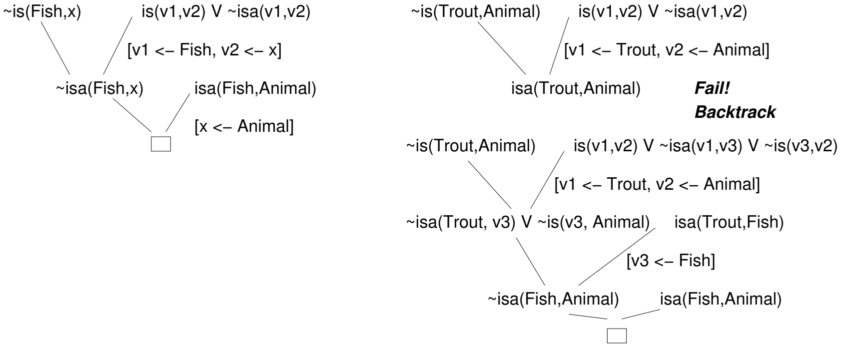
  is Prolog for:

$$
\begin{aligned}
\text{isa}(x,y) \land \text{is}(y,z) \rightarrow \text{is}(x,z) &\equiv \\
\neg(\text{isa}(x,y) \land \text{is}(y,z)) \lor \text{is}(x,z) &\equiv \\
\neg\text{isa}(x,y) \lor \neg\text{is}(y,z) \lor \text{is}(x,z) &
\end{aligned}
$$

- Variables which occur in the head of a clause are implicitly universally quantified. Variables which occur only in the body are existentially quantified.

$$\forall x \forall z \exists y : \neg\text{isa}(x,y) \lor \neg\text{is}(y,z) \lor \text{is}(x,z)$$

- Query: is(fish,X)
  (stands for $\exists x$ is(Fish, x))
- Negation of query: $\neg \exists x$: is(Fish, x) $\equiv \forall x$: $\neg$is(Fish, x)
- SLD-Resolution: *(extract)*



- When writing Prolog programs, one should be know how the interpreter is working (i.e., understand SLD-resolution)
- Sequence of clauses has influence whether an assertion which follows logically from a set of clauses can be derived!
- **Efficiency**: Facts before rules
- **Termination**: non-recursive rule before recursive

```
% Program                        % Query
isa(trout,fish).                 ? is(trout,animal).
isa(fish,animal).

is(X,Z) :- is(X,Y), isa(Y,Z).    is(trout,Y), isa(Y,animal)
is(X,Y) :- isa(X,Y).             is(trout,Y'),
                                 isa(Y',animal),
                                 isa(Y,animal)
                                 ...
```

```
X = 3+7.       % X = 3+7   unification, no evaluation!
X is 3+7.      % X = 10    true when Number is the value
               %           to which Expr evaluates
10 is 3+7.     % Yes
3+7 is 3+7.    % No        for equality use =:=

add(0, Y, Y).
add(succ(X), Y, succ(Z)) :- add(X, Y, Z).

fac(0, 1).
fac(N, V) :- N > 0, N1 is N-1, fac(N1,V1), V is N*V1.
                           % Not fac(N-1,V1)

append1(X,[],X).
append1([],Y,Y).
append1([H|T], Y, [H|Z]) :- append1(T,Y,Z).
```

https://www.swi-prolog.org/pldoc/doc_for?object=manual

Negation by Failure and Closed World Assumption

- Closed world assumption: Everything which is not known to be true or can be inferred to be true is false. (E.g.: It is false that a trout is a mammal since this information is neither given as a fact nor can be derived.)
- Negation by failure: every predicate that cannot be proved to be true is believed to be false.
- The predicate cut (written as !) inhibits backtracking.
- Underline represents a wild card. A variable only appearing in the head and not in the body of a rule results in a warning (singleton).

```
neg(A):- A, !, fail.
neg(_).
```

- PROLOG
- as *a basic* method for theorem proving (others: e.g. tableaux)
- **Question Answering Systems**

- Yes/No-Questions: Assertion/Query $mortal(s)$
- Query $is(trout, X)$ corresponds to "What is a trout?"
  The variable X is instantiated during resolution and the answer is "a fish".
- $buys(peter, john, X)$: "What does John buy from Peter?"
- $buys(peter, X, car)$: "Who buys a car from Peter?"

- Theorem provers typically are more general than Prolog: not only Horn clauses but full FOL; no interleaving of logic and control (i.e. ordering of formulas has no effect on result)
- Examples: Boyer-Moore (1979) theorem prover; OTTER, Isabelle
- Theorem provers for mathematics, for verification of hardware and software, for deductive program synthesis.

Forward- and Backward Chaining

- Rules (e.g. in Prolog) have the form:
  $Premises \rightarrow Conclusion$
- All rule-based systems (production systems, planners, inference systems) can be realized using either **forward-chaining** or **backward-chaining** algorithms.
- Forward chaining: Add a new fact to the knowledge base and derive all consequences (data-driven)
- Backward chaining: Start with a goal to be proved, find implication sentences that would allow to conclude the goal, attempt to prove the premises, etc.
- Well known example for a **backward reasoning expert system**: **MYCIN** (diagnosis of bacterial infections)

- Propositional logic and FOL are **classical logics**.
- Classical logic is **bivalent and monotonic**:
  There are only two truth values "true" and "false".
  Because of the *tertium non datur*, derived conclusions cannot be changed by new facts or conclusions (vs. **multi-valued** and **non-monotonic** logics).
- In classical logic, "everything" follows from a contradiction (*ex falso quod libet*).
  A theorem can be proven by contradiction.
  In contrast, in intuitionistic logic, all proofs must be *constructive*!

Logic Calculi in AI

- Variants of logic calculi are part of many AI systems
- Logic and logical inference is the base of most types of knowledge representation formalisms (e.g. description logics)
- Most knowledge-based systems (e.g. expert systems) are relying on some type of deductive inference mechanism
- Often, classical logic is not adequate: non-monotonic, probabilistic or fuzzy approaches
- Extensions of classical logic for dealing with time or believe: Modal Logic (e.g., BDI-Logic for Multi-agent Systems)

Basic Types of Inference: Deduction

(Charles Peirce)

- **Deduction**: Derive a conclusion from given axioms ("knowledge") and facts ("observations").

Example

(**axiom**)	*All humans are mortal.*
(**fact/premise**)	*Socrates is a human.*
(**conclusion**)	*Therefore, it follows that Socrates is mortal.*

- The conclusion can be derived by applying the *modus ponens* inference rule (Aristotelian/propositional logic).

- **Induction**: Derive a general rule (axiom) from background knowledge and observations.

Example

(**background knowledge**)	*Socrates is a human.*
(**observation/example**)	*Socrates is mortal.*
(**generalization**)	*Therefore, I hypothesize that all humans are mortal.*

- Induction means to infer (unsure) generalized knowledge from example observations.
- Induction is *the* inference mechanism for learning! (see lesson on Machine Learning)
- Analogy is a special kind of induction.

- **Abduction**: From a known axiom (theory) and some observation, derive a premise.

Example

(**theory**)	*All humans are mortal.*
(**observation**)	*Socrates is mortal.*
(**diagnosis**)	*Therefore,* *Socrates must have been a human.*

- Abduction is typical for diagnostic systems/expert systems. (*It is also the preferred reasoning method of Sherlock Holmes.*)
- **Simple medical diagnosis**:
  If one has the flue, one has moderate fewer.
  Patient X has moderate fewer.
  Therefore, he has the flue.

Summary

- Resolution is defined for clausal form.
- Logical formula can be rewritten in conjunctive normal form from which a set of clauses can be generated.
- Rewriting into clausal form relies on equivalence rules, Skolemization is not an equivalence transformation but a formula and its Skolemization are equivalent w.r.t. satisfiability.
- In FOL, identify of formulas can be established by restricting their scope: The most general unifier is defined as the minimal set of substitution of variables by terms to make two formulas equal.
- Resolution is a proof by contradiction.
- For implementing resolution, a strategy to select clauses for refutation is necessary.
- Prolog is a resolution prover based on SLD-resolution.
- Deduction is the only type of inference where correctness of derivations (conclusions) can be guaranteed.